LNCS 7965
ARCoSS

Fedor V. Fomin
Rūsiņš Freivalds
Marta Kwiatkowska
David Peleg (Eds.)

Automata, Languages, and Programming

40th International Colloquium, ICALP 2013
Riga, Latvia, July 2013
Proceedings, Part I

Springer

Lecture Notes in Computer Science 7965

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

Advanced Research in Computing and Software Science
Subline of Lectures Notes in Computer Science

Fedor V. Fomin Rūsiņš Freivalds
Marta Kwiatkowska David Peleg (Eds.)

Automata, Languages, and Programming

40th International Colloquium, ICALP 2013
Riga, Latvia, July 8-12, 2013
Proceedings, Part I

Springer

Volume Editors

Fedor V. Fomin
University of Bergen, Department of Informatics
Postboks 7803, 5020 Bergen, Norway
E-mail: fomin@ii.uib.no

Rūsiņš Freivalds
University of Latvia, Faculty of Computing
Raina bulv. 19, 1586 Riga, Latvia
E-mail: rusins.freivalds@lu.lv

Marta Kwiatkowska
University of Oxford, Department of Computer Science
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
E-mail: marta.kwiatkowska@cs.ox.ac.uk

David Peleg
Weizmann Institute of Science, Faculty of Mathematics and Computer Science
POB 26, 76100 Rehovot, Israel
E-mail: david.peleg@weizmann.ac.il

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-39205-4 e-ISBN 978-3-642-39206-1
DOI 10.1007/978-3-642-39206-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013941217

CR Subject Classification (1998): F.2, F.1, C.2, H.3-4, G.2, I.2, I.3.5, E.1

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

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

Printed on acid-free paper

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

Preface

ICALP, the International Colloquium on Automata, Languages and Programming, is arguably the most well-known series of scientific conferences on Theoretical Computer Science in Europe. The first ICALP was held in Paris, France, during July 3–7, 1972, with 51 talks. The same year EATCS, the European Association for Theoretical Computer Science, was established. Since then ICALP has been the flagship conference of EATCS.

ICALP 2013 was the 40th conference in this series (there was no ICALP in 1973). For the first time, ICALP entered the territory of the former Soviet Union. It was held in Riga, Latvia, during on July 8–12, 2013, in the University of Latvia. This year the program of ICALP was organized in three tracks: Track A (Algorithms, Complexity and Games), Track B (Logic, Semantics, Automata and Theory of Programming), and Track C (Foundations of Networked Computation: Models, Algorithms and Information Management).

In response to the Call for Papers, 436 papers were submitted; 14 papers were later withdrawn. The three Program Committees worked hard to select 71 papers for Track A (out of 249 papers submitted), 33 papers for Track B (out of 113 papers), and 20 papers for Track C (out of 60 papers). The average acceptance rate was 29%. The selection was based on originality, quality, and relevance to theoretical computer science. The quality of the submitted papers was very high indeed. The Program Committees acknowledge that many rejected papers deserved publication but regrettably it was impossible to extend the conference beyond 5 days.

The best paper awards were given to Mark Bun and Justin Thaler for their paper "Dual Lower Bounds for Approximate Degree and Markov-Bernstein Inequalities" (in Track A), to John Fearnley and Marcin Jurdziński for the paper "Reachability in Two-Clock Timed Automata is PSPACE-complete" (in Track B), and to Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pająk, and Przemysław Uznański for the paper "Fast Collaborative Graph Exploration" (in Track C). The best student paper awards were given to Radu Curticapean for the paper "Counting matchings of size k is $\#\mathsf{W}[1]$-hard" (in Track A) and to Nicolas Basset for the paper "A maximal entropy stochastic process for a timed automaton" (in Track B).

ICALP 2013 contained a special EATCS lecture on the occasion of the 40th ICALP given by:

– Jon Kleinberg, Cornell University

Invited talks were delivered by:

– Susanne Albers, Humboldt University
– Orna Kupferman, Hebrew University
– Dániel Marx, Hungarian Academy of Sciences

– Paul Spirakis, University of Patras
– Peter Widmayer, ETH Zürich

The main conference was preceded by a series of workshops on Sunday July 7, 2013 (i.e., one day before ICALP 2013). The list of workshops consisted of:

– Workshop on Automata, Logic, Formal languages, and Algebra (ALFA 2013)
– International Workshop on Approximation, Parameterized and Exact Algorithms (APEX 2013)
– Quantitative Models: Expressiveness and Analysis
– Foundations of Network Science (FONES)
– Learning Theory and Complexity
– 7th Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC 2013)
– Workshop on Quantum and Classical Complexity

We sincerely thank our sponsors, members of the committees, referees, and the many colleagues who anonymously spent much time and effort to make ICALP 2013 happen.

May 2013

Fedor V. Fomin
Rūsiņš Freivalds
Marta Kwiatkowska
David Peleg

Organization

Program Committee

Track A

Andris Ambainis	University of Latvia, Latvia
Edith Elkind	Nanyang Technological University, Singapore
Leah Epstein	University of Haifa, Israel
Rolf Fagerberg	University of Southern Denmark, Denmark
Fedor Fomin	University of Bergen, Norway (Chair)
Pierre Fraigniaud	CNRS and University Paris Diderot, France
Fabrizio Grandoni	Dalle Molle Institute, Switzerland
Joachim Gudmundsson	University of Sydney, Australia
Kazuo Iwama	Kyoto University, Japan
Valentine Kabanets	Simon Fraser University, Canada
Stavros Kolliopoulos	National and Kapodistrian University of Athens, Greece
Daniel Král'	University of Warwick, UK
Daniel Lokshtanov	University of California, San Diego, USA
Konstantin Makarychev	Microsoft Research, Redmond, USA
Peter Bro Miltersen	Aarhus University, Denmark
Ilan Newman	University of Haifa, Israel
Konstantinos Panagiotou	Ludwig Maximilians University, Munich, Germany
Alexander Razborov	University of Chicago, USA
Saket Saurabh	The Institute of Mathematical Sciences, India
David Steurer	Microsoft Research, New England, USA
Kunal Talwar	Microsoft Research, Silicon Valley, USA
Dimitrios Thilikos	National and Kapodistrian University of Athens, Greece
Virginia Vassilevska Williams	University of California, Berkeley, and Stanford, USA
Gerhard Woeginger	Eindhoven University of Technology, The Netherlands

Track B

Christel Baier	TU Dresden, Germany
Chiara Bodei	University of Pisa, Italy
Mikołaj Bojańczyk	University of Warsaw, Poland
Patricia Bouyer-Decitre	CNRS/ENS Cachan, France
Vassilis Christophides	University of Crete, Greece
Yuxin Deng	Shanghai Jiao-Tong University, China
Marcelo Fiore	University of Cambridge, UK

Patrice Godefroid	Microsoft Research, Redmond, USA
Andy Gordon	MSR Cambridge and University of Edinburgh, UK
Alexey Gotsman	Madrid Institute for Advanced Studies (IMDEA), Spain
Masami Hagiya	University of Tokyo, Japan
Michael Huth	Imperial College London, UK
Stephan Kreutzer	TU Berlin, Germany
Antonín Kučera	Masaryk University, Brno, Czech Republic
Viktor Kuncak	EPFL, Lausanne, Switzerland
Marta Kwiatkowska	University of Oxford, UK (Chair)
Leonid Libkin	University of Edinburgh, UK
Rupak Majumdar	Max Planck Institute, Kaiserslautern, Germany
Jerzy Marcinkowski	University of Wroclaw, Poland
Annabelle McIver	Macquarie University, Australia
Catuscia Palamidessi	INRIA Saclay and LIX, Ecole Polytechnique, Paris, France
Frank Pfenning	Carnegie Mellon University, USA
André Platzer	Carnegie Mellon University, USA
Jean-François Raskin	UL Brussels, Belgium
Jan Rutten	CWI and Radboud University Nijmegen, The Netherlands
Peter Selinger	Dalhousie University, Canada
Andreas Winter	University of Bristol, UK

Track C

James Aspnes	Yale Univerity, USA
Ioannis Caragiannis	University of Patras, Greece
Xavier Defago	JAIST, Japan
Josep Diaz	UPC, Barcelona, Spain
Stefan Dobrev	Slovak Academy of Sciences, Bratislava, Slovak Republic
Michele Flammini	University of L'Aquila, Italy
Leszek Gąsieniec	University of Liverpool, UK
Cyril Gavoille	Univerity of Bordeaux, France
David Kempe	University of Southern California, USA
Valerie King	University of Victoria, Canada
Amos Korman	CNRS, Paris, France
Miroslaw Kutylowski	Wroclaw University of Technology, Poland
Dahlia Malkhi	Microsoft Research, Silicon Valley, USA
Luca Moscardelli	University of Chieti, Pescara, Italy
Thomas Moscibroda	Microsoft Research Asia and Tsinghua University, China
Marina Papatriantafilou	Chalmers University of Technology, Goteborg, Sweden

David Peleg	Weizmann Institute of Science, Israel (Chair)
Yvonne Anne Pignolet	ETH Zurich, Switzerland
Sergio Rajsbaum	UNAM, Mexico
Liam Roditty	Bar-Ilan University, Israel
José Rolim	University of Geneva, Switzerland
Christian Scheideler	Technical University of Munich, Germany
Jennifer Welch	Texas A&M University, USA

Organizing Committee

(all from University of Latvia, Latvia)

Andris Ambainis	
Kaspars Balodis	
Juris Borzovs	(Organizing Chair)
Rūsiņš Freivalds	(Conference Chair)
Marats Golovkins	
Nikolay Nahimov	
Jeļena Poļakova	
Alexander Rivosh	
Agnis Škuškovniks	(Organizing Deputy Chair)
Juris Smotrovs	
Abuzer Yakaryılmaz	

Sponsoring Institutions

QuBalt
University of Latvia

Additional Reviewers

Aaronson, Scott	Arvind, V.	Barman, Siddharth
Aceto, Luca	Askalidis, Georgios	Barto, Libor
Adamaszek, Anna	Atserias, Albert	Belovs, Aleksandrs
Afshani, Peyman	Aumüller, Martin	Bendlin, Rikke
Agrawal, Manindra	Avigdor-Elgrabli, Noa	Benoit, Anne
Ahn, Kook Jin	Avis, David	Benzaken, Veronique
Aichholzer, Oswin	Badanidiyuru, Ashwinkumar	Berman, Itay
Albers, Susanne	Bae, Sang Won	Bertrand, Nathalie
Allouche, Jean-Paul	Balmau, Oana	Berwanger, Dietmar
Alur, Rajeev	Bampis, Evripidis	Bianchi, Giuseppe
Alvarez, Carme	Bansal, Nikhil	Biedl, Therese
Amano, Kazuyuki	Barcelo, Pablo	Bilò, Davide
Andoni, Alexandr		Bilò, Vittorio

Björklund, Andreas
Björklund, Henrik
Blais, Eric
Bläser, Markus
Blum, William
Bodirsky, Manuel
Bodlaender, Hans L.
Bohy, Aaron
Boker, Udi
Bollig, Benedikt
Bonnet, François
Bonsangue, Marcello
Bortolussi, Luca
Boularias, Abdeslam
Bourhis, Pierre
Boyar, Joan
Boyle, Elette
Brandes, Philipp
Brandstadt, Andreas
Braverman, Mark
Braverman, Vladimir
Brazdil, Tomas
Bringmann, Karl
Brodal, Gerth Stølting
Brody, Joshua
Bulatov, Andrei
Byrka, Jarek
Cabello, Sergio
Cachin, Christian
Carbone, Marco
Carton, Olivier
Cerny, Pavol
Cervesato, Iliano
Chakrabarty, Deeparnab
Chakraborty, Sourav
Chaloulos, Konstantinos
Chan, Siu On
Charatonik, Witold
Chase, Melissa
Chattopadhyay, Arkadev
Chávez, Edgar
Chawla, Sanjay
Chechik, Shiri
Chen, Jie
Chen, Kevin
Chen, Ning
Chen, Taolue
Chen, Xin
Chester, Sean
Chistikov, Dmitry
Chitnis, Rajesh
Chmelík, Martin
Chrobak, Marek
Cicalese, Ferdinando
Clark, Alex
Conchinha, Bruno
Cormode, Graham
Corradini, Andrea
Crescenzi, Pierluigi
Currie, James
Cygan, Marek
Czerwiński, Wojciech
Czumaj, Artur
Dal Lago, Ugo
Datta, Samir
Daum, Marcus
Davies, Rowan
Dawar, Anuj
de Gouw, Stijn
de Groote, Philippe
de Haan, Robert
de Wolf, Ronald
Dell, Holger
Deshpande, Amit
Devanur, Nikhil
Devroye, Luc
Díaz-Báñez, José-Miguel
Dinitz, Michael
Dittmann, Christoph
Doerr, Benjamin
Doerr, Carola
Dorrigiv, Reza
Dósa, György
Doty, David
Doyen, Laurent
Dregi, Markus
Drucker, Andrew
Dräger, Klaus
Ducas, Léo
Dunkelman, Orr
Ďuriš, Pavol
Dutta, Kunal
Dvořák, Zdeněk
Dziembowski, Stefan
Ebtekar, Aram
Eidenbenz, Raphael
Eikel, Martina
Eisenbrand, Friedrich
Elbassioni, Khaled
Elkin, Michael
Emek, Yuval
Ene, Alina
Englert, Matthias
Eppstein, David
Erlebach, Thomas
Escoffier, Bruno
Esmaeilsabzali, Shahram
Faenza, Yuri
Fanelli, Angelo
Faust, Sebastian
Favrholdt, Lene
Fehnker, Ansgar
Feige, Uri
Feldman, Moran
Feng, Yuan
Fenner, Stephen
Feret, Jérôme
Ferrari, Gianluigi
Fertin, Guillaume
Fijalkow, Nathanaël
Filmus, Yuval
Fineman, Jeremy
Fischer, Eldar
Fischlin, Marc
Fisher, Jasmin
Floderus, Peter
Fountoulakis, Nikolaos
Frandsen, Gudmund
Frati, Fabrizio
Friedrich, Tobias
Frieze, Alan
Friggstad, Zachary
Fu, Hongfei
Függer, Matthias
Fujioka, Kaoru

Funke, Stefan
Gagie, Travis
Gairing, Martin
Galletta, Letterio
Gamarnik, David
Ganesh, Vijay
Ganian, Robert
Garg, Jugal
Gärtner, Bernd
Gasarch, William
Gaspers, Serge
Gauwin, Olivier
Gavinsky, Dmitry
Gay, Simon
Geeraerts, Gilles
Gemulla, Rainer
Georgiadis, Giorgos
Ghorbal, Khalil
Giannopoulos, Panos
Gimbert, Hugo
Giotis, Ioannis
Gmyr, Robert
Goasdoué, François
Gogacz, Tomasz
Golas, Ulrike
Goldberg, Andrew
Goldhirsh, Yonatan
Göller, Stefan
Golovach, Petr
Goncharov, Sergey
Gopalan, Parikshit
Gorbunov, Sergey
Gorry, Thomas
Gottlieb, Lee-Ad
Gourvès, Laurent
Goyal, Navin
Graça, Daniel
Grenet, Bruno
Guo, Alan
Gupta, Anupam
Gupta, Sushmita
Gurvich, Vladimir
Gutwenger, Carsten
Habib, Michel
Hadzilacos, Vassos
Hahn, Ernst Moritz
Hähnle, Nicolai
Hajiaghayi, Mohammadtaghi
Halldórsson, Magnús M.
Hansen, Kristoffer Arnsfelt
Hanzlik, Lucjan
Harsha, Prahladh
Hassin, Refael
Hasuo, Ichiro
Hayman, Jonathan
He, Chaodong
He, Shan
Heggernes, Pınar
Heindel, Tobias
Hellwig, Matthias
Hirai, Yoichi
Hitchcock, John M.
Hliněný, Petr
Hoeksma, Ruben
Höfner, Peter
Hon, Wing-Kai
Horiyama, Takashi
Huang, Chien-Chung
Huber, Anna
Hüllmann, Martina
Hur, Chung-Kil
Ibsen-Jensen, Rasmus
Ilcinkas, David
Im, Sungjin
Imai, Katsunobu
Imreh, Csanád
Indyk, Piotr
Ishii, Toshimasa
Ito, Hiro
Ito, Tsuyoshi
Itoh, Toshiya
Iván, Szabolcs
Iwamoto, Chuzo
Jager, Tibor
Jain, Rahul
Jančar, Petr
Jansen, Bart
Jansen, Klaus
Jarry, Aubin
Jeż, Artur
Jeż, Łukasz
Jonsson, Bengt
Jordán, Tibor
Jurdziński, Tomasz
Jürjens, Jan
Kaibel, Volker
Kamiński, Marcin
Kammer, Frank
Kanellopoulos, Panagiotis
Kannan, Sampath
Kaplan, Haim
Kapralov, Michael
Karakostas, George
Karanikolas, Nikos
Karavelas, Menelaos I.
Karhumäki, Juhani
Kärkkäinen, Juha
Kartzow, Alexander
Kawahara, Jun
Kayal, Neeraj
Keller, Nathan
Keller, Orgad
Kellerer, Hans
Kemper, Stephanie
Kerenidis, Iordanis
Khot, Subhash
Kiayias, Aggelos
Kiefer, Stefan
Kik, Marcin
Kim, Eun Jung
Kissinger, Alexander
Klauck, Hartmut
Klein, Karsten
Kliemann, Lasse
Klíma, Ondřej
Klin, Bartek
Klonowski, Marek
Kluczniak, Kamil
Kniesburges, Sebastian
Kobayashi, Koji
Kobayashi, Yusuke

Koenemann, Jochen
Kolay, Sudeshna
Kolesnikov, Vladimir
Kollias, Kostas
Komm, Dennis
Komusiewicz, Christian
Konečný, Filip
Konrad, Christian
Kopczynski, Eryk
Kopelowitz, Tsvi
Kopparty, Swastik
Kortsarz, Guy
Kötzing, Timo
Koucký, Michal
Koutavas, Vasileios
Koutsopoulos, Andreas
Kowalik, Lukasz
Koza, Michał
Kozen, Dexter
Královič, Rastislav
Královič, Richard
Kratsch, Stefan
Krčál, Jan
Křetínský, Jan
Krishnaswamy, Ravishankar
Kristensen, Lars
Kritikos, Kyriakos
Krivine, Jean
Krokhin, Andrei
Krumke, Sven
Krysta, Piotr
Krzywiecki, Łukasz
Kunc, Michal
Kuperberg, Denis
Kupferman, Orna
Kurz, Alexander
Kutten, Shay
Kutzkov, Konstantin
Kyropoulou, Maria
Lachish, Oded
Lanese, Ivan
Lauks-Dutka, Anna
Laurent, Dominique
Lavi, Ron
Lawson, Mark V.
Le Gall, François
Lerays, Virginie
Leroux, Jérôme
Leucci, Stefano
Levin, Asaf
Lewenstein, Moshe
Lewi, Kevin
Li, Jian
Li, Shi
Liang, Guanfeng
Lin, Anthony Widjaja
Lingas, Andrzej
Linji, Yang
Loff, Bruno
Lohrey, Markus
Loos, Sarah
López-Ortiz, Alejandro
Lösch, Steffen
Lotker, Zvi
Lu, Pinyan
Lücke, Dominik
Lundell, Eva-Marta
Maffray, Frédéric
Mahajan, Meena
Makarychev, Yury
Malgouyres, Rémy
Mantaci, Roberto
Manthey, Bodo
Märcker, Steffen
Markey, Nicolas
Martin, Russell
Martins, João G.
Marx, Dániel
Mathieson, Luke
Matsuda, Takahiro
Matulef, Kevin
Mauro, Jacopo
Mazumdar, Arya
McAuley, Julian
McGregor, Andrew
McKenzie, Pierre
Meinicke, Larissa
Meister, Daniel
Mereacre, Alexandru
Mertzios, George B.
Messner, Jochen
Mestre, Julian
Meyer auf der Heide, Friedhelm
Mezzetti, Gianluca
Micciancio, Daniele
Michaliszyn, Jakub
Misra, Neeldhara
Misra, Pranabendu
Mitsch, Stefan
Mittal, Shashi
Miyazaki, Shuichi
Mokhov, Andrey
Møller, Anders
Monaco, Gianpiero
Montanari, Alessandro
Montgomery, Hart
Morgan, Carroll
Morin, Pat
Moseley, Ben
Movahedi, Mahnush
Moysoglou, Yannis
Mozes, Shay
Mucha, Marcin
Müller, Tobias
Mulmuley, Ketan
Muscholl, Anca
Myers, Robert
Myers, Steven
Nagarajan, Viswanath
Nanongkai, Danupon
Naor, Moni
Narayan, Arjun
Navarra, Alfredo
Navarro, Gonzalo
Nebel, Markus
Nederlof, Jesper
Nehama, Ilan
Nelson, Jelani
Ngo, Hung
Nguyen, Huy
Nguyen, Phong
Nicholson, Pat
Nishimura, Harumichi

Nonner, Tim
Nordström, Jakob
Novotný, Petr
Nowotka, Dirk
Nutov, Zeev
O'Donnell, Ryan
Obdrzalek, Jan
Ogierman, Adrian
Okamoto, Yoshio
Okawa, Satoshi
Oliehoek, Frans
Ollinger, Nicolas
Ölveczky, Peter
Onak, Krzysztof
Ono, Hirotaka
Ostrovsky, Rafail
Otachi, Yota
Ott, Sebastian
Oualhadj, Youssouf
Oveis Gharan, Shayan
Paes Leme, Renato
Pagh, Rasmus
Paluch, Katarzyna
Pandey, Omkant
Pandurangan, Gopal
Pandya, Paritosh
Panigrahi, Debmalya
Pankratov, Denis
Panolan, Fahad
Paparas, Dimitris
Pardubská, Dana
Pascual, Fanny
Pasechnik, Dmitrii
Passmore, Grant
Paul, Christophe
Paulusma, Daniël
Peikert, Chris
Perdrix, Simon
Petig, Thomas
Pferschy, Ulrich
Phillips, Jeff
Picaronny, Claudine
Pierrakos, George
Pietrzak, Krzysztof
Piliouras, Georgios
Pilipczuk, Marcin
Pilipczuk, Michał
Pinkas, Benny
Piskac, Ruzica
Polák, Libor
Policriti, Alberto
Porat, Ely
Pottier, François
Pouly, Amaury
Prabhakar, Pavithra
Prabhakaran, Manoj M.
Pratikakis, Polyvios
Pratt-Hartmann, Ian
Price, Eric
Puglisi, Simon
Quaas, Karin
Rabehaja, Tahiry
Rabinovich, Roman
Rabinovich, Yuri
Räcke, Harald
Radzik, Tomasz
Raghunathan, Ananth
Rajaraman, Rajmohan
Raman, Venkatesh
Ramanujan, M.S.
Ranzato, Francesco
Raptopoulos, Christoforos
Ravi, R.
Rawitz, Dror
Raz, Ran
Razenshteyn, Ilya
Regev, Oded
Řehák, Vojtěch
Rémy, Didier
Restivo, Antonio
Rettinger, Robert
Reutenauer, Christophe
Reyzin, Lev
Richerby, David
Rigo, Michel
Röglin, Heiko
Ron, Dana
Rosén, Adi
Rotbart, Noy
Rote, Günter
Roth, Aaron
Rouselakis, Yannis
Russo, Claudio
Rusu, Irena
Sadakane, Kunihiko
Saei, Reza
Saha, Barna
Sakai, Yoshifumi
Sakurada, Hideki
Salem, Iosif
Salinger, Alejandro
Sanders, Peter
Sankowski, Piotr
Sankur, Ocan
Santhanam, Rahul
Saptharishi, Ramprasad
Saraf, Shubhangi
Sassolas, Mathieu
Satti, Srinivasa Rao
Sau, Ignasi
Sauerwald, Thomas
Sawa, Zdeněk
Sağlam, Mert
Schäfer, Andreas
Schindelhauer, Christian
Schmid, Stefan
Schneider, Johannes
Schnoebelen, Philippe
Schröder, Matthias
Schubert, Aleksy
Schumacher, André
Schwartz, Roy
Schweitzer, Pascal
Schwentick, Thomas
Scott, Elizabeth
Sebő, András
Sedgewick, Bob
Segev, Danny
Seki, Shinnosuke
Sénizergues, Géraud
Sereni, Jean-Sébastien
Serna, Maria
Seshadhri, C.
Seto, Kazuhisa

Severini, Simone
Sgall, Jiří
Shapira, Asaf
Shavit, Nir
Sherstov, Alexander
Shi, Yaoyun
Shpilka, Amir
Siddharthan, Rahul
Sidiropoulos, Anastasios
Silva, Alexandra
Silveira, Rodrigo
Silvestri, Riccardo
Simmons, Robert
Simon, Hans
Singh, Mohit
Skrzypczak, Michał
Sloane, Tony
Sly, Allan
Soltanolkotabi, Mahdi
Soto, José A.
Špalek, Robert
Spirakis, Paul
Srba, Jiří
Srinathan, Kannan
Srinivasan, Srikanth
Stapleton, Gem
Staton, Sam
Stauffer, Alexandre
Stenman, Jari
Storandt, Sabine
Strauss, Martin
Strichman, Ofer
Strothmann, Thim
Subramani, K.
Suri, Subhash
Sutner, Klaus
Svensson, Ola
Svitkina, Zoya
Szegedy, Mario
Sznajder, Nathalie
Talmage, Edward
Tamir, Tami
Tan, Li-Yang
Tanabe, Yoshinori
Tanaka, Keisuke
Tancer, Martin
Tang, Bangsheng
Tassa, Tamir
Telikepalli, Kavitha
Tentes, Aris
Terauchi, Tachio
Tessaro, Stefano
Thachuk, Chris
Thaler, Justin
Thapper, Johan
Tirthapura, Srikanta
Todinca, Ioan
Toninho, Bernardo
Tonoyan, Tigran
Torenvliet, Leen
Toruńczyk, Szymon
Torán, Jacobo
Triandopoulos, Nikos
Tudor, Valentin
Tůma, Vojtěch
Tzameret, Iddo
Tzevelekos, Nikos
Ueckerdt, Torsten
Uehara, Ryuhei
Ueno, Kenya
Valencia, Frank
Valiant, Gregory
van Dam, Wim
van Den Heuvel, Jan
van Leeuwen, Erik Jan
van Melkebeek, Dieter
van Rooij, Johan M.M.
van Stee, Rob
Varacca, Daniele
Variyam, Vinod
Vatshelle, Martin
Veanes, Margus
Végh, László
Vereshchagin, Nikolay
Vergnaud, Damien
Verschae, José
Viderman, Michael
Vidick, Thomas
Vijayaraghavan, Aravindan
Villanger, Yngve
Visconti, Ivan
Vishnoi, Nisheeth
Vondrák, Jan
Vrgoč, Domagoj
Vrt'o, Imrich
Walukiewicz, Igor
Wan, Andrew
Wang, Haitao
Wang, Yajun
Watanabe, Osamu
Watrous, John
Watson, Thomas
Weimann, Oren
Weinstein, Omri
Wieczorek, Piotr
Wiese, Andreas
Wiesner, Karoline
Williams, Ryan
Wirth, Tony
Wojtczak, Dominik
Wollan, Paul
Wong, Prudence W.H.
Wood, David
Wootters, Mary
Worrell, James
Wulff-Nilsen, Christian
Xiao, David
Yamamoto, Mitsuharu
Yaroslavtsev, Grigory
Yehudayof, Amir
Yoshida, Yuichi
Zadimoghaddam, Morteza
Zawadzki, Erik
Zetzsche, Georg
Zhang, Qin
Zikas, Vassilis
Zimmermann, Martin
Živný, Stanislav
Zwick, Uri

Table of Contents – Part I

Track A – Algorithms, Complexity and Games

Table of Contents – Part II

EATCS Lecture

Invited Talks

Track B – Logic, Semantics, Automata and Theory of Programming

Track C – Foundations of Networked Computation

Exact Weight Subgraphs and the k-Sum Conjecture

Amir Abboud* and Kevin Lewi

Computer Science Department, Stanford University
{abboud,klewi}@cs.stanford.edu

Abstract. We consider the EXACT-WEIGHT-H problem of finding a (not necessarily induced) subgraph H of weight 0 in an edge-weighted graph G. We show that for every H, the complexity of this problem is strongly related to that of the infamous k-SUM problem. In particular, we show that under the k-SUM Conjecture, we can achieve tight upper and lower bounds for the EXACT-WEIGHT-H problem for various subgraphs H such as matching, star, path, and cycle.

One interesting consequence is that improving on the $O(n^3)$ upper bound for EXACT-WEIGHT-4-PATH or EXACT-WEIGHT-5-PATH will imply improved algorithms for 3-SUM, 5-SUM, ALL-PAIRS SHORTEST PATHS and other fundamental problems. This is in sharp contrast to the minimum-weight and (unweighted) detection versions, which can be solved easily in time $O(n^2)$. We also show that a faster algorithm for any of the following three problems would yield faster algorithms for the others: 3-SUM, EXACT-WEIGHT-3-MATCHING, and EXACT-WEIGHT-3-STAR.

1 Introduction

Two fundamental problems that have been extensively studied separately by different research communities for many years are the k-SUM problem and the problem of finding subgraphs of a certain form in a graph. We investigate the relationships between these problems and show tight connections between k-SUM and the "exact-weight" version of the subgraph finding problem.

The k-SUM problem is the parameterized version of the well known NP-complete problem SUBSET-SUM, and it asks if in a set of n integers, there is a subset of size k whose integers sum to 0. This problem can be solved easily in time $O(n^{\lceil k/2 \rceil})$, and Baran, Demaine, and Pătraşcu [3] show how the 3-SUM problem can be solved in time $O(n^2/\log^2 n)$ using certain hashing techniques. However, it has been a longstanding open problem to solve k-SUM for *some* k in time $O(n^{\lceil k/2 \rceil - \epsilon})$ for some $\epsilon > 0$. In certain restricted models of computation, an $\Omega(n^{\lceil k/2 \rceil})$ lower bound has been established initially by Erickson [7] and later generalized by Ailon and Chazelle [1], and recently, Pătraşcu and Williams [16] show that $n^{o(k)}$ time algorithms for all k would refute the Exponential Time Hypothesis. The literature seems to suggest the following hypothesis, which we call the k-SUM Conjecture:

* This work was done while the author was supported by NSF grant CCF-1212372.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 1–12, 2013.

Conjecture 1 [The k-SUM Conjecture]. *There does not exist a $k \geq 2$, an $\varepsilon > 0$, and a randomized algorithm that succeeds (with high probability) in solving k-SUM in time $O(n^{\lceil \frac{k}{2} \rceil - \varepsilon})$.*

The presumed difficulty of solving k-SUM in time $O(n^{\lceil k/2 \rceil - \varepsilon})$ for any $\varepsilon > 0$ has been the basis of many conditional lower bounds for problems in computational geometry. The $k = 3$ case has received even more attention, and proving 3-SUM-hardness has become common practice in the computational geometry literature. In a recent line of work, Pătraşcu [15], Vassilevska and Williams [17], and Jafargholi and Viola [11] show conditional hardness based on 3-SUM for problems in data structures and triangle problems in graphs.

The problem of determining whether a weighted or unweighted n-node graph has a subgraph that is isomorphic to a fixed k node graph H with some properties has been well-studied in the past [14,12,6]. There has been much work on detection and counting copies of H in graphs, the problem of listing all such copies of H, finding the minimum-weight copy of H, etc. [17,13]. Considering these problems for restricted types of subgraphs H has received further attention, such as for subgraphs H with large indepedent sets, or with bounded treewidth, and various other structures [17,18,13,8]. In this work, we focus on the following subgraph finding problem.

Definition 1 (The EXACT-WEIGHT-H Problem). *Given an edge-weighted graph G, does there exist a (not necessarily induced) subgraph isomorphic to H such that the sum of its edge weights equals a given target value t?*[1]

No non-trivial algorithms were known for this problem. Theoretical evidence for the hardness of this problem was given in [17], where the authors prove that for any H of size k, an algorithm for the exact-weight problem can give an algorithm for the minimum-weight problem with an overhead that is only $O(2^k \cdot \log M)$, when the weights of the edges are integers in the range $[-M, M]$. They also show that improving on the trivial $O(n^3)$ upper bound for EXACT-WEIGHT-3-CLIQUE to $O(n^{3-\varepsilon})$ for any $\varepsilon > 0$ would not only imply an $\tilde{O}(n^{3-\varepsilon})$ algorithm[2] for the minimum-weight 3-CLIQUE problem, which from [19] is in turn known to imply faster algorithms for the canonical ALL-PAIRS SHORTEST PATHS problem, but also an $O(n^{2-\varepsilon'})$ upper bound for the 3-SUM problem, for some $\varepsilon' > 0$. They give additional evidence for the hardness of the exact-weight problem by proving that faster than trivial algorithms for the k-CLIQUE problem will break certain cryptographic assumptions.

Aside from the aforementioned reduction from 3-SUM to EXACT-WEIGHT-3-CLIQUE, few other connections between k-SUM and subgraph problems were known. The standard reduction from Downey and Fellows [5] gives a way to reduce the unweighted k-CLIQUE detection problem to $\binom{k}{2}$-SUM on n^2 numbers. Also, in [15] and [11], strong connections were shown between the 3-SUM problem (or, the similar 3-XOR problem) and listing triangles in unweighted graphs.

[1] We can assume, without loss of generality, that the target value is always 0 and that H has no isolated vertices.

[2] In our bounds, k is treated as a constant. The notation $\tilde{O}(f(n))$ will hide $\mathsf{polylog}(n)$ factors.

1.1 Our Results

In this work, we study the exact-weight subgraph problem and its connections to k-SUM. We show three types of reductions: k-SUM to subgraph problems, subgraphs to other subgraphs, and subgraphs to k-SUM. These results give conditional lower bounds that can be viewed as showing hardness either for k-SUM or for the subgraph problems. We focus on showing implications of the k-SUM Conjecture and therefore view the first two kinds as a source for conditional lower bounds for EXACT-WEIGHT-H, while we view the last kind as algorithms for solving the problem. Our results are summarized in Table 1 and Figure 1, and are discussed below.

Hardness. By embedding the numbers of the k-SUM problem into the edge weights of the exact-weight subgraph problem, using different encodings depending on the structure of the subgraph, we prove four reductions that are summarized in Theorem 1:

Theorem 1. *Let $k \geq 3$. If for all $\varepsilon > 0$, k-SUM cannot be solved in time $O(n^{\lceil k/2 \rceil - \varepsilon})$, then none of the following problems[3] can be solved in time $O(n^{\lceil k/2 \rceil - \delta})$, for any $\delta > 0$:*

- EXACT-WEIGHT-H *on a graph on n nodes, for any subgraph H on k nodes.*
- EXACT-WEIGHT-k-MATCHING *on a graph on $\sqrt{n}$ nodes.*
- EXACT-WEIGHT-k-STAR *on a graph on $n^{(1-1/k)}$ nodes.*
- EXACT-WEIGHT-$(k$-1)-PATH *on a graph on n nodes.*

An immediate implication of Theorem 1 is that neither 3-STAR can be solved in time $O(n^{3-\varepsilon})$, nor can 3-MATCHING be solved in time $O(n^{4-\varepsilon})$ for some $\varepsilon > 0$, unless 3-SUM can be solved in time $O(n^{2-\varepsilon'})$ for some $\varepsilon' > 0$. We later show that an $O(n^{2-\varepsilon})$ algorithm for 3-SUM for some $\varepsilon > 0$ will imply both an $\tilde{O}(n^{3-\varepsilon})$ algorithm for 3-STAR and an $\tilde{O}(n^{4-2\varepsilon})$ algorithm for 3-MATCHING. In other words, either *all* of the following three statements are true, or *none* of them are:

- 3-SUM can be solved in time $O(n^{2-\varepsilon})$ for some $\varepsilon > 0$.
- 3-STAR can be solved in time $O(n^{3-\varepsilon})$ for some $\varepsilon > 0$.
- 3-MATCHING can be solved in time $O(n^{4-\varepsilon})$ for some $\varepsilon > 0$.

From [17], we already know that solving 3-CLIQUE in time $O(n^{3-\varepsilon})$ for some $\varepsilon > 0$ implies that 3-SUM can be solved in time $O(n^{2-\varepsilon})$ for some $\varepsilon > 0$. By Theorem 1, this would imply faster algorithms for 3-STAR and 3-MATCHING as well.

Another corollary of Theorem 1 is the fact that 4-PATH cannot be solved in time $O(n^{3-\varepsilon})$ for some $\varepsilon > 0$ unless 5-SUM can be solved in time $O(n^{3-\varepsilon'})$ for some $\varepsilon' > 0$. This is in sharp contrast to the unweighted version (and the min-weight version) of 4-PATH, which can both be solved easily in time $O(n^2)$.

Theorem 1 shows that the k-SUM problem can be reduced to the EXACT-WEIGHT-H problem for various types of subgraphs H, and as we noted, this implies connections between the exact-weight problem for different subgraphs. It is natural to ask if for any other subgraphs the exact-weight problems can be related to one another. We will

[3] k-MATCHING denotes the k-edge matching on $2k$ nodes. k-STAR denotes the k-edge star on $k+1$ nodes. k-PATH denotes the k-node path on k-1 edges.

answer this question in the affirmative—in particular, we show a tight reduction from 3-CLIQUE to 4-PATH.

To get this result, we use the edge weights to encode information about the nodes in order to prove a reduction from EXACT-WEIGHT-H_1 to EXACT-WEIGHT-H_2, where H_1 is what we refer to as a "vertex-minor" of H_2. Informally, a vertex-minor of a graph is one that is obtained by edge deletions and node identifications (contractions) for arbitrary pairs of nodes of the original graph (see Section 4 for a formal definition). For example, the triangle subgraph is a vertex-minor of the path on four nodes, which is itself a vertex-minor of the cycle on four nodes.

Theorem 2. *Let H_1, H_2 be subgraphs such that H_1 is a vertex-minor of H_2. For any $\alpha \geq 2$, if* EXACT-WEIGHT-H_2 *can be solved in time $O(n^\alpha)$, then* EXACT-WEIGHT-H_1 *can be solved in time $\tilde{O}(n^\alpha)$.*

Therefore, Theorem 2 allows us to conclude that 4-CYCLE cannot be solved in time $O(n^{3-\varepsilon})$ for some $\varepsilon > 0$ unless 4-PATH can be solved in time $\tilde{O}(n^{3-\varepsilon})$, which cannot happen unless 3-CLIQUE can be solved in time $\tilde{O}(n^{3-\varepsilon})$.

To complete the picture of relations between 3-edge subgraphs, consider the subgraph composed of a 2-edge path along with another (disconnected) edge. We call this the "VI" subgraph and we define the EXACT-WEIGHT-VI problem appropriately. Since the path on four nodes is a vertex-minor of VI, we have that an $O(n^{3-\varepsilon})$ for some $\varepsilon > 0$ algorithm for EXACT-WEIGHT-VI implies an $\tilde{O}(n^{3-\varepsilon})$ algorithm for 4-PATH. In Figure 1, we show this web of connections between the exact-weight 3-edge subgraph problems and its connection to 3-SUM, 5-SUM, and ALL-PAIRS SHORTEST PATHS. In fact, we will soon see that the conditional lower bounds we have established for these 3-edge subgraph problems are all tight. Note that the detection and minimum-weight versions of some of these 3-edge subgraph problems can all be solved much faster than $O(n^3)$ (in particular, $O(n^2)$), and yet such an algorithm for the exact-weight versions for *any* of these problems will refute the 3-SUM Conjecture, the 5-SUM Conjecture, and lead to breakthrough improvements in algorithms for solving ALL-PAIRS SHORTEST PATHS and other important graph and matrix optimization problems (cf. [19])!

Another $O(n^3)$ solvable problem is the EXACT-WEIGHT-5-PATH, and by noting that both 4-CYCLE and VI are vertex-minors of 5-PATH, we get that improved algorithms for 5-PATH will yield faster algorithms for all of the above problems. Moreover, from Theorem 1, 6-SUM reduces to 5-PATH. This established EXACT-WEIGHT-5-PATH as the "hardest" of the $O(n^3)$ time problems that we consider.

We also note that Theorem 2 yields some interesting consequences under the assumption that the k-CLIQUE problem cannot be solved in time $O(n^{k-\varepsilon})$ for some $\varepsilon > 0$. Theoretical evidence for this assumption was provided in [17], where they show how an $O(n^{k-\varepsilon})$ for some $\varepsilon > 0$ time algorithm for EXACT-WEIGHT-k-CLIQUE yields a subexponential time algorithm for the multivariate quadratic equations problem, a problem whose hardness is assumed in post-quantum cryptography.

We note that the 4-clique is a vertex-minor of the 8-node path, and so by Theorem 2, an $O(n^{4-\varepsilon})$ for some $\varepsilon > 0$ algorithm for 8-PATH will yield a faster 4-CLIQUE algorithm. Note that an $O(n^{5-\varepsilon})$ algorithm for 8-PATH already refutes the 9-SUM

Conjecture. However, this by itself is not known to imply faster clique algorithms[4]. Also, there are other subgraphs for which one can only rule out $O(n^{\lceil k/2 \rceil - \varepsilon})$ for $\varepsilon > 0$ upper bounds from the k-SUM Conjecture, while assuming hardness for the k-CLIQUE problem and using Theorem 2, much stronger lower bounds can be achieved.

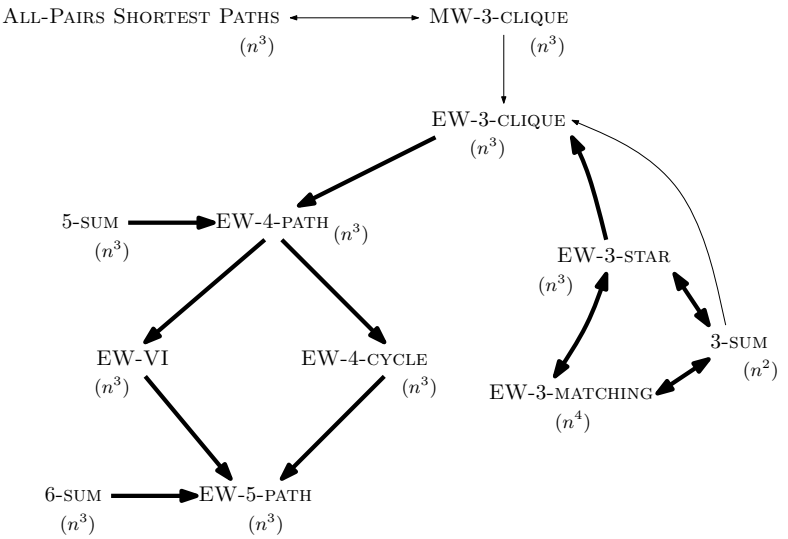


Fig. 1. A diagram of the relationships between EXACT-WEIGHT-H (denoted EW, for small subgraphs H) and other important problems. The best known running times are given for each problem, and an arrow $A \to B$ denotes that A can be tightly reduced to B, in the sense that improving the stated running time for B will imply an improvement on the stated running time for A. The reductions established in this work are displayed in bold, the others are due to [19], [17].

Algorithms. So far, our reductions only show one direction of the relationship between k-SUM and the exact-weight subgraph problems. We now show how to *use* k-SUM to solve EXACT-WEIGHT-H, which will imply that many of our previous reductions are indeed tight. The technique for finding an H-subgraph is to enumerate over a set of d smaller subgraphs that partition H in a certain way. Then, in order to determine whether the weights of these d smaller subgraphs sum up to the target weight, we use d-SUM. We say that $(S, H_1, \ldots, H_d)$ is a *d-separator* of H iff $S, H_1, \ldots, H_d$ partition $V(H)$ and there are no edges between a vertex in H_i and a vertex in H_j for any distinct $i, j \in [d]$.

Theorem 3. *Let $(S, H_1, \ldots, H_d)$ be a d-separator of H. Then,* EXACT-WEIGHT-H *can be reduced to $\tilde{O}(n^{|S|})$ instances of* d-SUM *each on $\max\{n^{|H_1|}, \ldots, n^{|H_d|}\}$ numbers.*

By using the known d-SUM algorithms, Theorem 3 gives a non-trivial algorithm for exact-weight subgraph problems. The running time of the algorithm depends on the choice of the separator used for the reduction. We observe that the optimal running

[4] It is not known whether the assumption that k-CLIQUE cannot be solved in time $O(n^{k-\varepsilon})$ for any $\varepsilon > 0$ is stronger or weaker than the k-SUM Conjecture.

time can be achieved even when $d = 2$, and can be identified (naively, in time $O(3^k)$) using the following expression. Let

$$\gamma(H) = \min_{(S,H_1,H_2) \text{ is a 2-separator}} \{|S| + \max\{|H_1|, |H_2|\}\}.$$

Corollary 1. EXACT-WEIGHT-H *can be solved in time* $\tilde{O}(n^{\gamma(H)})$.

Corollary 1 yields the upper bounds that we claim in Figure 1 and Table 1. For example, to achieve the $O(n^{\lceil (k+1)/2 \rceil})$ time complexity for k-PATH, observe that we can choose the set containing just the "middle" node of the path to be S, so that the graph $H \setminus S$ is split into two disconnected halves H_1 and H_2, each of size at most $\lceil (k-1)/2 \rceil$. Note that this is the optimal choice of a separator, and so $\gamma(k\text{-PATH}) = 1 + \lceil (k-1)/2 \rceil = \lceil (k+1)/2 \rceil$. It is interesting to note that this simple algorithm achieves running times that match many of our conditional lower bounds. This means that in many cases, improving on this algorithm will refute the k-SUM Conjecture, and in fact, we are not aware of any subgraph for which a better running time is known.

EXACT-WEIGHT-H is solved most efficiently by our algorithm when $\gamma(H)$ is small, that is, subgraphs with small "balanced" separators. Two such cases are when H has a large independent set and when H has bounded treewidth. We show that EXACT-WEIGHT-H can be solved in time $O(n^{k-\lfloor \frac{s}{2} \rfloor})$, if $\alpha(H) = s$, and in time $O(n^{\frac{2}{3} \cdot k + tw(H)})$. Also, we observe that our algorithm can be modified slightly to get an algorithm for the minimization problem.

Theorem 4. *Let H be a subgraph on k nodes, with independent set of size s. Given a graph G on n nodes with node and edge weights, the minimum total weight of a (not necessarily induced) subgraph of G that is isomorphic to H can be found in time* $\tilde{O}(n^{k-s+1})$.

This algorithm improves on the $O(n^{k-s+2})$ time algorithm of Vassilevska and Williams [17] for the MIN-WEIGHT-H problem.

Organization. We give formal definitions in Section 2. In Section 3 we present reductions from k-SUM to exact-weight subgraph problems that prove Theorem 1. In Section 4 we define vertex-minors and outline the proof of Theorem 2. In Section 5, we present the reduction of Theorem 3.

2 Preliminaries and Basic Constructions

For a graph G, we will use $V(G)$ to represent the set of vertices and $E(G)$ to represent the set of edges. The notation $N(v)$ will be used to represent the neighborhood of a vertex $v \in V(G)$.

2.1 Reducibility

We will use the following notion of reducibility between two problems. In weighted graph problems where the weights are integers in $[-M, M]$, n will refer to the number

Table 1. The results shown in this work for EXACT-WEIGHT-H for various H. The second column has the upper bound achieved by our algorithm from Corollary 1. Improvements on the lower bound in the third column will imply improvements on the best known algorithms for the problems in the condition column. These lower bounds are obtained by our redctions, except for the first row which was proved in [17]. For comparison, we give the running times for the (unweighted) detection and minimum-weight versions of the subgraph problems. The last row shows our conditional lower bounds for k-SUM. $\alpha(H)$ represents the independence number of H, $tw(H)$ is its treewidth. The results for the "Any" row hold for all subgraphs on k nodes. ETH stands for the Exponential Time Hypothesis.

Subgraph	Exact	Lower Bound	Condition	Detection	Min
3-CLIQUE	n^3	n^3	3-SUM, APSP	n^ω [10]	n^3
4-PATH	n^3	n^3	3-CLIQUE, 5-SUM	n^2	n^2
k-MATCHING	$n^{2\cdot\lceil\frac{k}{2}\rceil}$	$n^{2\cdot\lceil\frac{k}{2}\rceil}$	k-SUM	n^2	n^2
k-STAR	$n^{\lceil\frac{k}{2}\rceil+1}$	$n^{\lceil\frac{k+1}{2}\rceil}$ $n^{\lceil\frac{k}{2}\rceil\cdot\frac{k}{k-1}}$	$(k+1)$-SUM k-SUM	n	n^2
k-PATH	$n^{\lceil\frac{k+1}{2}\rceil}$	$n^{\lceil\frac{k+1}{2}\rceil}$	k-SUM	n^2	n^2
k-CYCLE	$n^{\lceil\frac{k}{2}\rceil+1}$	$n^{\lceil\frac{k+1}{2}\rceil}$	k-PATH	n^2	n^3
Any	n^k	$n^{\lceil k/2\rceil}$ $n^{\varepsilon k}$	k-SUM (ETH)	$n^{\omega k/3}$ [12]	n^k
$\alpha(H)=s$	$n^{k-\lfloor\frac{s}{2}\rfloor}$	$n^{\lceil\frac{k}{2}\rceil}$	k-SUM	n^{k-s+1} [13]	n^{k-s+1} [Thm. 4]
$tw(H)=w$	$n^{\frac{2}{3}k+w}$	$n^{\lceil\frac{k}{2}\rceil}$	k-SUM	n^{w+1} [2]	n^{2w} [8]
k-SUM	$n^{\lceil\frac{k}{2}\rceil}$	$n^{\lceil\frac{k}{2}\rceil}$	k-MATCHING, k-STAR	-	-

of nodes times $\log M$. For k-SUM problems where the input integers are in $[-M, M]$, n will refer to the number of integers times $\log M$. In the full version of the paper we formally define our notion of reducibility, which follows the definition of subcubic reductions in [19]. Informally, for any two decision problems A and B, we say that $A \ {}_a\!\leq_b B$ if for any $\epsilon > 0$, there exists a $\delta > 0$ such that if B can be solved (w.h.p.) in time $n^{b-\epsilon}$, then A can be solved (w.h.p.) in time $O(n^{a-\delta})$, where n is the size of the input. Note that polylog(n) factor improvements in solving B may not imply any improvements in solving A. Also, we say that $A \ {}_a\!\equiv_b B$ if and only if $A \ {}_a\!\leq_b B$ and $B \ {}_b\!\leq_a A$.

2.2 The k-SUM Problem

Throughout the paper, it will be more convenient to work with a version of the k-SUM problem that is more structured than the basic formulation. This version is usually referred to as either TABLE-k-SUM or k-SUM$'$, and is known to be equivalent to the basic formulation, up to k^k factors (by a simple extension of Theorem 3.1 in [9]). For con-

venience, and since $f(k)$ factors are ignored in our running times, we will refer to this problem as k-SUM.

Definition 2 (k-SUM). *Given k lists $L_1, \ldots, L_k$ each with n numbers where $L_i = \{x_{i,j}\}_{j\in[n]} \subseteq \mathbb{Z}$, do there exist k numbers $x_{1,a_1}, \ldots, x_{k,a_k}$, one from each list, such that $\sum_{i=1}^{k} x_{i,a_i} = 0$?*

In our proofs, we always denote an instance of k-SUM by $L_1, \ldots, L_k$, where $L_i = \{x_{i,j}\}_{j\in[n]} \subseteq \mathbb{Z}$, so that $x_{i,j}$ is the j^{th} number of the i^{th} list L_i. We define a k-solution to be a set of k numbers $\{x_{i,a_i}\}_{i\in[k]}$, one from each list. The sum of a k-solution $\{x_{i,a_i}\}_{i\in[k]}$ will be defined naturally as $\sum_{i=1}^{k} x_{i,a_i}$.

In [15], Pătraşcu defines the CONVOLUTION-3-SUM problem. We consider a natural extension of this problem.

Definition 3 (CONVOLUTION-k-SUM). *Given k lists $L_1, \ldots, L_k$ each with n numbers, where $L_i = \{x_{i,j}\}_{j\in[n]} \subseteq \mathbb{Z}$, does there exist a k-solution $\{x_{i,a_i}\}_{i\in[k]}$ such that $a_k = a_1 + \cdots + a_{k-1}$ and $\sum_{i=1}^{k} x_{i,a_i} = 0$?*

Theorem 10 in [15] shows that 3-SUM $_2\leq_2$ CONVOLUTION-3-SUM. By generalizing the proof, we show the following useful lemma (see proof in the full version of the paper).

Lemma 1. *For all $k \geq 2$, k-SUM $_{\lceil k/2\rceil}\leq_{\lceil k/2\rceil}$ CONVOLUTION-k-SUM.*

2.3 H-Partite Graphs

Let H be a subgraph on k nodes with $V(H) = \{h_1, \ldots, h_k\}$.

Definition 4 (H-partite graph). *Let G be a graph such that $V(G)$ can be partitioned into k sets $P_{h_1}, \ldots, P_{h_k}$, each containing n vertices. We will refer to these k sets as the* super-nodes *of G. A pair of super-nodes (P_{h_i}, P_{h_j}) will be called a* super-edge *if $(h_i, h_j) \in E(H)$. Then, we say that G is H-*partite *if every edge in $E(G)$ lies in some super-edge of G.*

We denote the set of vertices of an H-partite graph G by $V(G) = \{v_{i,j}\}_{i\in[k],j\in[n]}$, where $v_{i,j}$ is the j^{th} vertex in super-node P_{h_i}. We will say that G is the *complete H-partite graph* when $(v_{i,a}, v_{j,b}) \in E(G)$ if and only if (P_{h_i}, P_{h_j}) is a super-edge of G, for all $a, b \in [n]$.

An *H-subgraph* of an H-partite graph G, denoted by $\chi = \{v_{i,a_i}\}_{i\in[k]} \subseteq V(G)$, is a set of vertices for which there is exactly one vertex v_{i,a_i} from each super-node P_{h_i}, where a_i is an index in $[n]$. Given a weight function $w : (V(G) \cup E(G)) \to \mathbb{Z}$ for the nodes and edges of G, the total weight of the subgraph χ is defined naturally as

$$w(\chi) = \sum_{h_i\in V(H)} w(v_{i,a_i}) + \sum_{(h_i,h_j)\in E(H)} w(v_{i,a_i}, v_{j,a_j}).$$

Now, we define a more structured version of the EXACT-WEIGHT-H problem which is easier to work with.

Definition 5 (The EXACT-H Problem). *Given a complete H-partite graph graph G with a weight function $w : (V(G) \cup E(G)) \to \mathbb{Z}$ for the nodes and edges, does there exist an H-subgraph of total weight 0?*

In the full version of the paper, we prove the following lemma, showing that the two versions of the EXACT-WEIGHT-H problem are reducible to one another in a tight manner. All of our proofs will use the formulation of EXACT-H, yet the results will also apply to EXACT-WEIGHT-H. Note that our definitions of H-partite graphs uses ideas similar to color-coding [2].

Lemma 2. *Let $\alpha > 1$.* EXACT-WEIGHT-H $_{\alpha}\equiv_{\alpha}$ EXACT-H.

3 Reductions from k-SUM to Subgraph Problems

In this section we prove Theorem 1 by proving four reductions, each of these reductions uses a somewhat different way to encode k-SUM in the structure of the subgraph. First, we give a generic reduction from k-SUM to EXACT-H for an arbitrary H on k nodes. We set the node weights of the graph to be the numbers in the k-SUM instance, in a certain way. See the full version of the paper for a detailed proof.

Lemma 3 (k-SUM $_{\lceil k/2 \rceil}\leq_{\lceil k/2 \rceil}$ EXACT-H). *Let H be a subgraph with k nodes. Then, k-SUM on n numbers can be reduced to a single instance of* EXACT-H *on kn vertices.*

We utilize the edge weights of the graph, rather than the node weights, to prove a tight reduction to k-MATCHING. See the full version of the paper for a detailed proof.

Lemma 4 (k-SUM $_{\lceil k/2 \rceil}\leq_{2\cdot\lceil k/2 \rceil}$EXACT-$k$-MATCHING). *Let H be the k-*MATCHING *subgraph. Then, k-*SUM *on n numbers can be reduced to a single instance of* EXACT-H *on $k\sqrt{n}$ vertices.*

Another special type of subgraph which can be shown to be tightly related to the k-SUM problem is the k-edge star subgraph.

Lemma 5 (k-SUM $_{\lceil \frac{k}{2} \rceil}\leq_{\lceil \frac{k}{2} \rceil\cdot\frac{k}{k-1}}$ EXACT-k-STAR). *Let H be the k-*STAR *subgraph, and let $\alpha > 2$. If* EXACT-H *can be solved in $O(n^{\alpha})$ time, then k-*SUM *can be solved in $\tilde{O}(n^{(1-1/k)\cdot\alpha})$ time.*

To prove the lemma we define the problem k-SUM^n to be the following. Given a sequence of n k-SUM instances, each on n numbers, does there exist an instance in the sequence that has a solution of sum 0? Then, we prove two claims. The first showing that k-SUM^n can be reduced to k-STAR, by associating a node in the graph with a k-SUM instance, and setting the weights of the edges incident to it to the numbers of that instance, such that each k-solution of that instance will be a k-STAR centered on that node. The second claim shows that k-SUM can be reduced to k-SUM^n by showing a self reduction for k-SUM. We use a hashing scheme due to Dietzfelbinger [4], to hash the numbers into $O(n^{1/k})$ buckets with $O(n^{1-1/k})$ numbers each, such that for every

$k-1$ numbers, the numbers that can complete a k-solution of sum 0 can only lie in certain k buckets. We utilize this to generate $O((n^{1/k})^{k-1})$ instances of k-SUM, each with $O(n^{1-1/k})$ numbers, such that one of them will have a solution iff the original k-SUM instance had a solution. This is a k-SUM^N instance, for $N = O(n^{1-1/k})$. A more detailed proof is given in the full version of the paper.

Our final reduction between k-SUM and EXACT-H for a class of subgraphs H is as follows. First, define the k-PATH subgraph H to be such that $V(H) = \{h_1, \ldots, h_k\}$ and $E(H) = \{(h_1, h_2), (h_2, h_3), \ldots, (h_{k-1}, h_k)\}$.

Lemma 6 (k-SUM $\lceil \frac{k}{2} \rceil \leq \lceil \frac{k}{2} \rceil$ EXACT-$(k$-1$)$-PATH). *Let H be the k-*PATH *subgraph. If* EXACT-H *can be solved in time $O(n^{\lceil k/2 \rceil - \varepsilon})$ for some $\varepsilon > 0$, then k+1-*SUM *can be solved in time $O(n^{\lceil k/2 \rceil - \varepsilon'})$, for some $\varepsilon' > 0$.*

Proof. We prove that an instance of CONVOLUTION-(k+1)-SUM on n numbers can be reduced to a single instance of EXACT-k-PATH, and by applying Lemma 1, this completes the proof. Given $k+1$ lists $L_1, \ldots, L_{k+1}$ each with n numbers as the input to CONVOLUTION-(k+1)-SUM, we will construct a complete H-partite graph G on kn nodes. For every r and s such that $r - s \in [n]$, for all $i \in [k]$, define the edge weights of G in the following manner.

$$w(v_{i,r}, v_{i+1,s}) = \begin{cases} x_{1,r} + x_{2,s-r}, & \text{if } i = 1 \\ x_{i+1,s-r}, & \text{if } 1 < i < k \\ x_{k,s-r} + x_{k+1,s}, & \text{if } i = k \end{cases}$$

Otherwise, if $r - s \notin [n]$, we set $w(v_{i,r}, v_{i+1,s}) = -\infty$ for all $i \in [k]$. Now to see the correctness of the reduction, take any H-subgraph $\{v_{i,a_i}\}_{i \in [k]}$ of G, and consider the $(k+1)$-solution $\{x_{i,b_i}\}_{i \in [k+1]}$, where $b_1 = a_1$, $b_{k+1} = a_k$, and for $2 \leq i \leq k$, $b_i = a_i - a_{i-1}$. First, note that the $(k+1)$-solution satisfies the property that $b_1 + \ldots + b_k = b_{k+1}$. Now, note that its total weight is $\sum_{i=1}^{k} w(v_{i,a_i}, v_{i+1,a_{i+1}}) = x_{1,a_1} + x_{2,a_2-a_1} + x_{3,a_3-a_2} + \ldots + x_{k-1,a_{k-1}-a_{k-2}} + x_{k,a_k-a_{k-1}} + x_{k+1,a_k} = \sum_{i=1}^{k+1} x_{i,b_i}$ which is exactly the sum of the (k+1)-solution. For the other direction, consider the (k+1)-solution $\{x_{i,b_i}\}_{i \in [k+1]}$ for which $b_{k+1} = b_1 + \ldots + b_{k+1}$. Then, the H-subgraph $\{v_{i,a_i}\}_{i \in [k]}$, where $a_i = b_1 + \ldots + b_i$, has total weight $\sum_{i=1}^{k+1} x_{i,b_i}$. Therefore, there is a k-solution of sum 0 iff there is an H-subgraph in G of total weight 0. □

4 Relationships between Subgraphs

In this section we outline the proof of Theorem 2 showing that EXACT-H_1 can be reduced to EXACT-H_2 if H_1 is a *vertex-minor* of H_2. The complete proof is in the full version of the paper, along with an additional observation that gives a reverse reduction. We start by defining vertex-minors.

Definition 6 (Vertex-Minor). *A graph H_1 is called a vertex-minor of graph H_2, and denoted $H_1 \leq_{\mathsf{vm}} H_2$, if there exists a sequence of subgraphs $H^{(1)}, \ldots, H^{(\ell)}$ such that $H_1 = H^{(1)}$, $H_2 = H^{(\ell)}$, and for every $i \in [\ell - 1]$, $H^{(i)}$ can be obtained from $H^{(i+1)}$ by either*

- *Deleting a single edge* $e \in E(H^{(i+1)})$*, or*
- *Contracting two nodes*[5] $h_j, h_k \in V(H^{(i+1)})$ *to one node* $h_{jk} \in V(H^{(i)})$*, such that* $N(h_{jk}) = N(h_j) \cup N(h_k)$.

To prove Theorem 2 it suffices to show how to reduce EXACT-H_1 to EXACT-H_2 when H_1 is obtained by either a single edge deletion or a single edge contraction. The edge deletion reduction is easy, and the major part of the proof will be showing the contraction reduction. The main observation is that we can make two copies of nodes and change their node weights in a way such that any H_2-subgraph of total weight 0 that contains one of the copies will have to contain the other. This will allow us to claim that the subgraph obtained by replacing the two copies of a node with the original will be an H_1-subgraph of total weight 0.

5 Reductions to k-SUM (and Upper Bounds)

In this section we explain the reduction used to prove Theorem 3. The full version of the paper has the proof of correctness, followed by a discussion of the algorithmic implications of Theorem 3, including Corollary 1, and Theorem 4.

Recall the definition of a d-separator $(S, H_1, \ldots, H_d)$ of graph H. To solve EXACT-H, we do the following. For each choice of an S-subgraph $\chi_S = \{v_{j,a_j}\}_{h_j \in S}$ of G, we create an instance of d-SUM where the lists $L_1, \ldots, L_d$ are of size $\max\{n^{|H_1|}, \ldots, n^{|H_d|}\}$. We construct L_i by iterating over each H_i-subgraph $\chi_{H_i} = \{v_{j,a_j}\}_{h_j \in H_i}$ and adding the total weight of the $(H_i \cup S)$-subgraph $\chi_{H_i} \cup \chi_S$ as an integer to L_i. This process is repeated for each L_i, and the target of the d-SUM instance is set to be $w(\chi_S) \cdot (d-1)$. Then, if the input graph has an H-subgraph of total weight 0, then for some choice of an S-subgraph, the corresponding d-SUM instance will have a d-solution with sum equal to the target value.

6 Conclusions

We conclude with two interesting open questions:

1. Perhaps the simplest subgraph for which we cannot give tight lower and upper bounds is the 5-CYCLE subgraph. Can we achieve $O(n^{4-\varepsilon})$ for some $\varepsilon > 0$ without breaking the k-SUM Conjecture, or can we prove that it is not possible?
2. Can we prove that EXACT-WEIGHT-4-PATH $\,{}_3\!\leq_3$ EXACT-WEIGHT-3-STAR? This would show that breaking the 3-SUM Conjecture will imply an $O(n^{3-\varepsilon})$ for some $\varepsilon > 0$ algorithm for ALL-PAIRS SHORTEST PATHS.

Acknowledgements. The authors would like to thank Ryan and Virginia Williams for many helpful discussions and for sharing their insights, and Hart Montgomery for initiating the conversation that led up to this work. We would also like to thank the anonymous reviewers for their comments and suggestions.

[5] The difference between our definition of vertex-minor and the usual definition of a graph minor is that we allow contracting two nodes that are not necessarily connected by an edge.

References

1. Ailon, N., Chazelle, B.: Lower bounds for linear degeneracy testing. J. ACM 52(2), 157–171 (2005)
2. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM 42(4), 844–856 (1995)
3. Baran, I., Demaine, E.D., Pătraşcu, M.: Subquadratic algorithms for 3SUM. Algorithmica 50(4), 584–596 (2008); In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 409–421. Springer, Heidelberg (2005)
4. Dietzfelbinger, M.: Universal hashing and k-wise independent random variables via integer arithmetic without primes. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 569–580. Springer, Heidelberg (1996)
5. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness ii: On completeness for w[1] (1995)
6. Eisenbrand, F., Grandoni, F.: On the complexity of fixed parameter clique and dominating set. Theor. Comput. Sci. 326(1-3), 57–67 (2004)
7. Erickson, J.: Lower bounds for linear satisfiability problems. In: Clarkson, K.L. (ed.) SODA, pp. 388–395. ACM/SIAM (1995)
8. Fomin, F.V., Lokshtanov, D., Raman, V., Saurabh, S., Raghavendra Rao, B.V.: Faster algorithms for finding and counting subgraphs. J. Comput. Syst. Sci. 78(3), 698–706 (2012)
9. Gajentaan, A., Overmars, M.H.: On a class of $o(n^2)$ problems in computational geometry. Computational Geometry 5(3), 165–185 (1995)
10. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. In: STOC 1977, pp. 1–10. ACM, New York (1977)
11. Jafargholi, Z., Viola, E.: 3sum, 3xor, triangles. Electronic Colloquium on Computational Complexity (ECCC) 20, 9 (2013)
12. Kloks, T., Kratsch, D., Müller, H.: Finding and counting small induced subgraphs efficiently. Inf. Process. Lett. 74(3-4), 115–121 (2000)
13. Kowaluk, M., Lingas, A., Lundell, E.-M.: Counting and detecting small subgraphs via equations and matrix multiplication. In: SODA 2011, pp. 1468–1476. SIAM (2011)
14. Neetil, J., Poljak, S.: On the complexity of the subgraph problem. Commentationes Mathematicae Universitatis Carolinae 026(2), 415–419 (1985)
15. Pătraşcu, M.: Towards polynomial lower bounds for dynamic problems. In: Proc. 42nd ACM Symposium on Theory of Computing (STOC), pp. 603–610 (2010)
16. Pătraşcu, M., Williams, R.: On the possibility of faster sat algorithms. In: Proc. 21st ACM/SIAM Symposium on Discrete Algorithms (SODA), pp. 1065–1075 (2010)
17. Vassilevska, V., Williams, R.: Finding, minimizing, and counting weighted subgraphs. In: Mitzenmacher, M. (ed.) STOC, pp. 455–464. ACM (2009)
18. Williams, R.: Finding paths of length k in $o^*(2^k)$ time. Inf. Process. Lett. 109(6), 315–318 (2009)
19. Williams, V.V., Williams, R.: Subcubic equivalences between path, matrix and triangle problems. In: FOCS, pp. 645–654. IEEE Computer Society (2010)

Minimizing Maximum (Weighted) Flow-Time on Related and Unrelated Machines

S. Anand[1], Karl Bringmann[2,⋆], Tobias Friedrich[3], Naveen Garg[1,⋆⋆], and Amit Kumar[1]

[1] Department of Computer Science and Engineering, IIT Delhi, India
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[3] Friedrich-Schiller-Universität Jena, Germany

Abstract. We initiate the study of job scheduling on related and unrelated machines so as to minimize the maximum flow time or the maximum weighted flow time (when each job has an associated weight). Previous work for these metrics considered only the setting of parallel machines, while previous work for scheduling on unrelated machines only considered L_p, $p < \infty$ norms. Our main results are:

1. We give an $O(\varepsilon^{-3})$-competitive algorithm to minimize maximum weighted flow time on related machines where we assume that the machines of the online algorithm can process $1 + \varepsilon$ units of a job in 1 time-unit (ε speed augmentation).
2. For the objective of minimizing maximum flow time on unrelated machines we give a simple $2/\varepsilon$-competitive algorithm when we augment the speed by ε. For m machines we show a lower bound of $\Omega(m)$ on the competitive ratio if speed augmentation is not permitted. Our algorithm does not assign jobs to machines as soon as they arrive. To justify this "drawback" we show a lower bound of $\Omega(\log m)$ on the competitive ratio of *immediate dispatch* algorithms. In both these lower bound constructions we use jobs whose processing times are in $\{1, \infty\}$, and hence they apply to the more restrictive *subset parallel* setting.
3. For the objective of minimizing maximum weighted flow time on unrelated machines we establish a lower bound of $\Omega(\log m)$-on the competitive ratio of any online algorithm which is permitted to use $s = O(1)$ speed machines. In our lower bound construction, job j has a processing time of p_j on a subset of machines and infinity on others and has a weight $1/p_j$. Hence this lower bound applies to the subset parallel setting for the special case of minimizing maximum stretch.

1 Introduction

The problem of scheduling jobs so as to minimize the flow time (or response time) has received much attention. In the online setting of this problem, jobs

⋆ Karl Bringmann is a recipient of the *Google Europe Fellowship in Randomized Algorithms*, and this research is supported in part by this Google Fellowship.

⋆⋆ Naveen Garg is supported by the Indo-German Max Planck Center for Computer Science (IMPECS).

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 13–24, 2013.

arrive over time and the flow time of a job is the difference between its release time (or arrival time) and completion time (or finish time). We assume that the jobs can be preempted. The task of the scheduler is to decide which machine to schedule a job on and in what order to schedule the jobs assigned to a machine.

One way of combining the flow times of various jobs is to consider the sum of the flow times. An obvious drawback of this measure is that it is not *fair* since some job might have a very large flow time in the schedule that minimizes sum of their flow times. A natural way to overcome this is to minimize the L_p norm of the flow times of the jobs [2,5,10,11] which, for increasing values of p, would ensure better fairness. Bansal and Pruhs [5], however, showed that even for a single machine minimizing, the L_p-norm of flow times requires speed augmentation — the online algorithm must have machines that are, say, ε-fraction faster (can do $1+\varepsilon$ unit of work in one time-unit) than those of the offline algorithm. With a $(1+\varepsilon)$-speed augmentation Bansal and Pruhs [5] showed that a simple algorithm which schedules the shortest job first is $O(\varepsilon^{-1})$-competitive for any L_p-norm on a single machine; we refer to this as an $(1+\varepsilon, O(1/\varepsilon))$-competitive algorithm. Golovin et al. [10] used a majorizing technique to obtain a similar result for parallel machines. While both these results have a competitive ratio that is independent of p, the results of Im and Moseley [11] and Anand et al. [2] for unrelated machines have a competitive ratio that is linear in p and which therefore implies an unbounded competitive ratio for the L_∞-norm.

Our main contribution in this paper is to provide a comprehensive treatment of the problem of minimizing maximum flow time for different machine models. The two models that we consider are the *related machines* (each machine has speed s_i and the time required to process job j on machine i is p_j/s_i) and the *unrelated machines* (job j has processing time p_{ij} on machine i). A special case of the unrelated machine model is the *subset-parallel* setting where job j has a processing time p_j independent of the machines but can be assigned only to a subset of the machines.

Besides maximum flow time, another metric of interest is the maximum weighted flow time where we assume that job j has a weight w_j and the objective is to minimize $\max_j w_j F_j$ where F_j is the flow time of j in the schedule constructed. Besides the obvious use of job weights to model priority, if we choose the weight of a job equal to the inverse of its processing time then minimizing maximum weighted flow time is the same as minimizing maximum *stretch* where stretch is defined as the ratio of the flow time to the processing time of a job. Chekuri and Moseley [9] considered the problem of minimizing the maximum *delay factor* where a job j has a deadline d_j, a release date r_j and the delay factor of a job is defined as the ratio of its flow time to $(d_j - r_j)$. This problem is in fact equivalent to minimizing maximum weighted flow time and this can be easily seen by defining $w_j = (d_j - r_j)^{-1}$.

The problem of minimizing maximum stretch was first considered by Bender et al. [7] who showed a lower bound of $\Omega(P^{1/3})$ on the competitive ratio for a single machine where P is the ratio of the largest to the smallest processing time. Bender et al. [7] also showed a $O(P^{1/2})$-competitive algorithm for a single

Table 1. Previous results and the results obtained in this paper for the different machine models and metrics considered. The uncited results are from this paper.

	Max-Flow-time	Max-Stretch	Max-Weighted-Flow-time
Single Machine	polynomial time	$(1, \Omega(P^{2/5}))$ [9] and $(1, O(P^{1/2}))$ [7,8]	$(1+\varepsilon, O(\varepsilon^{-2}))$ [6]
Parallel Machines	$(1, 2)$ [1]		$(1+\varepsilon, O(\varepsilon^{-1}))$ [9]
Related Machines			$(1+\varepsilon, O(\varepsilon^{-3}))$
Subset Parallel	$(1, \Omega(m))$	$(O(1), \Omega(\log m))$	
Unrelated Machines	$(1+\varepsilon, O(\varepsilon^{-1}))$		

machine which was improved by [8], while the lower bound was improved to $\Omega(P^{0.4})$ by [9].

For minimizing maximum weighted flow time, Bansal and Pruhs [6] showed that the highest density first algorithm is $(1+\varepsilon, O(\varepsilon^{-2}))$-competitive for single machines. For parallel machines, Chekuri and Moseley [9] obtained a $(1+\varepsilon, O(\varepsilon^{-1}))$-competitive algorithm that is both *non-migratory* (jobs once assigned to a machine are scheduled only on that machine) and *immediate dispatch* (a job is assigned to a machine as soon as the job arrives). Both these qualities are desirable in any scheduling algorithm since they reduce/eliminate communication overheads amongst the central server/machines.

Our main results and the previous work for these three metrics (Max-Flow-time,Max-Stretch and Max-Weighted-Flow-time) on the various machine models (single, parallel, related, subset parallel and unrelated) are expressed in Table 1. Note that the Max-Flow-time metric is not a special case of the Max-Stretch metric, and neither is the model of related machines a special case of the subset-parallel setting. Nevertheless, a lower bound result (respectively an upper bound result) for a machine-model/metric pair extends to all model/metric pairs to the right and below (respectively to the left and above) in the table. Our main results are:

1. We give an $O(\varepsilon^{-3})$-competitive non-migratory algorithm to minimize maximum weighted flow time on related machines with ε speed augmentation. When compared to a migratory optimum our solution is $\mathrm{O}(\varepsilon^{-4})$-competitive.
2. For the objective of minimizing maximum flow time on unrelated machines we give a simple $2/\varepsilon$-competitive algorithm when we augment the speed by ε. For m machines we show a lower bound of $\Omega(m)$ on the competitive ratio if speed augmentation is not permitted. Our algorithm does not assign jobs to machines as soon as they arrive. However, [4] show a lower bound of $\Omega(\log m)$ on the competitive ratio of any *immediate dispatch* algorithm. Both these lower bound constructions use jobs whose processing times are in $\{1, \infty\}$, and hence they apply to the more restrictive *subset parallel* setting.
3. For the objective of minimizing maximum weighted flow time on unrelated machines, we establish a lower bound of $\Omega(\log m)$-on the competitive ratio of any online algorithm which is permitted to use $s = O(1)$ speed machines. In our lower bound construction, job j has a processing time of p_j on a subset

of machines and infinity on others and has a weight $1/p_j$. Hence this lower bound applies to the subset parallel setting for the special case of minimizing maximum stretch.

The problem of minimizing maximum (weighted) flow time also has interesting connections to *deadline scheduling*. In deadline scheduling besides its processing time and release time, job j has an associated deadline d_j and the objective is to find a schedule which meets all deadlines. For single machine it is known that the Earliest Deadline First (EDF) algorithm is optimum, in that it would find a feasible schedule if one exists. This fact implies a polynomial time algorithm for minimizing maximum flow time on a single machine. This is because, job j released at time r_j should complete by time $r_j + \texttt{opt}$, where $\texttt{opt}$ is the optimal value of maximum flow time. Thus $r_j + \texttt{opt}$ can be viewed as the deadline of job j. Hence EDF would schedule jobs in order of their release times and does not need to know $\texttt{opt}$.

For parallel machines it is known that no online algorithm can compute a schedule which meets all deadlines even when such a schedule exists. Phillips et al. [12] showed that EDF can meet all deadlines if the machines of the on-line algorithm have twice the speed of the offline algorithms. This bound was improved to $\frac{e}{e-1}$ by Anand et al. [3] for a schedule derived from the Yardstick bound. Our results imply that for related machines a constant speedup suffices to ensure that all deadlines are met while for the subset parallel setting, no constant (independent of number of machines) speedup can ensure that we meet deadlines.

The paper is organized as follows. In Section 2 and Section 3 we consider the problem of minimizing maximum weighted flow time on related machines and unrelated machines, respectively. Section 4 considers the problem of minimizing maximum flow time on unrelated machines. Section 5 presents a lower bound for the L_p norm of the stretches.

2 Max-Weighted-Flow-time on Related Machines

In this section we consider the Max-Weighted-Flow-time on related machines where the on-line algorithm is given $(1+\varepsilon)$-speed augmentation for some arbitrary small constant $\varepsilon > 0$. In the related machines setting, each job j has weight w_j, release date r_j and processing requirement p_j. We are given m machines with varying speed. Instead of working with speed, it will be more convenient to work with *slowness* of machines: the slowness of a machine i, denoted by s_i, is the reciprocal of its speed. Assume that $s_1 \leq \ldots \leq s_m$. For an instance $\mathcal{I}$, let $\texttt{opt}(\mathcal{I})$ denote the value of the optimal off-line solution for $\mathcal{I}$. We assume that the on-line algorithm is given $(1+4\varepsilon)$-speed augmentation. We say that a job j is *valid* for a machine i, if its processing time on i, i.e., $p_j s_i$, is at most $\frac{T}{w_j}$. Observe that a (non-migratory) off-line optimum algorithm will process a job j on a valid machine only.

We assume that all weights w_j are of the form 2^k, where k is a non-negative integer (this affects the competitive ratio by a factor of 2 only). We say that a

job is of *class* k if its weight is 2^k. To begin with, we shall assume that the on-line algorithm knows the value of $\texttt{opt}(\mathcal{I})$ — call it T. In the next section, we describe an algorithm, which requires a small amount of "look-ahead". We describe it as an off-line algorithm. Subsequently, we show that it can be modified to an on-line algorithm with small loss of competitive ratio.

2.1 An Off-Line Algorithm

We now describe an off-line algorithm $\mathcal{A}$ for $\mathcal{I}$. We allow machines speedup of $1 + 2\varepsilon$. First we develop some notation. For a class k and integer l, let $I(l,k)$ denote the interval $\left[\frac{lT}{\varepsilon 2^k}, \frac{(l+1)T}{\varepsilon 2^k}\right)$. We say that a job j is of *type* (k,l) if it is of class k and $r_j \in I(k,l)$. Note that the intervals $I(k,l)$ form a nested set of intervals.

The algorithm $\mathcal{A}$ is described below. It schedules jobs in a particular order: it picks jobs in decreasing order of their class, and within each class, it goes by the order of release dates. When considering a job j, it tries machines in order of increasing speed, and schedules j in the first machine on which it can find enough free slots (i.e., slots which are not occupied by the jobs scheduled before j). We will show that it will always find some machine. Note that $\mathcal{A}$ may not respect release dates of jobs.

Algorithm $\mathcal{A}(\mathcal{I}, T)$:

For $k = K$ downto 1 (K is the highest class of a job)
 For $l = 1, 2, \ldots$
  For each job j of type (k,l)
   For $i = m_j$ downto 1 (m_j is the slowest machine on which j is valid)
    if there are at least $p_j s_i$ free slots on machine i during $I(k,l)$ then
     schedule j on i during the first such free slots (without caring about r_j)

Analysis. In this section, we prove that the algorithm $\mathcal{A}$ will always find a suitable machine for every job. We prove this by contradiction: let $j^\star$ be the first job for which we are not able to find such a machine. Then we will show that the $\texttt{opt}(\mathcal{I})$ must be more than T, which will contradict our assumption.

In the discussion below, we only look at jobs which were considered before $j^\star$ by $\mathcal{A}$. We build a set S of jobs recursively. Initially S just contains $j^\star$. We add a job j' of type (k', l') to S if there is a job j of type (k,l) in S satisfying the following conditions:

- The class k of j is at most k'.
- The algorithm $\mathcal{A}$ processes j' on a machine i which is valid for j as well.
- The algorithm $\mathcal{A}$ processes j' during $I(k,l)$, i.e., $I(k',l') \subseteq I(k,l)$.

We use this rule to add jobs to S as long as possible. For a machine i and interval $I(k,l)$, define the *machine-interval* $I_i(k,l)$ as the time interval $I(k,l)$ on machine i. We construct a set $\mathcal{N}$ of machine-intervals as follows. For every job $j \in S$ of

type (k,l), we add the intervals $I_i(k,l)$ to $\mathcal{N}$ for all machines i such that j is valid for i. We say that an interval $I_i(k,l) \in \mathcal{N}$ is *maximal* if there is no other interval $I_i(k',l') \in \mathcal{N}$ which contains $I_i(k,l)$ (note that both of the intervals are on the same machine). Observe that every job in S except $j^\star$ gets processed in one of the machine-intervals in $\mathcal{N}$. Let $\mathcal{N}'$ denote the set of maximal intervals in $\mathcal{N}$. We now show that the jobs in S satisfy the following crucial property.

Lemma 1. *For any maximal interval $I_i(k,l) \in \mathcal{N}$, the algorithm $\mathcal{A}$ processes jobs of S on all but $\frac{\varepsilon}{1+2\varepsilon}$-fraction of the slots in it.*

Proof. We prove that this property holds whenever we add a new maximal interval to $\mathcal{N}$. Suppose this property holds at some point in time, and we add a job j' to S. Let j,k,l,k',l',i be as in the description of S. Since $k \le k'$, and j is valid for i, $\mathcal{N}$ already contains the intervals $I_{i'}(k,l)$ for all $i' \le i$. Hence, the intervals $I_{i'}(k',l')$, $i' \le i$, cannot be maximal. Suppose an interval $I_{i'}(k',l')$ is maximal, where $i' > i$, and j' is valid for i' (so this interval gets added to $\mathcal{N}$). Now, our algorithm would have considered scheduling j' on i' before going to i — so it must be the case that all but $p_{j'}s_{i'}$ slots in $I_{i'}(k',l')$ are busy processing jobs of class at least k'. Further, all the jobs being processed on these slots will get added to S (by definition of S, and the fact that $j' \in S$). The lemma now follows because $p_{j'}s_{i'} \le \frac{T}{2^{k'}} \le \varepsilon|I(k',l')|$, and $\mathcal{A}$ can do $(1+2\varepsilon)|I(k,l)|$ amount of processing during $I(k,l)$. □

Corollary 1. *The total volume of jobs in S is greater than $\sum_{I(k,l)\in\mathcal{N}'}(1+\varepsilon)|I(k,l)|$.*

Proof. Lemma 1 shows that given any maximal interval $I_i(k,l)$, $\mathcal{A}$ processes jobs of S for at least $\frac{1+\varepsilon}{1+2\varepsilon}$-fraction of the slots in it. The total volume that it can process in $I(k,l)$ is $(1+2\varepsilon)|I(k,l)|$. The result follows because maximal intervals are disjoint (we have strict inequality because $\mathcal{A}$ could not complete j^*). □

We now show that the total volume of jobs in S cannot be too large, which leads to a contradiction.

Lemma 2. *If $\mathtt{opt}(\mathcal{I}) \le T$, then the total volume of jobs in S is at most $\sum_{I(k,l)\in\mathcal{N}'}(1+\varepsilon)|I(k,l)|$.*

Proof. Suppose $\mathtt{opt}(\mathcal{I}) \le T$. For an interval $I_i(k,l)$, let $I_i^\varepsilon(k,l)$ be the interval of length $(1+\varepsilon)|I_i(k,l)|$ which starts at the same time as $I(k,l)$. It is easy to check that if $I(k',l') \subseteq I(k,l)$, then $I^\varepsilon(k',l') \subseteq I^\varepsilon(k,l)$.

Let $j \in S$ be a job of type (k,l). The off-line optimal solution must schedule it within $\frac{T}{2^k}$ of its release date. Since $r_j \in I(k,l)$, the optimal solution must process a job j during $I^\varepsilon(k,l)$. So, the total volume of jobs in S can be at most

$$\left|\bigcup_{I(k,l)\in\mathcal{N}} I^\varepsilon(k,l)\right| = \left|\bigcup_{I(k,l)\in\mathcal{N}'} I^\varepsilon(k,l)\right| \le \sum_{I(k,l)\in\mathcal{N}'} |I^\varepsilon(k,l)| = \sum_{I(k,l)\in\mathcal{N}'} (1+\varepsilon)|I(k,l)|. \qquad \square$$

Clearly, Corollary 1 contradicts Lemma 2. So, algorithm $\mathcal{A}$ must be able to process all the jobs.

2.2 Off-Line to On-Line

Now, we give an on-line algorithm for the instance $\mathcal{I}$. Recall that $\mathcal{A}$ is an off-line algorithm for $\mathcal{I}$ and may not even respect release dates. The on-line algorithm $\mathcal{B}$ is a non-migratory algorithm which maintains a queue for each machine i and time t. For each job j, it uses $\mathcal{A}$ to figure out which machine the job j gets dispatched to.

Note that the algorithm $\mathcal{A}$ can be implemented in a manner such that for any job j of type (k,l), the slots assigned by $\mathcal{A}$ to j are known by the end of interval $I(k,l)$ — jobs which get released after $I(k,l)$ do not affect the schedule of j. Also note that the release date of j falls in $I(k,l)$. This is described more formally as follows.

Algorithm $\mathcal{A}(\mathcal{I},T)$:

For $t = 0, 1, 2, \ldots$
 For $k = 1, 2, \ldots$
  If t is the end-point of an interval $I(k,l)$ for some l, then
   For each job j of type (k,l)
    For $i = m_j$ downto 1 (m_j is the slowest machine on which j is valid)
     If there are at least $p_j s_i$ free slots on machine i during $I(k,l)$ then
      schedule j on i during the first such free slots (without caring about r_j)

We now describe the algorithm $\mathcal{B}$. It maintains a queue of jobs for each machine. For each job j of class k and releasing during $I(k,l)$, if j gets processed on machine i by $\mathcal{A}$, then $\mathcal{B}$ adds j to the queue of i at end of $I(k,l)$. Observe that $\mathcal{B}$ respects release dates of jobs — a job j of type (k,l) has release date in $I(k,l)$, but it gets dispatched to a machine at the end of the interval $I(k,l)$. For each machine i, $\mathcal{B}$ prefers jobs of higher class, and within a particular class, it follows the ordering given by $\mathcal{A}$ (or it could just go by release dates). Further, we give machines in $\mathcal{B}$ $(1+3\varepsilon)$-speedup.

Analysis. We now analyze $\mathcal{B}$. For a class k, let $J_{\geq k}$ be the jobs of class at least k. For a class k, integer l and machine i, let $Q(i,k,l)$ denote the jobs of $J_{\geq k}$ which are in the queue of machine i at the beginning of $I(k,l)$. First we note some properties of $\mathcal{B}$:

(i) A job j gets scheduled in $\mathcal{B}$ only in later slots than those it was scheduled on by $\mathcal{A}$: A job j of type (k,l) gets scheduled during $I(k,l)$ in $\mathcal{A}$. However, it gets added to the queue of a machine by $\mathcal{B}$ only at the end of $I(k,l)$.

(ii) For a class k, integer l and machine i, the total remaining processing time (on the machine i) of jobs in $Q(i,k,l)$ is at most $\frac{(1+2\varepsilon)T}{\varepsilon 2^k}$: Suppose this is true for some i,k,l. We want to show that this holds for $i,k,l+1$ as well.

The jobs in the queue $Q(i,k,l+1)$ could consist of either (i) the jobs in $Q(i,k,l)$, or (ii) the jobs of $J_{\geq k}$ which get processed by $\mathcal{A}$ during $I_i(k,l)$. Indeed, jobs of $J_{\geq k}$ which get released before the the interval $I_i(k,l)$ finish before this interval begins (in $\mathcal{A}$). Hence, in $\mathcal{B}$, any such job would either finish before $I(k,l)$ begins, or will be in the queue $Q(i,k,l)$. The jobs of $J_{\geq k}$ which get released during $I(k,l)$ will complete processing in this interval (in $\mathcal{A}$) and hence may get added to the queue $Q(i,k,l+1)$.
Now, the total processing time of the jobs in (ii) above would be at most $(1+2\varepsilon)|I(k,l)|$ (recall that the machines in $\mathcal{A}$ have speedup of $(1+2\varepsilon)$). Suppose in the schedule $\mathcal{B}$, the machine i processes a job of class greater than k during some time in $I_i(k,l)$ — then it must have finished processing all the jobs in $Q(i,k,l)$, and so $Q(i,k,l+1)$ can only contain jobs from (ii) above, and hence, their total processing time is at most $(1+2\varepsilon)|I(k,l)|$ and we are done. If the machine i is busy during $I_i(k,l)$ processing jobs from $J_{\geq k}$ (in $\mathcal{B}$), then it does at least $(1+2\varepsilon)|I(k,l)|$ amount of processing , and so, the property holds at the end of $I(k,l)$ as well.

We are now ready to prove the main theorem.

Theorem 1. *In the schedule $\mathcal{B}$, a job j of class k has flow-time at most $\frac{T(1+3\varepsilon)}{\varepsilon^2 2^k}$. Hence, for any instance, $\mathcal{B}$ is an $\left(\frac{2(1+3\varepsilon)}{\varepsilon^2}\right)$-competitive algorithm with $(1+3\varepsilon)$-speedup.*

Proof. Consider a job j of class type (k,l). Suppose it gets processed on machine i. The algorithm $\mathcal{B}$ adds j to the queue $Q(i,k,l)$. Property (ii) above implies that the total remaining processing time of these jobs (on i) is at most $(1+2\varepsilon)|I(k,l)|$. Consider an interval I which starts at the beginning of $I(k,l)$ and has length $\frac{(1+2\varepsilon)|I(k,l)|}{\varepsilon} = \frac{(1+2\varepsilon)T}{\varepsilon^2 2^k}$. The jobs of $J_{\geq k}$ that $\mathcal{B}$ can process on i during I are either (i) jobs in $Q(i,k,l)$, or (ii) jobs processed by $\mathcal{A}$ on machine i during I (using property (i) above). The total processing time of the jobs in (ii) is at most $(1+2\varepsilon)|I|$, whereas $\mathcal{B}$ can process $(1+3\varepsilon)|I|$ volume during I (on machine i). This still leaves us with $\varepsilon|I| = \frac{(1+2\varepsilon)T}{\varepsilon 2^k}$ — this is enough to process all the jobs in $Q(i,k,l)$. So the flow-time of j is at most $|I| + |I(k,l)| = \frac{T}{2^k}\left(\frac{1}{\varepsilon} + \frac{1+2\varepsilon}{\varepsilon^2}\right)$. Finally, given any instance, we lose an extra factor of 2 in the competitive ratio because we scale all weights to powers of 2. □

Extensions. We mention some easy extensions of the result above.

Comparison with migratory off-line optimum: Here, we allow the off-line optimum to migrate jobs across machines. To deal with this, we modify the definition of when a job is valid on a machine. We will say that a job j of class k is valid for a machine i if its processing time on i is at most $\frac{T}{2^k} \cdot \frac{1+\varepsilon}{\varepsilon}$. Note that even a migratory algorithm will process at most $\frac{\varepsilon}{1+\varepsilon}$-fraction of a job on machines which are not valid for it. Further, we modify the definition of $I(l,k)$ to be $\left[\frac{(1+\varepsilon)lT}{\varepsilon^2 2^k}, \frac{(1+\varepsilon)(l+1)T}{\varepsilon^2 2^k}\right)$. The rest of the analysis can be carried out as above.

We can show that the on-line algorithm is $O\left(\frac{(1+\varepsilon)^2}{\varepsilon^3}\right)$-competitive with $(1+\varepsilon)$-speedup.

Deadline scheduling on related machines: In this setting, the input instance also comes with deadline d_j for each job j. The assumption is that there is a schedule (off-line) which can schedule all jobs (with migration) such that each job finishes before its deadline. The question is: is there a constant s and an on-line algorithm $\mathcal{S}$ such that with speedup s, it can meet all the deadlines? Using the above result, it is easy to show that our online algorithm has this property provided we give it constant speedup. We give the proof in the full version of the paper.

Corollary 2. *There is a constant s, and a non-migratory scheduling algorithm which, given any instance of the deadline scheduling problem, completes all the jobs within their deadline if we give speed-up of c to all the machines.*

So far our on-line algorithm has assumed that we know the optimal value of an instance. In the full version of this paper, we show how to get rid of this assumption.

3 Max-Flow-time on Unrelated Machines

We consider the (unweighted) Max-Flow-time on unrelated machines. We first show that a constant competitive algorithm cannot have the property of immediate dispatch and it requires speed augmentation. Since our instances use unit-sized jobs, the lower bound also holds for Max-Stretch. Recall that a scheduling algorithm is called *immediate dispatch* if it decides, at the time of a job's arrival, which machine to schedule the job on.

The lower bound for an immediate dispatch algorithm follows from the lower bound of Azar et al. [4] for minimizing total load in the subset parallel settings. Here, we are given a set of machines, and jobs arrive in a sequence. Each job specifies a subset of machines it can go to, and the on-line algorithm needs to dispatch a job on its arrival to one such machine. The goal is to minimize the maximum number of jobs which get dispatched to a machine. Azar et al. [4] prove that any randomized on-line algorithm for this problem is $\Omega(\log m)$-competitive. From this result, we can easily deduce the following lower bound for Max-Flow-time in the subset parallel setting with unit size jobs (given an instance of the load balancing problem, give each job size of 1 unit, and make them arrive at time 0 in the same sequence as in this given instance).

Theorem 2. *Any immediate dispatch randomized on-line algorithm for* Max-Flow-time *in the subset parallel setting with unit job sizes must have competitive ratio of $\Omega(\log m)$.*

Any randomized on-line algorithm with bounded competitive ratio needs speed augmentation. We give the proof in the full version of the paper.

Theorem 3. *Any online algorithm for minimizing* Max-Flow-time *on subset-parallel machines which allows non-immediate dispatch but does not allow speed augmentation has a competitive ratio of $\Omega(m)$. This holds even for unit-sized jobs.*

3.1 A $(1+\varepsilon, O(1/\varepsilon))$-Competitive Algorithm

We now describe an $\left(\frac{2}{\varepsilon}\right)$-competitive algorithm for Max-Flow-time on multiple unrelated machines with $(1+\varepsilon)$-speed augmentation. The algorithm proceeds in several phases: denote these by $\Pi_1, \Pi_2, \ldots$, where phase Π_i begins at time t_{i-1} and ends at time t_i. In phase Π_i, we will schedule all jobs released during the phase Π_{i-1}.

In the initial phase, Π_1, we consider the jobs released at time $t_0 = 0$, and find an optimal schedule which minimizes the makespan of jobs released at time t_0. This phase ends at the time we finish processing all these jobs. Now, suppose we have defined $\Pi_1, \ldots, \Pi_l$, and have scheduled jobs released during $\Pi_1, \ldots, \Pi_{l-1}$. We consider the jobs released during Π_l, and starting from time t_l, we find a schedule which minimized their makespan (assuming all of these jobs are released at time t_l). Again, this phase ends at the time we finish processing all these jobs. Note that this algorithm is a non-immediate dispatch algorithm and does not require migration. We now prove that this algorithm has the desired properties.

Theorem 4. *Assuming $\varepsilon \leq 1$, The algorithm described above has competitive ratio $\frac{2}{\varepsilon}$ with $(1+\varepsilon)$-speed augmentation.*

Proof. Consider an instance $\mathcal{I}$ and assume that the optimal off-line schedule has maximum flow time of T. We will be done if we show that each of the phases Π_i has length at most $\frac{T}{\varepsilon}$. For Π_1, this is true because all the jobs released at time 0 can be scheduled within T units of time. Suppose this is true for phase Π_i. Now, we know that the jobs released during Π_i can be scheduled in an interval of length $\Pi_i + T$. Using $(1+\varepsilon)$-speed augmentation, the length of the next phase is at most

$$\frac{|\Pi_i| + T}{1+\varepsilon} \leq \frac{T/\varepsilon + T}{1+\varepsilon} = \frac{T}{\varepsilon}.$$

□

4 Max-Weighted-Flow-time on Unrelated Machines

In this section, given any constant speedup, any on-line algorithm for Max-Weighted-Flow-time on unrelated machines is $\Omega(\log m)$-competitive. This bound holds for the special case of subset parallel model, and even extends to the Max-Stretch metric. We give the proof of the following theorem in the full version of the paper.

Theorem 5. *Given any large enough parameter c, integer $s \geq 1$, and an on-line algorithm $\mathcal{A}$ which is allowed speedup of $(s+1)/2$, there exists an instance $\mathcal{I}(s,c)$ of* Max-Weighted-Flow-time *on subset parallel machines such that $\mathcal{A}$ is not c-competitive on $\mathcal{I}(s,c)$. The instance $\mathcal{I}(s,c)$ has jobs with s different weights only, and uses $(O(s))^{O(cs^2)}$ machines.*

5 Lower Bound for L_p Norm of Stretch

We show a lower bound for the competitive ratio for the L_p-norm of the stretches, with speed augmentation by a factor of $1+\varepsilon$. We assume that there is an online algorithm with competitive ratio $c = o(\frac{p}{\varepsilon^{1-3/p}})$ and derive a contradiction.

The construction uses $m = 2^p$ machines. We start with the typical construction to get a large load on one machine. For this we consider 2 machines. At time 0 we release two jobs of size 1 (and weight 1) - each can go on exactly one machine. Then until time 1 we release tiny jobs, i.e., at each δ time step a job of size δ (and weight $1/\delta$) is released that can go on any of the two machines. Note that at time 1 at least one of the machines has load (of size 1 jobs) at least $1/2 - \varepsilon - c\delta$. This is because, the total volume of jobs is 3, the two machines can process at most $2(1+\varepsilon)$ units, and all tiny jobs except the last c have to be processed. It makes sense to set $\delta = \varepsilon/c$ and hence $c\delta \leq \varepsilon$.

Now, we can use this as a gadget, starting with $m/2$ pairs of machines we then take the $m/2$ machines with large load and pair them up arbitrarily and recursively do the same construction. We end up with one machine with load $\Omega(\log m)$ (if ε is sufficiently smaller than $1/2$). This concludes the first of two phases.

Now that we have a machine with large load, we release tiny jobs for a time interval of length $\log(m)/\varepsilon$. Since the tiny jobs have to be processed first, the initial load of $\Omega(\log m)$ needs time $\Omega(\log(m)/\varepsilon)$ to be fully processed, as it can be processed only in the time that we have additional due to resource augmentation. Hence, at least one size 1 job has stretch at least $\Omega(\log(m)/\varepsilon)$. This concludes the second phase.

Let us bound the number of jobs k that we release in these 2 phases. In the first phase of the construction we release $m + m/2 + m/4 + \ldots = O(m)$ jobs of size 1 and $O(m/\delta)$ tiny jobs. In the second phase we release $O(\log(m)/(\varepsilon\delta))$ tiny jobs. Thus, $k = O(m/\delta + \log(m)/(\varepsilon\delta))$. Note that we can bound $1/\delta \leq p/\varepsilon^{2-3/p}$ and hence $k = O(mp/\varepsilon^{3-3/p})$.

We want to repeat these two phases n/k times. After the first 2 phases have been completed (by the optimal offline algorithm) we release again the 2 phases, and we repeat this n/k times. Thus, for the optimal offline algorithm all repetitions will be independent. Then in total we released any desired number n of jobs, where $n \geq k$.

Note that the optimal offline algorithm would have a max-stretch of 2 and, thus, also an L_p norm of the stretches of $\left(\frac{1}{n}\sum_i v_i^p\right)^{1/p} \leq 2$.

We now lower bound the L_p norm of the stretches of the online algorithm. We already have a lower bound on the maximal stretch of any job, $\Omega(\log(m)/\varepsilon)$, and we know that there are at least n/k jobs with such a large stretch, one for each repetition of the 2 phases. Now, let v_i be the stretch of the i-th job. Then the L_p norm of the stretches is

$$c \geq \Omega\left(\frac{1}{n}\sum_i v_i^p\right)^{1/p}$$

Since we know that there are n/k jobs with $v_i = \Omega(\log(m)/\varepsilon)$ this is at least

$$c \geq \Omega\left(\frac{\log(m)}{\varepsilon}\left(\frac{n/k}{n}\right)^{1/p}\right) = \Omega\left(\frac{\log(m)}{\varepsilon k^{1/p}}\right).$$

Plugging in our bound on $k = O(mp/\varepsilon^{3-3/p})$ this yields a bound of

$$c \geq \Omega\left(\frac{\log(m)}{\varepsilon(mp)^{1/p}/\varepsilon^{3/p}}\right).$$

Since $m = 2^p$ and noting that $p^{1/p} = O(1)$ this yields the desired contradiction to c begin too small, $c \geq \Omega\left(\frac{p}{\varepsilon^{1-3/p}}\right)$. The only condition for this was $n \geq k = \frac{2^{\Theta(p)}}{\varepsilon^{\Theta(1)}}$, which implies that n just has to be sufficiently large.

References

1. Ambühl, C., Mastrolilli, M.: On-line scheduling to minimize max flow time: An optimal preemptive algorithm. Oper. Res. Lett. 33(6), 597–602 (2005)
2. Anand, S., Garg, N., Kumar, A.: Resource augmentation for weighted flow-time explained by dual fitting. In: 23rd Symp. Discrete Algorithms (SODA), pp. 1228–1241 (2012)
3. Anand, S., Garg, N., Megow, N.: Meeting deadlines: How much speed suffices? In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 232–243. Springer, Heidelberg (2011)
4. Azar, Y., Naor, J., Rom, R.: The competitiveness of on-line assignments. J. Algorithms 18(2), 221–237 (1995)
5. Bansal, N., Pruhs, K.: Server scheduling in the ℓ_p norm: A rising tide lifts all boats. In: 35th Symp. Theory of Computing (STOC), pp. 242–250 (2003)
6. Bansal, N., Pruhs, K.: Server scheduling in the weighted ℓ_p norm. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 434–443. Springer, Heidelberg (2004)
7. Bender, M.A., Chakrabarti, S., Muthukrishnan, S.: Flow and stretch metrics for scheduling continuous job streams. In: 9th Symp. Discrete Algorithms (SODA), pp. 270–279 (1998)
8. Bender, M.A., Muthukrishnan, S., Rajaraman, R.: Improved algorithms for stretch scheduling. In: 13th Symp. Discrete Algorithms (SODA), pp. 762–771 (2002)
9. Chekuri, C., Moseley, B.: Online scheduling to minimize the maximum delay factor. In: 20th Symp. Discrete Algorithms (SODA), pp. 1116–1125 (2009)
10. Golovin, D., Gupta, A., Kumar, A., Tangwongsan, K.: All-norms and all-ℓ_p-norms approximation algorithms. In: 28th Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 199–210 (2008)
11. Im, S., Moseley, B.: An online scalable algorithm for minimizing ℓ_k-norms of weighted flow time on unrelated machines. In: 22nd Symp. Discrete Algorithms (SODA), pp. 95–108 (2011)
12. Phillips, C.A., Stein, C., Torng, E., Wein, J.: Optimal time-critical scheduling via resource augmentation. Algorithmica 32(2), 163–200 (2002)

Tight Lower Bound for Linear Sketches of Moments

Alexandr Andoni[1], Huy L. Nguyễn[2], Yury Polyanskiy[3], and Yihong Wu[4]

[1] Microsoft Research SVC
andoni@microsoft.com
[2] Princeton U
hlnguyen@princeton.edu
[3] MIT
yp@mit.edu
[4] UIUC
yihongwu@illinois.edu

Abstract. The problem of estimating frequency moments of a data stream has attracted a lot of attention since the onset of streaming algorithms [AMS99]. While the space complexity for approximately computing the p^{th} moment, for $p \in (0, 2]$ has been settled [KNW10], for $p > 2$ the exact complexity remains open. For $p > 2$ the current best algorithm uses $O(n^{1-2/p} \log n)$ words of space [AKO11,BO10], whereas the lower bound is of $\Omega(n^{1-2/p})$ [BJKS04].

In this paper, we show a tight lower bound of $\Omega(n^{1-2/p} \log n)$ words for the class of algorithms based on linear sketches, which store only a sketch Ax of input vector x and some (possibly randomized) matrix A. We note that all known algorithms for this problem are linear sketches.

1 Introduction

One of the classical problems in the streaming literature is that of computing the p-frequency moments (or p-norm) [AMS99]. In particular, the question is to compute the norm $\|x\|_p$ of a vector $x \in \mathbb{R}^n$, up to $1 + \epsilon$ approximation, in the streaming model using low space. Here, we assume the most general model of streaming, where one sees updates to x of the form (i, δ_i) which means to add a quantity $\delta_i \in \mathbb{R}$ to the coordinate i of x.[1] In this setting, linear estimators, which store Ax for a matrix A, are particularly useful as such an update can be easily processed due to the equality $A(x + \delta_i e_i) = Ax + A(\delta_i e_i)$.

The frequency moments problem is among the problems that received the most attention in the streaming literature. For example, the space complexity for $p \leq 2$ has been fully understood. Specifically, for $p = 2$, the foundational paper of [AMS99] showed that $O_\epsilon(1)$ words (linear measurements) suffice to approximate

[1] For simplicity of presentation, we assume that $\delta_i \in \{-n^{O(1)}, \ldots, n^{O(1)}\}$, although more refined bounds can be stated otherwise. Note that in this case, a "word" (or measurement in the case of linear sketch — see definition below) is usually $O(\log n)$ bits.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 25–32, 2013.

the Euclidean norm[2]. Later work showed how to achieve the same space for all $p \in (0, 2)$ norms [Ind06,Li08,KNW10]. This upper bound has a matching lower bound [AMS99,IW03,Bar02,Woo04]. Further research focused on other aspects, such as algorithms with improved update time (time to process an update (i, δ_i)) [NW10,KNW10,Li08,GC07,KNPW11].

In constrast, when $p > 2$, the exact space complexity still remains open. After a line of research on both upper bounds [AMS99,IW05,BGKS06,MW10], [AKO11,BO10,Gan11] and lower bounds [AMS99,CKS03,BJKS04,JST11,PW12], we presently know that the best space upper bound is of $O(n^{1-2/p} \log n)$ words, and the lower bound is $\Omega(n^{1-2/p})$ bits (or linear measurements). (Very recently also, in a *restricted* streaming model — when $\delta_i = 1$ — [BO12] achieves an improved upper bound of nearly $O(n^{1-2/p})$ words.) In fact, since for $p = \infty$ the right bound is $O(n)$ (without the log factor), it may be tempting to assume that there the right upper bound should be $O(n^{1-2/p})$ in the general case as well.

In this work, we prove a tight lower bound of $\Omega(n^{1-2/p} \log n)$ for the case of *linear estimator*. A linear estimator uses a distribution over $m \times n$ matrices A such that with high probability over the choice of A, it is possible to calculate the p^{th} moment $\|x\|_p$ from the sketch Ax. The parameter m, the number of words used by the algorithm, is also called the number of measurements of the algorithm. Our new lower bound is of $\Omega(n^{1-2/p} \log n)$ measurements/words, which matches the upper bound from [AKO11,BO10]. We stress that essentially all known algorithms in the general streaming model are in fact linear estimators.

Theorem 1. *Fix $p \in (2, \infty)$. Any linear sketching algorithm for approximating the p^{th} moment of a vector $x \in \mathbb{R}^n$ up to a multiplicative factor 2 with probability 99/100 requires $\Omega(n^{1-2/p} \log n)$ measurements.*

In other words, for any $p \in (2, \infty)$ there is a constant C_p such that for any distribution on $m \times n$ matrices A with $m < C_p n^{1-2/p} \log n$ and any function $f : \mathbb{R}^{m \times n} \times \mathbb{R}^m \to \mathbb{R}_+$ we have

$$\inf_{x \in \mathbb{R}^n} \Pr\left(\frac{1}{2}\|x\|_p \le f(A, Ax) \le 2\|x\|_p\right) \le \frac{99}{100}. \tag{1}$$

The proof uses similar hard distributions as in some of the previous work, namely all coordinates of an input vector x have random small values except for possibly one location. To succeed on these distributions, the algorithm has to distinguish between a mixture of Gaussian distributions and a pure Gaussian distribution. Analyzing the optimal probability of success directly seems too difficult. Instead, we use the χ^2-divergence to bound the success probability, which turns out to be much more amenable to analysis.

From a statistical perspective, the problem of linear sketches of moments can be recast as a minimax statistical estimation problem where one observes the pair (Ax, A) and produces an estimate of $\|x\|_p$. More specifically, this is a functional estimation problem, where the goal is to estimation some functional

[2] The exact bound is $O(1/\epsilon^2)$ words; since in this paper we concentrate on the case of $\epsilon = \Omega(1)$ only, we drop dependence on ϵ.

(in this case, the p^{th} moment) of the parameter x instead of estimating x directly. Under this decision-theoretic framework, our argument can be understood as Le Cam's two-point method for deriving minimax lower bounds [LC86]. The idea is to use a binary hypotheses testing argument where two priors (distributions of x) are constructed, such that 1) the p^{th} moment of x differs by a constant factor under the respective prior; 2) the resulting distributions of the sketches Ax are indistinguishable. Consequently there exists no moment estimator which can achieve constant relative error. This approach is also known as the method of fuzzy hypotheses [Tsy09, Section 2.7.4]. See also [BL96,IS03,Low10,CL11] for the method of using χ^2-divergence in minimax lower bound.

We remark that our proof does not give a lower bound as a function of ϵ (but [Woo13] independently reports progress on this front).

1.1 Preliminaries

We use the following definition of divergences.

Definition 1. *Let P and Q be probability measures. The* χ^2-divergence *from P to Q is*

$$\begin{aligned}\chi^2(P||Q) &\triangleq \int \left(\frac{\mathrm{d}P}{\mathrm{d}Q} - 1\right)^2 \mathrm{d}Q \\ &= \int \left(\frac{\mathrm{d}P}{\mathrm{d}Q}\right)^2 \mathrm{d}Q - 1\end{aligned}$$

The total variation distance *between P and Q is*

$$V(P,Q) \triangleq \sup_A |P(A) - Q(A)| = \frac{1}{2}\int |\mathrm{d}P - \mathrm{d}Q| \tag{2}$$

The operational meaning of the total variation distance is as follows: Denote the optimal sum of Type-I and Type-II error probabilities of the binary hypotheses testing problem $H_0 : X \sim P$ versus $H_1 : X \sim Q$ by

$$\mathcal{E}(P,Q) \triangleq \inf_A \{P(A) + Q(A^{\mathrm{c}})\}, \tag{3}$$

where the infimum is over all measurable sets A and the corresponding test is to declare H_1 if and only if $X \in A$. Then

$$\mathcal{E}(P,Q) = 1 - V(P,Q). \tag{4}$$

The total variation and the χ^2-divergence are related by the following inequality [Tsy09, Section 2.4.1]:

$$2V^2(P,Q) \le \log(1 + \chi^2(P||Q)) \tag{5}$$

Therefore, in order to establish that two hypotheses cannot be distinguished with vanishing error probability, it suffices to show that the χ^2-divergence is bounded.

One additional fact about V and χ^2 is the data-processing property [Csi67]: If a measurable function $f : A \to B$ carries probability measure P on A to P' on B, and carries Q to Q' then

$$V(P, Q) \geq V(P', Q') \,. \tag{6}$$

2 Lower Bound Proof

In this section we prove Theorem 1 for arbitrary fixed measurement matrix A. Indeed, by Yao's minimax principle, we only need to demonstrate an input distribution and show that any deterministic algorithm succeeding on this distribution with probability 99/100 must use $\Omega(n^{1-2/p} \log n)$ measurements.

Fix $p \in (2, \infty)$. Let $A \in \mathbb{R}^{m \times n}$ be a fixed matrix which is used to produce the linear sketch, where $m < C_p n^{1-2/p} \log n$ is the number of measurements and C_p is to be specified. Next, we construct distributions D_1 and D_2 for x to fulfill the following properties:

1. $\|x\|_p \leq Cn^{1/p}$ on the entire support of D_1, and $\|x\|_p \geq 4Cn^{1/p}$ on the entire support of D_2, for some appropriately chosen constant C.
2. Let E_1 and E_2 denote the distribution of Ax when x is drawn from D_1 and D_2 respectively. Then $V(E_1, E_2) \leq 98/100$.

The above claims immediately imply the desired (1) via the relationship between statistical tests and estimators. To see this, note that any moment estimator f induces a test for distinguishing E_1 versus E_2: declare D_2 if and only if $\frac{f(A,Ax)}{2Cn^{1/p}} \geq 1$. In other words,

$$\begin{aligned}
&\Pr_{x \sim \frac{1}{2}(D_1 + D_2)} \left(\frac{1}{2}\|x\|_p \leq f(A, Ax) \leq 2\|x\|_p \right) \\
&\leq \frac{1}{2} \Pr_{x \sim D_2} \left(f(A, Ax) > 2Cn^{1/p} \right) + \frac{1}{2} \Pr_{x \sim D_1} \left(f(A, Ax) \leq 2Cn^{1/p} \right) \qquad (7) \\
&\leq \frac{1}{2}(1 + V(E_1, E_2)) \leq \frac{99}{100} \,, \qquad (8)
\end{aligned}$$

where the last line follows from the characterization of the total variation in (2).

The idea for constructing the desired pair of distributions is to use the Gaussian distribution and its sparse perturbation. Since the moment of a Gaussian random vector takes values on the entire $\mathbb{R}_+$, we need to further truncate by taking its conditioned version. To this end, let $y \sim N(0, I_n)$ be a standard normal random vector and t a random index uniformly distributed on $\{1, \ldots, n\}$ and independently of y. Let $\{e_1, \ldots, e_n\}$ denote the standard basis of $\mathbb{R}^n$. Let $\bar{D}_1$ and $\bar{D}_2$ be input distributions defined as follows: Under the distribution $\bar{D}_1$, we let the input vector x equal to y. Under the distribution $\bar{D}_2$, we add a one-sparse perturbation by setting $x = y + C_1 n^{1/p} e_t$ with an appropriately chosen constant C_1. Now we set D_1 to be $\bar{D}_1$ conditioned on the event

$E = \{z : \|z\|_p \le Cn^{1/p}\}$, i.e., $D_1(\cdot) = \frac{\bar{D}_1(\cdot \cap E)}{\bar{D}_1(E)}$, and set D_2 to be $\bar{D}_2$ conditioned on the event $F = \{z : \|z\|_p \ge 4Cn^{1/p}\}$. By the triangle inequality,

$$\begin{aligned} V(E_1, E_2) &\le V(\bar{E}_1, \bar{E}_2) + V(\bar{E}_1, E_1) + V(\bar{E}_2, E_2) \\ &\le V(\bar{E}_1, \bar{E}_2) + V(\bar{D}_1, D_1) + V(\bar{D}_2, D_2) \\ &= V(\bar{E}_1, \bar{E}_2) + \Pr_{x \sim \bar{D}_1}(\|x\|_p \ge Cn^{1/p}) + \Pr_{x \sim \bar{D}_2}(\|x\|_p \le 4Cn^{1/p}), \end{aligned} \quad (9)$$

where the second inequality follows from the data-processing inequality (6) (applied to the mapping $x \mapsto Ax$). It remains to bound the three terms in (9).

First observe that for any i, $\mathbb{E}[|y_i|^p] = t_p$ where $t_p = 2^{p/2}\Gamma(\frac{p+1}{2})\pi^{-1/2}$. Thus, $\mathbb{E}[\|y\|_p^p] = nt_p$. By Markov inequality, $\|y\|_p^p \ge 100nt_p$ holds with probability at most $1/100$. Now, if we set

$$C_1 = 4 \cdot (100t_p)^{1/p} + 10, \quad (10)$$

we have $(y_t + C_1 n^{1/p})^p > 4^p \cdot 100nt_p$ with probability at least $99/100$, and hence the third term in (9) is also smaller than $\frac{1}{100}$. It remains to show that $V(\bar{E}_1, \bar{E}_2) \le 96/100$.

Without loss of generality, we assume that the rows of A are orthonormal since we can always change the basis of A after taking the measurements. Let ϵ be a constant smaller than $1 - 2/p$. Assume that $m < \frac{\epsilon}{100C_1^2} \cdot n^{1-2/p} \log n$. Let A_i denote the i^{th} column of A. Let S be the set of indices i such that $\|A_i\|_2 \le 10\sqrt{m/n} \le n^{-1/p}\sqrt{\epsilon \log n}/C_1$. Let $\bar{S}$ be the complement of S. Since $\sum_{i=1}^n \|A_i\|_2^2 = m$, we have $|\bar{S}| \le n/100$. Let s be uniformly distributed on S and $\tilde{E}_2$ the distribution of $y + C_1 n^{1/p} e_s$. By the convexity of $(P, Q) \mapsto V(P, Q)$ and the fact that $V(P, Q) \le 1$, we have $V(\bar{E}_1, \bar{E}_2) \le V(\bar{E}_1, \tilde{E}_2) + \frac{|\bar{S}|}{n} \le V(\bar{E}_1, \tilde{E}_2) + 1/100$. In view of (5), it suffices to show that

$$\chi^2(\tilde{E}_2 \| \bar{E}_1) \le c \quad (11)$$

for some sufficiently small constant c. To this end, we first prove a useful fact about the measurement matrix A.

Lemma 1. *For any matrix A with $m < \frac{\epsilon}{100C_1^2} \cdot n^{1-2/p} \log n$ orthonormal rows, denote by S the set of column indices i such that $\|A_i\|_2 \le 10\sqrt{m/n}$. Then*

$$|S|^{-2} \sum_{i,j \in S} e^{C_1^2 n^{2/p} \langle A_i, A_j \rangle} \le 1.03C_1^4(n^{-2+4/p+\epsilon}m + n^{2/p-1}\sqrt{m}) + 1$$

Proof. Because $AA^T = I_m$, we have

$$\sum_{i,j \in [n]} \langle A_i, A_j \rangle^2 = \sum_{i,j \in [n]} (A^T A)_{ij}^2 = \|A^T A\|_F^2 = \mathrm{tr}(A^T A A^T A) = \mathrm{tr}(A^T A) = \|A\|_F^2 = m.$$

We consider the following relaxation: let $x_1, \ldots, x_{|S|^2} \geq 0$ where $\sum_i x_i^2 \leq C_1^4 n^{4/p}$. m and $x_i \leq \epsilon \log n$. We now upper bound $|S|^{-2} \sum_{i=1}^{|S|^2} e^{x_i}$. We have

$$\begin{aligned}
|S|^{-2} \sum_{i=1}^{|S|^2} e^{x_i} &= |S|^{-2} \sum_{i=1}^{|S|^2} \left(1 + x_i + \sum_{j \geq 2} \frac{x_i^j}{j!}\right) \\
&\leq 1 + |S|^{-2} \sum_{i=1}^{|S|^2} x_i + |S|^{-2} \sum_{i=1}^{|S|^2} x_i^2 \sum_{j \geq 2} \frac{(\max_{i \in [n^2]} x_i)^{j-2}}{j!} \\
&\leq 1 + |S|^{-2} \sqrt{|S|^2 \sum_i x_i^2} + |S|^{-2} (C_1^4 m n^{4/p}) \left(\frac{e^{\epsilon \log n}}{(\epsilon \log n)^2}\right) \\
&\leq 1 + 1.03 C_1^2 \sqrt{m} n^{2/p-1} + 1.03 C_1^4 n^{-2+4/p+\epsilon} m.
\end{aligned}$$

The last inequality uses the fact that $99n/100 \leq |S| \leq n$. Applying the above upper bound to $x_{(i-1)|S|+j} = C_1^2 n^{2/p} |\langle A_i, A_j \rangle| \leq C_1^2 n^{2/p} \|A_i\| \cdot \|A_j\| \leq \epsilon \log n$, we conclude the lemma.

We also need the following lemma [IS03, p. 97] which gives a formula for the χ^2-divergence from a Gaussian location mixture to a standard Gaussian distribution:

Lemma 2. *Let P be a distribution on $\mathbb{R}^m$. Then*

$$\chi^2(N(0, I_m) * P \,||\, N(0, I_m)) = \mathbb{E}[\exp(\langle X, X' \rangle)] - 1\,,$$

where X and X' are independently drawn from P.

We now proceed to proving an upper bound on the χ^2-divergence between $\bar{E}_1$ and $\tilde{E}_2$.

Lemma 3.

$$\chi^2(\tilde{E}_2 \| \bar{E}_1) \leq 1.03 C_1^4 (n^{-2+4/p+\epsilon} m + n^{2/p-1} \sqrt{m})$$

Proof. Let $p_i = 1/|S| \ \forall i \in S$ be the probability $t = i$. Recall that s is the random index uniform on the set $S = \{i \in [n] : \|A_i\|_2 \leq 10\sqrt{m/n}\}$. Note that $Ay \sim N(0, AA^T)$. Since $AA^T = I_m$, we have $\bar{E}_1 = N(0, I_m)$. Therefore $A(y + C_1 n^{1/p}) \sim \tilde{E}_2 = \frac{1}{|S|} \sum_{i \in S} N(A_i, I_m)$, a Gaussian location mixture.

Applying Lemma 2 and then Lemma 1, we have

$$\begin{aligned}
\chi^2(\tilde{E}_2 \| \bar{E}_1) &= \sum_{i,j \in S} p_i p_j e^{C_1^2 n^{2/p} \langle A_i, A_j \rangle} - 1 \\
&\leq 1.03 C_1^4 (n^{-2+4/p+\epsilon} m + n^{2/p-1} \sqrt{m}).
\end{aligned}$$

Finally, to finish the lower bound proof, since $\epsilon < 1 - 2/p$ we have $n^{-2+4/p+\epsilon} m + n^{2/p-1} \sqrt{m} = o(1)$, implying (11) for all sufficiently large n and completing the proof of $V(E_1, E_2) \leq 98/100$.

3 Discussions

While Theorem 1 is stated only for constant p, the proof also gives lower bounds for p depending on n. At one extreme, the proof recovers the known lower bound for approximating the ℓ_∞-norm of $\Omega(n)$. Notice that the ratio between the $\ell_{(\ln n)/\varepsilon}$-norm and the ℓ_∞-norm of any vector is bounded by e^ε so it suffices to consider $p = (\ln n)/\varepsilon$ with a sufficiently small constant ε. Applying the Stirling approximation to the crude value of C_1 in the proof, we get $C_1 = \Theta(\sqrt{p})$. Thus, the lower bound we obtain is $\Omega(n^{1-2/p}(\log n)/C_1^2) = \Omega(n)$.

At the other extreme, when $p \to 2$, the proof also gives super constant lower bounds up to $p = 2 + \Theta(\log\log n/\log n)$. Notice that ϵ can be set to $1 - 2/p - \Theta(\log\log n/\log n)$ instead of a positive constant strictly smaller than $1 - 2/p$. For this value of p, the proof gives a polylog(n) lower bound. We leave it as an open question to obtain tight bounds for $p = 2 + o(1)$.

Acknowledgments. HN was supported by NSF CCF 0832797, and a Gordon Wu Fellowship. YP's work was supported by the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370.

References

AKO11. Andoni, A., Krauthgamer, R., Onak, K.: Streaming algorithms from precision sampling. In: Proceedings of the Symposium on Foundations of Computer Science, FOCS (2011), Full version appears on arXiv:1011.1263

AMS99. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. J. Comp. Sys. Sci. 58, 137–147 (1999), Previously appeared in STOC 1996

Bar02. Bar-Yossef, Z.: The complexity of massive data set computations. PhD thesis, UC Berkeley (2002)

BGKS06. Bhuvanagiri, L., Ganguly, S., Kesh, D., Saha, C.: Simpler algorithm for estimating frequency moments of data streams. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 708–713 (2006)

BJKS04. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. J. Comput. Syst. Sci. 68(4), 702–732 (2004)

BL96. Brown, L.D., Low, M.G.: A constrained risk inequality with applications to nonparametric functional estimation. The Annals of Statistics 24, 2524–2535 (1996)

BO10. Braverman, V., Ostrovsky, R.: Recursive sketching for frequency moments. CoRR, abs/1011.2571 (2010)

BO12. Braverman, V., Ostrovsky, R.: Approximating large frequency moments with pick-and-drop sampling. CoRR, abs/1212.0202 (2012)

CKS03. Chakrabarti, A., Khot, S., Sun, X.: Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In: IEEE Conference on Computational Complexity, pp. 107–117 (2003)

CL11. Cai, T.T., Low, M.G.: Testing composite hypotheses, Hermite polynomials and optimal estimation of a nonsmooth functional. The Annals of Statistics 39(2), 1012–1041 (2011)

Csi67. Csiszár, I.: Information-type measures of difference of probability distributions and indirect observations. Studia Sci. Math. Hungar. 2, 299–318 (1967)
Gan11. Ganguly, S.: Polynomial estimators for high frequency moments. arXiv, 1104.4552 (2011)
GC07. Ganguly, S., Cormode, G.: On estimating frequency moments of data streams. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX and RANDOM 2007. LNCS, vol. 4627, pp. 479–493. Springer, Heidelberg (2007)
Ind06. Indyk, P.: Stable distributions, pseudorandom generators, embeddings and data stream computation. J. ACM 53(3), 307–323 (2006); Previously appeared in FOCS 2000
IS03. Ingster, Y.I., Suslina, I.A.: Nonparametric goodness-of-fit testing under Gaussian models. Springer, New York (2003)
IW03. Indyk, P., Woodruff, D.: Tight lower bounds for the distinct elements problem. In: Proceedings of the Symposium on Foundations of Computer Science (FOCS), pp. 283–290 (2003)
IW05. Indyk, P., Woodruff, D.: Optimal approximations of the frequency moments of data streams. In: Proceedings of the Symposium on Theory of Computing, STOC (2005)
JST11. Jowhari, H., Saglam, M., Tardos, G.: Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In: Proceedings of the ACM Symposium on Principles of Database Systems (PODS), pp. 49–58 (2011), Previously `http://arxiv.org/abs/1012.4889`
KNPW11. Kane, D.M., Nelson, J., Porat, E., Woodruff, D.P.: Fast moment estimation in data streams in optimal space. In: Proceedings of the Symposium on Theory of Computing (STOC) (2011); A previous version appeared as ArXiv:1007.4191, `http://arxiv.org/abs/1007.4191`
KNW10. Kane, D.M., Nelson, J., Woodruff, D.P.: On the exact space complexity of sketching small norms. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA (2010)
LC86. Le Cam, L.: Asymptotic methods in statistical decision theory. Springer, New York (1986)
Li08. Li, P.: Estimators and tail bounds for dimension reduction in l_p $(0 < p \leq 2)$ using stable random projections. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA (2008)
Low10. Low, M.G.: Chi-square lower bounds. In: Borrowing Strength: Theory Powering Applications - A Festschrift for Lawrence D. Brown, pp. 22–31 (2010)
MW10. Monemizadeh, M., Woodruff, D.: 1-pass relative-error l_p-sampling with applications. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA (2010)
NW10. Nelson, J., Woodruff, D.: Fast manhattan sketches in data streams. In: Proceedings of the ACM Symposium on Principles of Database Systems, PODS (2010)
PW12. Price, E., Woodruff, D.P.: Applications of the Shannon-Hartley theorem to data streams and sparse recovery. In: Proceedings of the 2012 IEEE International Symposium on Information Theory, pp. 1821–1825 (2012)
Tsy09. Tsybakov, A.B.: Introduction to Nonparametric Estimation. Springer, New York (2009)
Woo04. Woodruff, D.: Optimal space lower bounds for all frequency moments. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA (2004)
Woo13. Woodruff, D.: Personal communication (February 2013)

Optimal Partitioning for Dual Pivot Quicksort*
(Extended Abstract)

Martin Aumüller and Martin Dietzfelbinger

Faculty of Computer Science and Automation, Ilmenau University of Technology,
98694 Ilmenau, Germany
martin.aumueller@tu-ilmenau.de, martin.dietzfelbinger@tu-ilmenau.de

Abstract. *Dual pivot quicksort* refers to variants of classical quicksort where in the partitioning step two pivots are used to split the input into three segments. This can be done in different ways, giving rise to different algorithms. Recently, a dual pivot algorithm due to Yaroslavskiy received much attention, because it replaced the well-engineered quicksort algorithm in Oracle's Java 7 runtime library. Nebel and Wild (ESA 2012) analyzed this algorithm and showed that on average it uses $1.9n \ln n + O(n)$ comparisons to sort an input of size n, beating standard quicksort, which uses $2n \ln n + O(n)$ comparisons. We introduce a model that captures all dual pivot algorithms, give a unified analysis, and identify new dual pivot algorithms that minimize the average number of key comparisons among all possible algorithms up to lower order or linear terms. This minimum is $1.8n \ln n + O(n)$.

1 Introduction

Quicksort [4] is a thoroughly analyzed classical sorting algorithm, described in standard textbooks such as [2,5,9] and with implementations in practically all algorithm libraries. Following the divide-and-conquer paradigm, on an input consisting of n elements quicksort uses a pivot element to partition its input elements into two parts, those smaller than the pivot and those larger than the pivot, and then uses recursion to sort these parts. It is well known that if the input consists of n elements with distinct keys in random order and the pivot is picked by just choosing an element then on average quicksort uses $2n \ln n + O(n)$ comparisons. In 2009, Yaroslavskiy announced[1] that he had found an improved quicksort implementation, the claim being backed by experiments. After extensive empirical studies, in 2009 Yaroslavskiy's algorithm became the new standard quicksort algorithm in Oracle's Java 7 runtime library. This algorithm employs two pivots to split the elements. If two pivots p and q with $p < q$ are used, the partitioning

* For a full version containing all proofs, a discussion of swap strategies, more experiments, and pseudocode of the algorithms, see [1].

[1] An archived version of the relevant discussion in a Java newsgroup can be found at http://permalink.gmane.org/gmane.comp.java.openjdk.core-libs.devel/2628. Also see [10].

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 33–44, 2013.

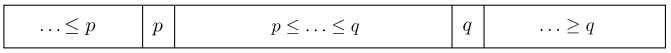


Fig. 1. Result of the partition step in dual pivot quicksort schemes using two pivots p, q with $p \leq q$. All elements $\leq p$ are moved to the left of p; all elements $\geq q$ are moved to the right of q. All other elements lie between p and q.

step partitions the remaining $n-2$ elements into 3 parts: those smaller than p (*small* elements), those in between p and q (*medium* elements), and those larger than q (*large* elements), see Fig. 1.[2] Recursion is then applied to the three parts. As remarked in [10], it came as a surprise that two pivots should help, since in his thesis [6] Sedgewick had proposed and analyzed a dual pivot approach that was inferior to classical quicksort. Later, Hennequin in his thesis [3] studied the general approach of using $k \geq 1$ pivot elements. According to [10], he found only slight improvements that would not compensate for the more involved partitioning procedure. (See [10] for a short discussion.)

In [10], Nebel and Wild formulated and analyzed a simplified version of Yaroslavskiy's algorithm. They showed that it makes $1.9n \ln n + O(n)$ key comparisons on average, in contrast to the $2n \ln n + O(n)$ of standard quicksort and the $\frac{32}{15} n \ln n + O(n)$ of Sedgewick's dual pivot algorithm. On the other hand, they showed that the number of swap operations in Yaroslavskiy's algorithm is $0.6n \ln n + O(n)$ on average, which is much higher than the $0.33n \ln n + O(n)$ swap operations in classical quicksort. In this paper, also following tradition, we concentrate on the comparison count as cost measure and on asymptotic results.

The authors of [10] state that the reason for Yaroslavskiy's algorithm being superior were that his "partitioning method is able to take advantage of certain asymmetries in the outcomes of key comparisons". They also state that "[Sedgewick's dual pivot method] fails to utilize them, even though being based on the same abstract algorithmic idea". So the abstract algorithmic idea of using two pivots can lead to different algorithms with different behavior. In this paper we describe the design space from which all these algorithms originate. We fully explain which simple property makes some dual pivot algorithms perform better and some perform worse w.r.t. the average comparison count and identify optimal members (up to lower order or linear terms) of this design space. The best ones use $1.8n \ln n + O(n)$ comparisons on average—even less than Yaroslavskiy's method.

The first observation is that everything depends on the cost, i.e., the comparison count, of the partitioning step. This is not new at all. Actually, in Hennequin's thesis [3] the connection between partitioning cost and overall cost for quicksort variants with more than one pivot is analyzed in detail. The result relevant for us is that if two pivots are used and the (average) partitioning cost for n elements can be bounded by $a \cdot n + O(1)$, for a constant a, then the average cost for sorting n elements is

[2] For ease of discussion we assume in this theoretical study that all elements have different keys. Of course, in implementations equal keys are an important issue that requires a lot of care [7].

$$\frac{6}{5}a \cdot n \ln n + O(n). \tag{1}$$

Throughout the present paper all that interests us is the constant factor with the leading term. (The reader should be warned that for real-life n the linear term, which can even be negative, can have a big influence on the average number of comparisons.)

The second observation is that the partitioning cost depends on certain details of the partitioning procedure. This is in contrast to standard quicksort with one pivot where partitioning always takes $n-1$ comparisons. In [10] it is shown that Yaroslavskiy's partitioning procedure uses $\frac{19}{12}n + O(1)$ comparisons on average, while Sedgewick's uses $\frac{16}{9}n + O(1)$ many. For understanding what is going on it is helpful to forget about concrete implementations with loops in which pointers sweep across arrays and entries are swapped, and look at partitioning with two pivots in a more abstract way. For simplicity we shall always assume that the input is a permutation of $\{1, \ldots, n\}$. Now pivots p and q with $p < q$ are chosen. The task is to *classify* the remaining $n-2$ elements into classes "small" ($s = p-1$ many), "medium" ($m = q-p-1$ many), and "large" ($\ell = n-p$ many), by comparing these elements one after the other with the smaller pivot or the larger pivot, or both of them if necessary. Note that for symmetry reasons it is inessential in which order the elements are treated. The only choice the algorithm can make is whether to compare the current element with the smaller pivot or the larger pivot first. (In Sedgewick's and in Yaroslavskiy's algorithm this decision is based on the position of certain pointers and the state of control.) Let the random variable S_2 denote the number of small elements compared with the larger pivot first, and let L_2 denote the number of large elements compared with the smaller pivot first. Then the total number of comparisons is $n - 2 + m + S_2 + L_2$.

Averaging over all inputs and all possible choices of the pivots the term $n - 2 + m$ will lead to $\frac{4}{3}n + O(1)$ key comparisons on average, independently of the algorithm. Let $W = S_2 + L_2$, the number of elements that are compared with the "wrong" pivot first. Then $\mathrm{E}(W)$ is the only quantity that can be influenced by using a clever partitioning procedure.

In the paper, we will first devise an easy method to calculate $\mathrm{E}(W)$. The result of this analysis will lead to an (asymptotically) optimal strategy. The basic approach is the following. Assume a partitioning procedure is given, and assume p, q and hence $s = p-1$ and $\ell = n-q$ are fixed, and let $w_{s,\ell} = \mathrm{E}(W \mid s, \ell)$. Denote the average number of elements compared to the smaller [larger] pivot first by $f^{\mathrm{p}}_{s,\ell}$ [$f^{\mathrm{q}}_{s,\ell}$]. If the elements to be classified were chosen to be small, medium, and large independently with probabilities $s/(n-2)$, $m/(n-2)$, and $\ell/(n-2)$, resp., then the expected number of small elements compared with the large pivot first would be $f^{\mathrm{q}}_{s,\ell} \cdot s/(n-2)$, similarly for the large elements. Of course, the actual input is a sequence with exactly s [m, ℓ] small [medium, large] elements, and there is no independence. Still, we will show that the randomness in the order is sufficient to guarantee that

$$w_{s,\ell} = f^{\mathrm{q}}_{s,\ell} \cdot s/n + f^{\mathrm{p}}_{s,\ell} \cdot \ell/n + o(n). \tag{2}$$

The details of the partitioning procedure will determine $f^{\mathrm{p}}_{s,\ell}$ and $f^{\mathrm{q}}_{s,\ell}$, and hence $w_{s,\ell}$ up to $o(n)$. This seemingly simple insight has two consequences, one for the analysis and one for the design of dual pivot algorithms:

(i) In order to *analyze* the average comparison count of a dual pivot algorithm (given by its partitioning procedure) up to lower order terms determine $f^{\mathrm{p}}_{s,\ell}$ and $f^{\mathrm{q}}_{s,\ell}$ for this partitioning procedure. This will give $w_{s,\ell}$ up to lower order terms, which must then be averaged over all s, ℓ to find the average number of comparisons in partitioning. Then apply (1).

(ii) In order to *design* a good partitioning procedure w.r.t. the average comparison count, try to make $f^{\mathrm{q}}_{s,\ell} \cdot s/n + f^{\mathrm{p}}_{s,\ell} \cdot \ell/n$ small.

We shall demonstrate approach (i) in Section 4. An example: As explained in [10], if s and ℓ are fixed, in Yaroslavskiy's algorithm we have $f^{\mathrm{q}}_{s,\ell} \approx \ell$ and $f^{\mathrm{p}}_{s,\ell} \approx s+m$. By (2) we get $w_{s,\ell} = (\ell s + (s+m)\ell)/n + o(n)$. This must be averaged over all possible values of s and ℓ. The result is $\frac{1}{4}n + o(n)$, which together with $\frac{4}{3}n + O(1)$ gives $\frac{19}{12}n + o(n)$, close to the result from [10].

Principle (ii) will be used to identify an (asymptotically) optimal partitioning procedure that makes $1.8n \ln n + o(n \ln n)$ key comparisons on average. In brief, such a strategy should achieve the following: If $s > \ell$, compare (almost) all entries with the smaller pivot first ($f^{\mathrm{p}}_{s,\ell} \approx n$ and $f^{\mathrm{q}}_{s,\ell} \approx 0$), otherwise compare (almost) all entries with the larger pivot first ($f^{\mathrm{p}}_{s,\ell} \approx 0$ and $f^{\mathrm{q}}_{s,\ell} \approx n$). Of course, some details have to be worked out: How can the algorithm decide which case applies? In which technical sense is this strategy optimal? We shall see in Section 5 how a sampling technique resolves these issues.

At the end of this paper, in Section 6, we will consider the following simple and intuitive strategy: *Compare the current element to the smaller pivot first if more small elements than large elements have been seen so far, otherwise compare it to the larger pivot first.* It can be shown [1] that this is optimal w.r.t. the average comparison count with an error term of only $O(n)$ instead of $o(n \ln n)$.

As noted by Wild *et al.* [11], considering only key comparisons and swap operations does not suffice for evaluating the practicability of sorting algorithms. In Section 7, we will present preliminary experimental results that indicate the following: When sorting integers, the "optimal" method of Section 5 is slower than Yaroslavskiy's algorithm. When making key comparisons artificially expensive, e.g., by sorting strings, we gain a small advantage.

We emphasize that the purpose of this paper is not to arrive at better and better quicksort algorithms by using all kinds of variations, but rather to thoroughly analyze the situation with two pivots, showing the potential and the limitations of this approach. For an analysis of other variations of classical quicksort, notably those that use random sampling to find a good pivot, see, e.g., [8]. Furthermore, we concentrate only on the average comparison count. The average swap count can be analyzed independently of this. We mention here that there exists a swapping strategy which beats the strategy employed by Yaroslavskiy's algorithm theoretically. Preliminary results for this analysis can be found in [1].

2 Basic Approach to Analyzing Dual Pivot Quicksort

We assume the input sequence $(a_1, \ldots, a_n)$ to be a random permutation of $\{1, \ldots, n\}$, each permutation occurring with probability $(1/n!)$. If $n \leq 1$, there is nothing to do; if $n = 2$, sort by one comparison. Otherwise, choose the first element a_1 and the last element a_n as pivots, and set $p = \min(a_1, a_n)$ and $q = \max(a_1, a_n)$. Partition the remaining elements into elements $< p$ ("small" elements), elements in between p and q ("medium" elements), and elements $> q$ ("large" elements), see Fig. 1. Then apply the procedure recursively to these three groups. Clearly, each pair p, q with $1 \leq p < q \leq n$ appears as pivots with probability $1/\binom{n}{2}$. Our cost measure is the number of key comparisons needed to sort the given input. Let C_n be the random variable counting this number. Let P_n denote the partitioning cost to partition the $n-2$ non-pivot elements into the three groups. As explained by Wild and Nebel [10, Appendix A], the average number of key comparisons obeys the following recurrence:

$$\mathrm{E}(C_n) = \mathrm{E}(P_n) + \frac{1}{\binom{n}{2}} \cdot 3 \sum_{k=0}^{n-2} (n-k-1) \cdot \mathrm{E}(C_k).$$

If $\mathrm{E}(P_n) = a \cdot n + O(1)$, for a constant a, this can be solved (*cf.* [3,10]) to give

$$\mathrm{E}(C_n) = \frac{6}{5} a \cdot n \ln n + O(n). \tag{3}$$

Abstracting from moving elements around in arrays, we arrive at the following "classification problem": Given a random permutation $(a_1, \ldots, a_n)$ of $\{1, \ldots, n\}$ as the input sequence and a_1 and a_n as the two pivots p and q, with $p < q$, classify each of the remaining $n-2$ elements as being small, medium, or large. Note that there are exactly $s := p - 1$ small elements, $m := q - p - 1$ medium elements, and $\ell := n - q$ large elements. Although this classification does not yield an actual partition of the input sequence, a classification algorithm can be turned into a partitioning algorithm using only swap operations but no additional key comparisons. Since elements are only compared with the two pivots, the randomness of subarrays is preserved. Thus, in the recursion we may always assume that the input is arranged randomly.

We make the following observations (and fix notation) for all classification algorithms. One key comparison is needed to decide which of the elements a_1 and a_n is the smaller pivot p and which is the larger pivot q. For classification, each of the remaining $n-2$ elements has to be compared against p or q or both. Each *medium* element has to be compared to p *and* q. We expect $(n-2)/3$ medium elements. Let S_2 denote the number of small elements that are compared to the larger pivot first, i.e., the number of small elements that need 2 comparisons for classification. Analogously, let L_2 denote the number of large elements compared to the smaller pivot first. Conditioning on the pivot choices, and hence the values of s and ℓ, we may calculate $\mathrm{E}(P_n)$ as follows:

$$\mathrm{E}(P_n) = (n-1) + (n-2)/3 + \frac{1}{\binom{n}{2}} \sum_{s+\ell \leq n-2} \mathrm{E}(S_2 + L_2 \mid s, \ell). \tag{4}$$

We call the third summand the *additional cost term (ACT)*, as it is the only value that depends on the actual classification algorithm.

3 Analyzing the ACT

We will use the following formalization of a partitioning procedure: A classification algorithm (or *strategy*) is a three-way decision tree T with a root and $n-2$ levels of inner nodes as well as one leaf level. The root is on level 0. Each node v is labeled with an index $i \in \{2, \ldots, n-1\}$ and an element $l(v) \in \{\mathrm{p}, \mathrm{q}\}$. If $l(v)$ is p, then at node v element a_i is compared with the smaller pivot first; otherwise, i.e., $l(v) = \mathrm{q}$, it is compared with the larger pivot first. The three edges out of a node are labeled σ, μ, λ, resp., representing the outcome of the classification as small, medium, large, respectively. The label of edge e is called $c(e)$. On each of the 3^{n-2} paths each index i occurs exactly once. Each input determines exactly one path w from the root to a leaf in the obvious way; the classification of the elements can then be read off from the node and edge labels along this path.

We now describe how we can calculate the ACT of a decision tree T. Fix s and ℓ, and let the input excepting the pivots be arranged randomly. Let $p^v_{s,\ell}$ be the probability that node v is reached. The probability that at some node v the algorithm classifies the element treated at v as small, medium, or large, resp., depends only on the number of small, medium, and large elements classified so far, but not on which of the $n-2-\mathrm{level}(v)$ still unclassified elements is chosen. This follows from the assumption that the input sequence is in random order.

For a node v in T, we let $Y_v = 0$ if $l(v) = \mathrm{p}$, and $Y_v = 1$ if $l(v) = \mathrm{q}$. We let s_v, m_v, and ℓ_v, resp., denote the number of edges labeled σ, μ, and λ, resp., from the root to v. The probability that the element classified at v is "small" is exactly $(s - s_v)/(n - 2 - \mathrm{level}(v))$. The probability that it is "medium" is $(m - m_v)/(n - 2 - \mathrm{level}(v))$, and that it is "large" is $(\ell - \ell_v)/(n - 2 - \mathrm{level}(v))$. The probability that the edge labeled σ out of a node v is used by the algorithm is then $p^v_{s,\ell} \cdot (s - s_v)/(n - 2 - \mathrm{level}(v))$. Similarly, the probability that the edge labeled λ is used is $p^v_{s,\ell} \cdot (\ell - \ell_v)/(n - 2 - \mathrm{level}(v))$. For a random input, we let S_2^T $[L_2^T]$ denote the random variable that counts the number of small [large] elements classified in nodes with label q [p]. Then

$$\mathrm{E}(S_2^T + L_2^T \mid s, \ell) = \sum_{v \in T} p^v_{s,\ell} \left(Y_v \cdot \frac{s - s_v}{n-2-\mathrm{level}(v)} + (1 - Y_v) \cdot \frac{\ell - \ell_v}{n-2-\mathrm{level}(v)} \right). \quad (5)$$

The setup developed so far makes it possible to describe the connection between a decision tree T and its average comparison count in general. Let F_{p}^T resp. F_{q}^T be two random variables that denote the number of elements that are compared with the smaller resp. larger pivot first when using T. Then let $f^{\mathrm{q}}_{s,\ell} = \mathrm{E}\left(F_{\mathrm{q}}^T \mid s, \ell\right)$ resp. $f^{\mathrm{p}}_{s,\ell} = \mathrm{E}\left(F_{\mathrm{p}}^T \mid s, \ell\right)$ denote the average number of comparisons with the larger resp. smaller pivot first, given s and ℓ. Now, if it was decided in each step by independent random experiments with the correct expectations $s/(n-2)$, $m/(n-2)$, and $\ell/(n-2)$, resp., whether an element is small, medium, or large,

it would be clear that for example $f^{\mathrm{q}}_{s,\ell} \cdot s/(n-2)$ is the average number of small elements that are compared with the larger pivot first. The next lemma shows that one can indeed use this intuition in the calculation of the average comparison count, excepting that one gets an additional $o(n)$ term due to the elements tested not being independent. The proof of this lemma uses (5) and martingale tail estimates to show that most of the time the probability of seeing small resp. large elements is close to $s/(n-2)$ resp. $\ell/(n-2)$.

Lemma 1. *Let T be a decision tree. Let $\mathrm{E}(P_n^T)$ be the average number of key comparisons for classifying an input of n elements using T. Then*

$$\mathrm{E}(P_n^T) = \frac{4}{3}n + \frac{1}{\binom{n}{2}\cdot(n-2)} \sum_{s+\ell\le n-2} \left(f^{\mathrm{q}}_{s,\ell}\cdot s + f^{\mathrm{p}}_{s,\ell}\cdot \ell \right) + o(n).$$

There are two technical complications when using this lemma in analyzing a strategy that is turned into a dual pivot quicksort algorithm. The cost bound is $a \cdot n + o(n)$. Equation (3) cannot be applied directly to such partitioning costs. Furthermore, the $o(n)$ term in Lemma 1 will get out of control if the subarrays appearing in the recursion become too small. However, the next theorem says that the leading term of (3) applies to this situation as well, but we get an error term of $o(n \ln n)$ instead of $O(n)$.

Theorem 1. *Let $\mathcal{A}$ be a dual pivot quicksort algorithm that gives rise to a decision tree T_n for each subarray of length n. Assume $\mathrm{E}(P_n^{T_n}) = a \cdot n + o(n)$ for all n, for some constant a. Then $\mathrm{E}\left(C_n^{\mathcal{A}}\right) = \frac{6}{5}an\ln n + o(n\ln n)$.*

Roughly, the proof proceeds as follows. Fix an arbitrarily small number $\varepsilon > 0$. Then there is some n_ε such that for all $n' \ge n_\varepsilon$ the average partitioning cost on a subarray of size n' is smaller than $(a+\varepsilon)n'$. We only consider n so large that $n^{1/\ln\ln n} \ge n_\varepsilon$ and that $(\ln n)/\ln\ln n < \varepsilon \ln n$. We split the analysis of the expected cost into two parts. For subarrays of length $n' \ge n_0 = n^{1/\ln\ln n}$ the average partitioning cost is at most $(a+\varepsilon)n'$; for a subarray of size $n' < n_0$ we charge $(a+\varepsilon)n'$ to the first part of the analysis. Then (3) can be used to estimate the contribution of the first part as $\frac{6}{5}(a+\varepsilon)n\ln n + O(n) = \frac{6}{5}an\ln n + \frac{6}{5}\varepsilon n \ln n + O(n)$. In the second part we collect the contributions from splitting subarrays of size n' not captured by the first part. Each such contribution is bounded by $2n'$ (even in absolute value). The second part of the analysis consists in adding $2n'$ for each subarray of size $n' < n_0$ and 0 for each larger subarray. For this, we wait until the algorithm has created a subarray of size $n' < n_0$ and assess the total contribution from the recursion starting from this subarray as not more than $\frac{12}{5}n'\ln n' + O(n')$, by (3). This means we must sum $\frac{12}{5}n_i \ln n_i + O(n_i)$, $1 \le i \le k$, over a sequence of disjoint subarrays of length $n_1, \ldots, n_k$. Since all n_i are smaller than n_0, $n_1 + \ldots + n_k \le n$, and since $x \mapsto x\ln x$ is a convex function, this sums up to no more than $\frac{n}{n_0}\cdot\frac{6}{5}\cdot 2n_0\ln n_0 + O(n) = \frac{12}{5}n\ln(n_0) + O(n) < \varepsilon n\ln n + O(n)$ for n large enough. Adding both parts, and choosing n so large that the two $O(n)$ terms can be bounded by $\varepsilon n\ln n$ we get the bound $\frac{6}{5}an\ln n + \frac{16}{5}\varepsilon n\ln n$, which is sufficient since ε was arbitrary.

Lemma 1 and Theorem 1 tell us that for the analysis of the average comparison count of a dual pivot quicksort algorithm we just have to find out what $f^{\mathrm{p}}_{s,\ell}$ and $f^{\mathrm{q}}_{s,\ell}$ are for this algorithm. Moreover, to design a good algorithm (w.r.t. the average comparison count), we should try to make $f^{\mathrm{q}}_{s,\ell} \cdot s + f^{\mathrm{p}}_{s,\ell} \cdot \ell$ small.

4 Analysis of Some Known Partitioning Methods

In this section, we will study three different partitioning methods in the light of the formulas from Section 3.

Smaller Pivot First. We consider strategy $\mathcal{P}$: *Always compare with the smaller pivot p first.* This strategy makes sure that $f^{\mathrm{p}}_{s,\ell} = n - 2$ and $f^{\mathrm{q}}_{s,\ell} = 0$. Applying Lemma 1 gives us $\mathrm{E}(P_n) = \frac{5}{3}n + o(n)$. Using Thm. 1 we get $\mathrm{E}(C_n) = 2n \ln n + o(n \ln n)$—the leading term being the same as in standard quicksort.[3]

Yaroslavskiy's Algorithm. Following [10, Section 3.2], Yaroslavskiy's algorithm is an implementation of the following strategy $\mathcal{Y}$: *Compare ℓ elements to q first, and compare the other elements to p first.* We get that $f^{\mathrm{q}}_{s,\ell} = \ell$ and $f^{\mathrm{p}}_{s,\ell} = s + m$. Applying Lemma 1, we calculate $\mathrm{E}(P_n) = \frac{19}{12}n + o(n)$. Using Thm. 1 gives $\mathrm{E}(C_n) = 1.9n \ln n + o(n \ln n)$, as in [10].

Sedgewick's Algorithm. Following [10, Section 3.2], Sedgewick's algorithm amounts to an implementation of the following strategy $\mathcal{S}$: *Compare (on average) a fraction of $s/(s+\ell)$ of the keys with q first, and compare the other keys with p first.* We get $f^{\mathrm{q}}_{s,\ell} = (n-2) \cdot s/(s+\ell)$ and $f^{\mathrm{p}}_{s,\ell} = (n-2) \cdot \ell/(s+\ell)$. Using Lemma 1, we calculate $\mathrm{E}(P_n) = \frac{16}{9}n + o(n)$. Applying Thm. 1 gives $\mathrm{E}(C_n) = 2.133\ldots \cdot n \ln n + o(n \ln n)$, as known from [10].

5 An (Asymptotically) Optimal Partitioning Method

Looking at the previous sections, all methods used the idea that we should compare a certain fraction of elements to p first, and all other elements to q first. In this section, we will study the following strategy $\mathcal{D}$: *If $s > \ell$ then always compare with p first, otherwise always compare with q first.*

Of course, for an implementation of this strategy we have to deal with the problem of finding out which case applies before all comparisons have been made.

5.1 Analysis of the Ideal Classification Strategy

Assume for a moment that for a given random input with pivots p, q the strategy "magically" knows whether $s > \ell$ or not and correctly determines the pivot that should be used for all comparisons. For the average comparison count one obtains by a standard calculation:

[3] The same result is obtained if one picks the pivot to compare with by a coin flip.

$$\mathrm{E}(P_n) = \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+\ell \le n-2} \min(s,\ell) + o(n) = \frac{3}{2}n + o(n). \tag{6}$$

Applying Thm. 1, we get $\mathrm{E}(C_n) = 1.8n \ln n + o(n \ln n)$, which is by $0.1n \ln n$ smaller than the average number of key comparisons in Yaroslavskiy's algorithm.

To see that this method is (asymptotically) optimal, recall that according to Lemma 1 the average comparison count is determined up to lower order terms by the parameters $f^{\mathrm{q}}_{s,\ell}$ and $f^{\mathrm{p}}_{s,\ell} = n - 2 - f^{\mathrm{q}}_{s,\ell}$. Strategy $\mathcal{D}$ chooses these values such that each term of the sum in Lemma 1 is minimized—thus minimizing the sum.

5.2 Guessing Whether $s < \ell$ or Not

We explain how the ideal classification algorithm can be approximated by an implementation. The idea simply is to make a few comparisons and use the outcome as a basis for a guess.

After p and q are chosen, classify $sz = o(n)$ many elements and calculate s' and ℓ', the number of small and large elements in the sample. If $s' < \ell'$, compare with q first, otherwise compare with p first. We say that the guess was *correct*, if the relation "$s' < \ell'$" correctly reflects whether $s < \ell$ or not.

We incorporate guessing errors into (6) as follows:

$$\begin{aligned}\mathrm{E}(P_n) &= \frac{4}{3}n + o(n) + \frac{1}{\binom{n}{2}} \sum_{s+\ell \le n-2} \Big(\Pr(\text{guess correct}) \cdot \min(s,\ell) + \\ &\qquad \Pr(\text{guess wrong}) \cdot \max(s,\ell) \Big) \\ &= \frac{4}{3}n + o(n) + \frac{2}{\binom{n}{2}} \sum_{s=0}^{n/2} \sum_{\ell=s+1}^{n-s} \Big(\Pr(\text{guess correct}) \cdot s + \\ &\qquad \Pr(\text{guess wrong}) \cdot \ell \Big). \end{aligned} \tag{7}$$

The following lemma says that for a wide range of values s and ℓ, the probability of a guessing error is exponentially small.

Lemma 2. *Let s and ℓ with $s \le \ell - n^{3/4}$ and $\ell \ge n^{3/4}$ for $n \in \mathbb{N}$ be given. Let $sz = n^{2/3}$. Then $\Pr(s' > \ell') \le \exp\left(-2n^{1/6}/9\right)$.*

Our classification algorithm will now work as follows. It starts by sampling $sz = n^{2/3}$ many elements and then decides which pivot to use for the first comparison. Then it inspects the remaining elements according to this approach.

Theorem 2. *Let $sz = n^{2/3}$. Then the average comparison count of the algorithm described above is $1.8n \ln n + o(n \ln n)$.*

The proof is based on (7) and uses that in the two cases that (i) there is only a small number of small resp. large elements, and (ii) the difference between the number of small and large elements is at most $n^{2/3}$, the contribution of

these terms to (7) is $o(n)$. Otherwise, the contribution of wrong guesses is small according to Lemma 2.

While being optimal, this strategy has an error term of $o(n \ln n)$. In the next section, we will present a different strategy that will be optimal up to $O(n)$.

6 An Optimal Partitioning Strategy with Small Error

We will consider two more strategies. One optimal (but not algorithmic) strategy, and one algorithmic strategy that is optimal up to a very small error term.

We first study the (unrealistic!) setting where s and ℓ, i.e., the number of small resp. large elements, are known to the algorithm after the pivots are chosen, and the decision tree can have different node labels for each such pair of values. Recall that s_v and ℓ_v, resp., denote the number of elements that have been classified as small and large, resp., when at node v in the decision tree. We consider the following strategy $\mathcal{O}$: *Given s and ℓ, the comparison at node v is with the smaller pivot first if $s - s_v > \ell - \ell_v$, otherwise, it is with the larger pivot first.*[4]

Theorem 3. *Strategy $\mathcal{O}$ is optimal, i.e., its ACT is at most as large as ACT_T for every single tree T. When using $\mathcal{O}$ in a dual pivot quicksort algorithm, we get* $\mathrm{E}(C_n^{\mathcal{O}}) = 1.8n \ln n + O(n)$.

The proof of the first statement uses that for fixed pivots this strategy makes the choice that minimizes each term in the sum of (5), and linearity of expectation. Calculating the average comparison count is harder; a proof can be found in the full paper [1]. It uses that one can bound the number of "wrong" comparisons by the number of small [large] elements in the input, if $s < \ell$ [$s > \ell$]. However, this is only an upper bound. For an exact calculation one must take into account the average (over all inputs) number of positions i where we have the same number of small and large elements in $\{a_2, \ldots, a_{i-1}\}$ or $\{a_2, \ldots, a_i\}$, depending on whether i is even or odd, since in these cases the algorithm errs only in half of the cases. A somewhat involved calculation shows that this number is $O(\log n)$. Additionally, one can also show that a (negative) term of $O(\log n)$ decreases the total average number of key comparisons only by $O(n)$ when we use such a strategy to implement a dual pivot quicksort algorithm. For details see [1].

While being optimal w.r.t. minimizing the ACT, the assumption that the exact number of small and large elements is known is of course not true for a real algorithm or for a fixed tree. We can, however, identify a real, algorithmic partitioning strategy whose ACT differs from the optimal one only by a logarithmic term. We study the following strategy $\mathcal{L}$: *The comparison at node v is with the smaller pivot first if $s_v > l_v$, otherwise it is with the larger pivot first.*

While $\mathcal{O}$ looks into the future ("Are there more small elements or more large elements left?"), strategy $\mathcal{L}$ looks into the past ("Have I seen more small or more large elements so far?"). It is not hard to see that for a given input the number of additional comparisons of strategy $\mathcal{O}$ and $\mathcal{L}$ can differ significantly. The next

[4] This strategy was suggested to us by Thomas Hotz (personal communication).

theorem shows that averaged over all possible inputs, however, there is only a small difference.

Theorem 4. *Let $ACT_{\mathcal{O}}$ resp. $ACT_{\mathcal{L}}$ be the ACT for classifying n elements using strategy $\mathcal{O}$ resp. $\mathcal{L}$. Then $ACT_{\mathcal{L}} = ACT_{\mathcal{O}} + O(\log n)$. When using $\mathcal{L}$ in a dual pivot quicksort algorithm, we get* $\mathrm{E}(C_n^{\mathcal{L}}) = 1.8n \ln n + O(n)$.

The proof of this theorem uses the following key insight: Assume that strategy $\mathcal{O}$ inspects the elements in the order $a_{n-1}, \ldots, a_2$, while $\mathcal{L}$ uses the order $a_2, \ldots, a_{n-1}$. If the strategies compare the element a_i to different pivots, then there are exactly as many small elements as there are large elements in $\{a_2, \ldots, a_{i-1}\}$ or $\{a_2, \ldots, a_i\}$, depending on whether i is even or odd. A somewhat involved calculation shows that $\mathrm{ACT}_{\mathcal{L}} - \mathrm{ACT}_{\mathcal{O}}$ is $O(\log n)$, which again sums up to a total additive contribution of $O(n)$ when used in a dual pivot quicksort algorithm. Thus, dual pivot quicksort with strategy $\mathcal{L}$ has average cost at most $O(n)$ larger than dual pivot quicksort using the (unrealistic) strategy $\mathcal{O}$.

7 Experiments

We have implemented the methods presented in this paper in C++. These algorithms have not been fine-tuned (in particular, in our method swapping was carried out in a naïve way); this section is hence meant to provide only preliminary results and does not replace a thorough experimental study. Our experiments were carried out on an Intel Xeon E5645 at 2.4 GHz with 48 GB Ram running Ubuntu 12.04 with kernel version 3.2.0. The source code was compiled with *gcc* using the *-O2* optimization flag. We have incorporated random sampling into the partitioning step of our method (strategy $\mathcal{D}$) by comparing the first $n' = \max(n/100, 7)$ elements with p first. We switched to comparing with q first if the algorithm has seen more large than small elements after n' steps.

With respect to running times, we see that Yaroslavskiy's algorithm is superior to the other algorithms when sorting random permutations of $\{1, \ldots, n\}$.

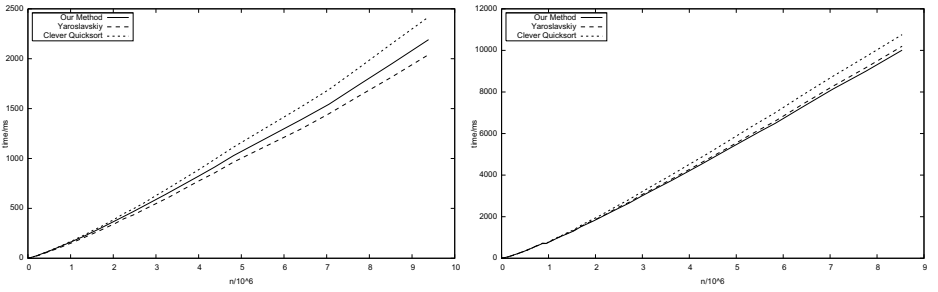


Fig. 2. *Left*: Running time (in milliseconds) needed to sort a random permutation of $\{1, \ldots, n\}$. *Right*: Running time needed to sort a random set of n article headers of the English Wikipedia. Running times were averaged over 1000 trials for each n. Clever quicksort uses the median in a sample of three elements as the pivot. Strategy $\mathcal{L}$ and strategy $\mathcal{D}$ behaved almost identically in the experiments.

Strategy $\mathcal{D}$ is about 5% slower. When sorting strings (making key comparisons artificially more expensive), strategy $\mathcal{D}$ can actually beat Yaroslavskiy's algorithm, see Figure 2. However, the difference is only about 2%. Note that Java 7 does not use Yaroslavskiy's algorithm when sorting strings.

As found in [11], it is a major enterprise to make an asymptotically good algorithm also perform well in practice.

8 Conclusion and Open Questions

We have studied dual pivot quicksort algorithms in a unified way and found optimal partitioning methods that minimize the average number of key comparisons up to lower order terms and even up to $O(n)$. From an engineering point of view, one might look for a clever implementation of our classification idea that beats Yaroslavskiy's algorithm. To this end, studying different swap strategies might be beneficial. In standard quicksort it helps to choose the pivot from a sample [8]. Similary, as studied by Hennequin [3], one could choose two pivots from a sample. It is a question how all these strategies compare. Preliminary results [1] indicate that starting from a sample of size 7 classical quicksort beats dual pivot quicksort using a sample of size 8 w.r.t. the average comparison count.

Acknowledgements. The authors thank Thomas Hotz for a useful suggestion regarding the optimal strategy of Section 6, and Pascal Klaue for implementing the algorithms and carrying out initial experiments. The first author thanks Michael Rink for interesting and useful discussions. We thank the referees of the conference submission for their insightful comments, which helped a lot in improving the presentation.

References

1. Aumüller, M., Dietzfelbinger, M.: Optimal partitioning for dual pivot quicksort. CoRR abs/1303.5217 (2013)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
3. Hennequin, P.: Analyse en moyenne d'algorithmes: tri rapide et arbres de recherche. Ph.D. thesis, Ecole Politechnique, Palaiseau (1991)
4. Hoare, C.A.R.: Quicksort. Comput. J. 5(1), 10–15 (1962)
5. Knuth, D.E.: The Art of Computer Programming. Sorting and Searching, vol. III. Addison-Wesley (1973)
6. Sedgewick, R.: Quicksort. Ph.D. thesis, Standford University (1975)
7. Sedgewick, R.: Quicksort with equal keys. SIAM J. Comput. 6(2), 240–268 (1977)
8. Sedgewick, R.: Implementing quicksort programs. Commun. ACM 21(10), 847–857 (1978)
9. Sedgewick, R., Flajolet, P.: An introduction to the analysis of algorithms. Addison-Wesley-Longman (1996)
10. Wild, S., Nebel, M.E.: Average case analysis of Java 7's dual pivot quicksort. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 825–836. Springer, Heidelberg (2012)
11. Wild, S., Nebel, M.E., Reitzig, R., Laube, U.: Engineering Java 7's dual pivot quicksort using MaLiJan. In: ALENEX 2013, pp. 55–69 (2013)

Space–Time Tradeoffs for Subset Sum: An Improved Worst Case Algorithm*

Per Austrin[1,2], Petteri Kaski[3], Mikko Koivisto[4], and Jussi Määttä[3]

[1] Aalto Science Institute, Aalto University, Finland
[2] KTH Royal Institute of Technology, Sweden
[3] HIIT & Department of Information and Computer Science, Aalto University, Finland
[4] HIIT & Department of Computer Science, University of Helsinki, Finland

Abstract. The technique of Schroeppel and Shamir (SICOMP, 1981) has long been the most efficient way to trade space against time for the SUBSET SUM problem. In the random-instance setting, however, improved tradeoffs exist. In particular, the recently discovered dissection method of Dinur et al. (CRYPTO 2012) yields a significantly improved space–time tradeoff curve for instances with strong randomness properties. Our main result is that these strong randomness assumptions can be removed, obtaining the same space–time tradeoffs in the worst case. We also show that for small space usage the dissection algorithm can be almost fully parallelized. Our strategy for dealing with arbitrary instances is to instead inject the randomness into the dissection process itself by working over a carefully selected but random composite modulus, and to introduce explicit space–time controls into the algorithm by means of a "bailout mechanism".

1 Introduction

The protagonist of this paper is the SUBSET SUM problem.

Definition 1. *An instance* $(\boldsymbol{a}, t)$ *of* SUBSET SUM *consists of a vector* $\boldsymbol{a} \in \mathbb{Z}_{\geq 0}^n$ *and a target* $t \in \mathbb{Z}_{\geq 0}$. *A solution of* $(\boldsymbol{a}, t)$ *is a vector* $\boldsymbol{x} \in \{0,1\}^n$ *such that* $\sum_{i=1}^n a_i x_i = t$.

The problem is NP-hard (in essence, Karp's formulation of the knapsack problem [6]), and the fastest known algorithms take time and space that grow exponentially in n. We will write T and S for the exponential factors and omit the possible polynomial factors. The brute-force algorithm, with $T = 2^n$ and $S = 1$, was beaten four decades ago, when Horowitz and Sahni [4] gave a simple yet powerful meet-in-the-middle algorithm that achieves $T = S = 2^{n/2}$ by halving the set arbitrarily, sorting the $2^{n/2}$ subsets of each half, and then quickly scanning through the relevant pairs of subsets that could sum to the target. Some years later, Schroeppel and Shamir [10] improved the space

* P.A. supported by the Aalto Science Institute, the Swedish Research Council grant 621-2012-4546, and ERC Advanced Investigator grant 226203. P.K. supported by the Academy of Finland, grants 252083 and 256287. M.K. supported by the Academy of Finland, grants 125637, 218153, and 255675. A full version of the present conference abstract is available at: http://arxiv.org/abs/1303.0609

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 45–56, 2013.

requirement of the algorithm to $S = 2^{n/4}$ by designing a novel way to list the half-sums in sorted order in small space. However, if allowing only polynomial space, no better than the trivial time bound of $T = 2^n$ is known. Whether the constant bases of the exponentials in these bounds can be improved is a major open problem in the area of moderately exponential algorithms [12].

The difficulty of finding faster algorithms, whether in polynomial or exponential space, motivates the study of space–time tradeoffs. From a practical point of view, large space usage is often the bottleneck of computation, and savings in space usage can have significant impact even if they come at the cost of increasing the time requirement. This is because a smaller-space algorithm can make a better use of fast cache memories and, in particular, because a smaller-space algorithm often enables easier and more efficient large-scale parallelization. Typically, one obtains a smooth space–time tradeoff by combining the fastest exponential time algorithm with the fastest polynomial space algorithm into a hybrid scheme that interpolates between the two extremes. An intriguing question is then whether one can beat the hybrid scheme at some point, that is, to get a faster algorithm at some space budget—if one can break the hybrid bound somewhere, maybe one can break it everywhere. For SUBSET SUM a hybrid scheme is obtained by first guessing some g elements of the solution, and then running the algorithm of Schroeppel and Shamir for the remaining instance on $n - g$ elements. This yields $T = 2^{(n+g)/2}$ and $S = 2^{(n-g)/4}$, for any $0 \leq g \leq n$, and thereby the smooth tradeoff curve $S^2T = 2^n$ for $1 \leq S \leq 2^{n/4}$. We call this the *Schroeppel–Shamir tradeoff*.

While the Schroeppel–Shamir tradeoff has remained unbeaten in the usual worst-case sense, there has been remarkable recent progress in the random-instance setting. In 2010 Howgrave-Graham and Joux [5] introduced new techniques based on modular arithmetic to get a faster exponential-space algorithm. In 2011 Becker, Coron, and Joux [1] extended the techniques and presented, not only yet a faster algorithm, but also a tradeoff curve satisfying $ST = 2^{3n/4}$ in the range $2^{n/16} \leq S \leq 2^{n/4}$, so beating the Schroeppel–Shamir tradeoff in this range. Very recently, Dinur, Dunkelman, Keller, and Shamir [3] introduced a recursive approach and obtained a tradeoff curve that matches the Schroeppel–Shamir tradeoff at the extreme points $S = 1$ and $S = 2^{n/4}$ but is strictly better in between. The tradeoff is achieved by a novel *dissection* method that recursively decomposes the problem into smaller subproblems in two different "dimensions", the first dimension being the current subset of the n items, and the other dimension being (roughly speaking) the bits of information of each item. The algorithm of Dinur et al. runs in space $S = 2^{\sigma n}$ and time $T = 2^{\tau(\sigma)n}$ on random instances ($\tau(\sigma)$ is defined momentarily). See Figure 1 for an illustration and comparison to the Schroeppel–Shamir tradeoff. The tradeoff curve $\tau(\sigma)$ is piecewise linear and determined by what Dinur et al. call the "magic sequence" $2, 4, 7, 11, 16, 22, \ldots$, obtained as evaluations of $\rho_\ell = 1 + \ell(\ell+1)/2$ at $\ell = 1, 2, \ldots$.

Definition 2. *Define $\tau : (0,1] \to [0,1]$ as follows. For $\sigma \in (0, 1/2]$, let ℓ be the solution to $1/\rho_{\ell+1} < \sigma \leq 1/\rho_\ell$. Then*

$$\tau(\sigma) = 1 - \frac{1}{\ell+1} - \frac{\rho_\ell - 2}{\ell+1}\sigma\,. \tag{1}$$

If there is no such ℓ, that is, if $\sigma > 1/2$, define $\tau(\sigma) = 1/2$.

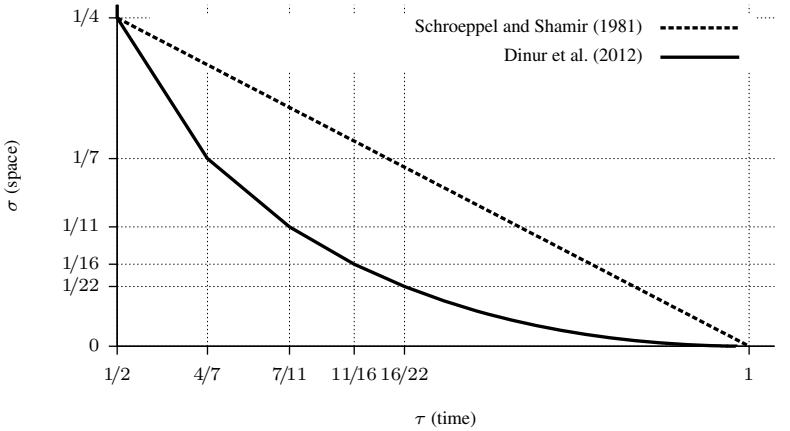


Fig. 1. Space–time tradeoff curves for the SUBSET SUM problem [10,3]. The space and time requirements are $S = 2^{\sigma n}$ and $T = 2^{\tau n}$, omitting factors polynomial in the instance size n.

For example, at $\sigma = 1/8$, we have $\ell = 3$, and thereby $\tau(\sigma) = 19/32$. Asymptotically, when σ is small, ℓ is essentially $\sqrt{2/\sigma}$ and $\tau(\sigma) \approx 1 - \sqrt{2\sigma}$.

In this paper, we show that this space–time tradeoff result by Dinur et al. [3] can be made to hold also in the worst case:

Theorem 1. *For each $\sigma \in (0, 1]$ there exists a randomized algorithm that solves the* SUBSET SUM *problem with high probability, and runs in $O^*(2^{\tau(\sigma)n})$ time and $O^*(2^{\sigma n})$ space. The O^* notation suppresses factors that are polynomial in n, and the polynomials depend on σ.*

To the best of our knowledge, Theorem 1 is the first improvement to the Schroeppel–Shamir tradeoff in the worst-case setting. Independently of our work, however, Wang [11] has very recently presented a different randomized hashing approach and established a tradeoff curve that beats the Schroeppel–Shamir tradeoff but does not quite achieve the bounds in Theorem 1. Here we should also remark that, in the random-instance setting, there are results that improve on both the Schroeppel–Shamir and the Dinur et al. tradeoffs for certain specific choices of the space budget S. In particular, Becker et al. give a $2^{0.72n}$ time polynomial space algorithm and a $2^{0.291n}$ time exponential space algorithm [1]. A natural question that remains is whether these two results could be extended to the worst-case setting. Such an extension would be a significant breakthrough (cf. [12]).

We also prove that the dissection algorithm lends itself to parallelization very well. As mentioned before, a general guiding intuition is that algorithms that use less space can be more efficiently parallelized. The following theorem shows that, at least in the case of the dissection algorithm, this intuition can be made formal: the smaller the space budget σ is, the closer we can get to full parallelization.

Remark. For reasons of space, this conference abstract does not contain the detailed proofs of all the claims; these are indicated with a "†" symbol and can be found in the full version of this paper (see footnote on title page).

Theorem 2 (†). *The algorithm of Theorem 1 can be implemented to run in* $O^*(2^{\tau(\sigma)n}/P)$ *parallel time on* P *processors each using* $O^*(2^{\sigma n})$ *space, provided* $P \leq 2^{(2\tau(\sigma)-1)n}$.

When σ is small, $\tau(\sigma) \approx 1 - \sqrt{2\sigma}$ and the bound on P is roughly $2^{(\tau(\sigma)-\sqrt{2\sigma})n}$. In other words we get a linear speedup almost all the way up to $2^{\tau(\sigma)n}$ processors, almost full parallelization.

Our Contributions and Overview of the Proof. At a high level, our approach will follow the Dinur et al. dissection framework, with essential differences in preprocessing and low-level implementation to alleviate the assumptions on randomness. In particular, while we split the instance analogously to Dinur et al. to recover the tradeoff curve, we require more careful control of the sub-instances beyond just subdividing the bits of the input integers and assuming that the input is random enough to guarantee sufficient uniformity to yield the tradeoff curve. Accordingly we find it convenient to revisit the derivation of the tradeoff curve and the analysis of the basic dissection framework to enable a self-contained exposition.

In contrast with Dinur et al., our strategy for dealing with arbitrary instances is, essentially, to instead inject the required randomness into the dissection process itself. We achieve this by observing that dissection can be carried out over any algebraic structure that has a sufficiently rich family of homomorphisms to enable us to inject entropy by selection of *random* homomorphisms, while maintaining an appropriate recursive structure for the selected homomorphisms to facilitate dissection. For the SUBSET SUM problem, in practice this means reduction from $\mathbb{Z}$ to $\mathbb{Z}_M$ over a composite M with a carefully selected (but random) lattice of divisors to make sure that we can still carry out recursive dissections analogously to Dinur et al. This approach alone does not provide sufficient control over an arbitrary problem instance, however.

The main obstacle is that, even with the randomness injected into the algorithm, it is very hard to control the resource consumption of the algorithm. To overcome this, we add explicit resource controls into the algorithm, by means of a somewhat cavalier "bailout mechanism" which causes the algorithm to simply stop when too many partial solutions have been generated. We set the threshold for such a bailout to be roughly the number of partial solutions that we would have expected to see in a random instance. This allows us to keep the running time and space usage of the algorithm in check, perfectly recovering the Dinur et al. tradeoff curve. The remaining challenge is then to prove correctness, i.e., that these thresholds for bailout are high enough so that no hazardous bailouts take place and a solution is indeed found. To do this we perform a localized analysis on the subtree of the recursion tree that contains a solution. Using that the constructed modulus M contains a lot of randomness (a consequence of the density of the primes), we can show that the probability of a bailout in any node of this subtree is $o(1)$, meaning that the algorithm finds a solution with high probability.

A somewhat curious effect is that in order for our analysis to go through, we require the original SUBSET SUM instance to have few, say $O(1)$, distinct solutions. In order to

achieve this, we preprocess the instance by employing routine isolation techniques in $\mathbb{Z}_P$ *but implemented over* $\mathbb{Z}$ to control the number of solutions over $\mathbb{Z}$. The reason why we need to implement the preprocessing over $\mathbb{Z}$ rather than than work in the modular setting is that the dissection algorithm itself needs to be able to choose a modulus M very carefully to deliver the tradeoff, and that choice is incompatible with having an extra prime P for isolation. This is somewhat curious because, intuitively, the more solutions an instance has, the easier it should be to find one. The reason why that is not the case in our setting is that, further down in the recursion tree, when operating with a small modulus M, every original solution gives rise to many additional spurious solutions, and if there are too many original solutions there will be too many spurious solutions.

A further property needed to support the analysis is that the numbers in the SUBSET SUM instance must not be too large, in particular we need $\log t = O(n)$. This we can also achieve by a simple preprocessing step where we hash down modulo a random prime, but again with implementation over the integers for the same reason as above.

Related Work. The SUBSET SUM problem has recently been approached from related angles, with the interest in small space. Lokshtanov and Nederlof [9] show that the well-known pseudo-polynomial-time dynamic programming algorithm can be implemented in truly-polynomial space by algebraization. Kaski, Koivisto, and Nederlof [7] note that the sparsity of the dynamic programming table can be exploited to speedup the computations even if allowing only polynomial space.

Smooth space–time tradeoffs have been studied also for several other hard problems. Björklund et al. [2] derive a hybrid scheme for the Tutte polynomial that is a host of various counting problems on graphs. Koivisto and Parviainen [8] consider a class of permutation problems (including, e.g., the traveling salesman problem and the feedback arc set problem) and show that a natural hybrid scheme can be beaten by a partial ordering technique.

2 The Main Dissection Algorithm

Before describing the main algorithm, we condense some routine preprocessing steps into the following theorem.

Theorem 3 (†). *There is a polynomial-time randomized algorithm for preprocessing instances of* SUBSET SUM *which, given as input an instance* $(\boldsymbol{a}, t)$ *with* n *elements, outputs a collection of* $O(n^3)$ *instances* $(\boldsymbol{a}', t')$*, each with* n *elements and* $\log t' = O(n)$*, such that if* $(\boldsymbol{a}, t)$ *is a NO instance then so are all the new instances with probability* $1 - o(1)$*, and if* $(\boldsymbol{a}, t)$ *is a YES instance then with probability* $\Omega(1)$ *at least one of the new instances is a YES instance with at most* $O(1)$ *solutions.*

By applying this preprocessing we may assume that the main algorithm receives an input $(\boldsymbol{a}, t)$ that has $O(1)$ solutions and $\log t = O(n)$. We then introduce a random modulus M and transfer into a modular setting.

Algorithm 1: GENERATESOLUTIONS$(\boldsymbol{a}, t, M, \sigma)$

Data: $(\boldsymbol{a}, t, M)$ is an n-element MODULAR SUBSET SUM instance, $\sigma \in (0, 1]$
Result: Iterates over up to $\Theta^*(2^n/M)$ solutions of $(\boldsymbol{a}, t, M)$ while using space $O^*(2^{\sigma n})$

begin
  if $\sigma \geq 1/4$ **then**
    Report up to $\Theta^*(2^n/M)$ solutions using the Shroeppel-Shamir algorithm;
    return;
  Choose $\alpha \in (0, 1), \beta \in (0, 1)$ appropriately (according to Theorem 4) based on σ;
  Let M' be a factor of M of magnitude $\Theta(2^{\beta n})$;
  for $s' = 0, 1, \ldots, M' - 1$ **do**
    Allocate an empty lookup table;
    Let $\boldsymbol{l} = (a_1, a_2, \ldots, a_{\alpha n})$ be the first αn items of $\boldsymbol{a}$;
    Let $\boldsymbol{r} = (a_{\alpha n+1}, a_{\alpha n+2}, \ldots, a_n)$ be the remaining $(1 - \alpha)n$ items of $\boldsymbol{a}$;
    for $\boldsymbol{y} \in$ GENERATESOLUTIONS$(\boldsymbol{l}, s', M', \frac{\sigma}{\alpha})$ **do**
      Let $s = \sum_{i=1}^{\alpha n} a_i y_i \bmod M$;
      Store $[s \to \boldsymbol{y}]$ in the lookup table;
    for $\boldsymbol{z} \in$ GENERATESOLUTIONS$(\boldsymbol{r}, t - s', M', \frac{\sigma}{1-\alpha})$ **do**
      Let $s = t - \sum_{i=\alpha n+1}^{n} a_i z_i \bmod M$;
      foreach $[s \to \boldsymbol{y}]$ *in the lookup table* **do**
        Report solution $\boldsymbol{x} = (\boldsymbol{y}, \boldsymbol{z})$;
        if *at least* $\Theta^*(2^n/M)$ *solutions reported* **then**
          Stop iteration and **return**;
    Release the lookup table;

Definition 3. *An instance* $(\boldsymbol{a}, t, M)$ *of* MODULAR SUBSET SUM *consists of a vector* $\boldsymbol{a} \in \mathbb{Z}_{\geq 0}^n$*, a target* $t \in \mathbb{Z}_{\geq 0}$*, and a modulus* $M \in \mathbb{Z}_{\geq 1}$*. A solution of* $(\boldsymbol{a}, t, M)$ *is a vector* $\boldsymbol{x} \in \{0, 1\}^n$ *such that* $\sum_{i=1}^{n} a_i x_i \equiv t \pmod{M}$.

The reason why we transfer to the modular setting is that the recursive dissection strategy extensively uses the fact that we have available a sufficiently rich family of homomorphisms to split the search space. In particular, in the modular setting this corresponds to the modulus M being "sufficiently divisible" (in a sense to be made precise later) to obtain control of the recursion.

Pseudocode for the main algorithm is given in Algorithm 1. In addition to the modular instance $(\boldsymbol{a}, t, M)$, the algorithm accepts as further input the space parameter $\sigma \in (0, 1]$.

The key high-level idea in the algorithm is to "meet in the middle" by splitting an instance of n items to two sub-instances of αn items and $(1-\alpha)n$ items, guessing (over a smaller modulus M' that divides M) what the sum should be after the first and before the second sub-instance, and then recursively solving the two sub-instances subject to the guess. Figure 2 illustrates the structure of the algorithm.

We continue with some further high-level remarks.

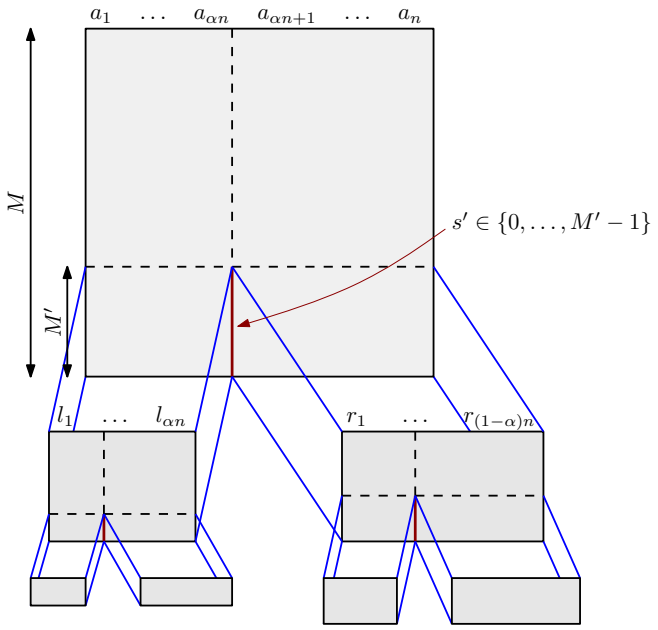


Fig. 2. Illustration of the recursive dissections made by the algorithm

1. In the algorithm, two key parameters α and β are chosen, which control how the MODULAR SUBSET SUM instance is subdivided for the recursive calls. The precise choice of these parameters is given in Theorem 4 below, but at this point the reader is encouraged to simply think of them as some parameters which should be chosen appropriately so as to optimize running time.
2. The algorithm also chooses a factor M' of M such that $M' = \Theta(2^{\beta n})$. The existence of sufficient factors at all levels of recursion is established in Section 3.
3. The algorithm should be viewed as an *iterator* over solutions. In other words, the algorithm has an internal state, and a *next item* functionality that we tacitly use by writing a for-loop over all solutions generated by the algorithm, which should be interpreted as a short-hand for repeatedly asking the iterator for the next item.
4. The algorithm uses a "bailout mechanism" to control the running time and space usage. Namely, each recursive call will bail out after $\Theta^*(2^n/M)$ solutions are reported. (The precise bailout bound has a further multiplicative factor polynomial in n that depends on the top-level value of σ.) A preliminary intuition for the bound is that this is what one would expect to receive in a particular congruence class modulo M if the 2^n possible sums are randomly placed into the congruence classes.

As a warmup to the analysis, let us first observe that, if we did not have the bailout step in line 19, correctness of the algorithm would be more or less immediate: for any solution $\boldsymbol{x}$ of $(\boldsymbol{a}, t, M)$, let $s = \sum_{i=1}^{\alpha n} a_i x_i \bmod M$. Then, when $s' = s \bmod M'$ in the outer for-loop (line 7), by an inductive argument we will find $\boldsymbol{y}$ and $\boldsymbol{z}$ in the two separate recursive branches and join the two partial solutions to form $\boldsymbol{x}$.

The challenge, of course, is that without the bailout mechanism we lack control over the resource consumption of the algorithm. Even though we have applied isolation to guarantee that there are not too many solutions of the top-level instance $(\boldsymbol{a}, t)$, it may be that some branches of the recursion generate a huge number of solutions, affecting both running time and space (since we store partial solutions in a lookup table).

Let us then proceed to analyzing the algorithm with the bailout mechanism in place. The two main claims are as follows.

Theorem 4 (†)**.** *Given a space budget $\sigma \in (0, 1]$ and $M \geq 2^n$, if in each recursive step of Algorithm 1 the parameters α and β are chosen as*

$$\alpha = 1 - \tau(\sigma) \qquad \text{and} \qquad \beta = 1 - \tau(\sigma) - \sigma\,, \tag{2}$$

then the algorithm runs in $O^*(2^{\tau(\sigma)n})$ time and $O^*(2^{\sigma n})$ space.

Theorem 5. *For every $\sigma \in (0, 1]$ there is a randomized algorithm that runs in time polynomial in n and chooses a top-level modulus $M \geq 2^n$ so that Algorithm 1 reports a solution of the non-modular instance $(\boldsymbol{a}, t)$ with high probability over the choices of M, assuming that at least one and at most $O(1)$ solutions exist and that $\log t = O(n)$.*

We prove Theorem 5 in Section 3; the proof of Theorem 4 appears in the full version of this conference abstract.

Let us however here briefly discuss the specific choice of α and β in Theorem 4. We arrived at (2) by analyzing the recurrence relation describing the running time of Algorithm 1. Unfortunately this recurrence in its full form is somewhat complicated, and our process of coming up with (2) involved a certain amount of experimenting and guesswork. We do have some guiding (non-formal) intuition which might be instructive:

1. One needs to make sure that $\alpha - \beta \leq \sigma$. This is because for a random instance, the left subinstance is expected to have roughly $2^{(\alpha-\beta)n}$ solutions, and since we need to store these there had better be at most $2^{\sigma n}$ of them.
2. Since $\beta \geq \alpha - \sigma$ and β has a very direct impact on running time (due to the $2^{\beta n}$ time outer loop), one will typically want to set α relatively small. The tension here is of course that the smaller α becomes, the larger $1 - \alpha$ (that is, the size of the right subinstance) becomes.
3. Given this tension, setting $\alpha - \beta = \sigma$ is natural.

So in an intuitive sense, the bottleneck for space comes from the left subinstance, or rather the need to store all the solutions found for the left subinstance (this is not technically true since we give the right subinstance $2^{\sigma n}$ space allowance as well), whereas the bottleneck for time comes from the right subinstance, which tends to be much larger than the left one.

3 Choice of Modulus and Analysis of Correctness

In this section we prove Theorem 5, giving the correctness of the dissection algorithm.

3.1 The Dissection Tree

Now that we have the choice of α and β in Algorithm 1, we can look more closely at the recursive structure of the algorithm. To this end, we make the following definition.

Definition 4 (Dissection tree). *For $\sigma \in (0, 1]$, the* dissection tree $\mathcal{DT}(\sigma)$ *is the ordered binary tree defined as follows. If $\sigma \geq 1/4$ then $\mathcal{DT}(\sigma)$ is a single node. Otherwise, let $\alpha = 1 - \tau(\sigma)$. The left child of $\mathcal{DT}(\sigma)$ is $\mathcal{DT}(\sigma/\alpha)$, and the right child of $\mathcal{DT}(\sigma)$ is $\mathcal{DT}(\sigma/(1-\alpha))$.*

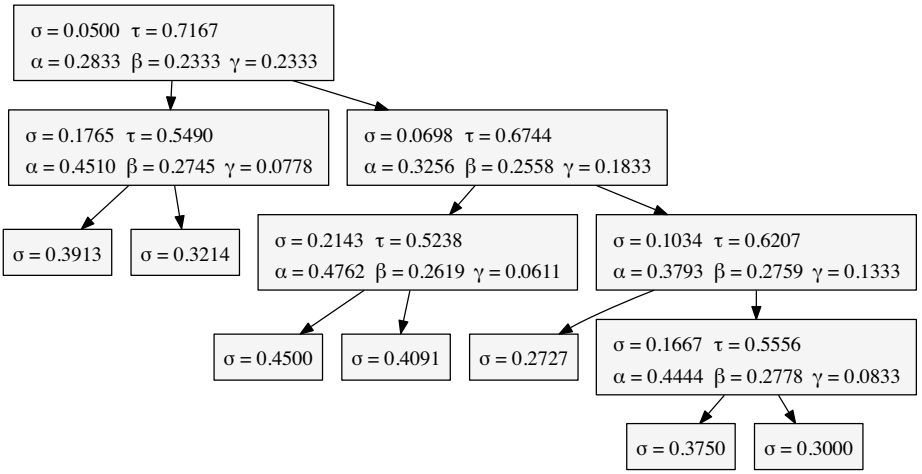


Fig. 3. The dissection tree $\mathcal{DT}(0.05)$. For each internal node v, we display the parameters $\sigma_v, \tau_v = \tau(\sigma_v), \alpha_v, \beta_v, \gamma_v$ as defined in Section 3.

Figure 3 shows $\mathcal{DT}(0.05)$. The dissection tree captures the essence of the recursive behaviour of the dissection algorithm when being run with parameter σ. The actual recursion tree of the dissection algorithm is huge due to the for-loop over s' in line 7, but if we consider a fixed choice of s' in every recursive step then the recursion tree of the algorithm becomes identical to the corresponding dissection tree.

Lemma 1. *The recursion depth of Algorithm 1 is the height of $\mathcal{DT}(\sigma)$. In particular, the recursion depth is a constant that depends only on σ.*

We now describe how to choose *a priori* a random M that is "sufficiently divisible" for the algorithm's desires, and to show correctness of the algorithm.

Fix a choice of the top-level value $\sigma \in (0, 1]$. Consider the corresponding dissection tree $\mathcal{DT}(\sigma)$. For each node v of $\mathcal{DT}(\sigma)$, write σ_v for the associated σ value. For an internal node v let us also define $\alpha_v = 1 - \tau(\sigma_v)$ and $\beta_v = 1 - \sigma_v - \tau(\sigma_v)$. In other words, if v_1 and v_2 are the two child nodes of v, then $\sigma_{v_1} = \sigma_v/\alpha_v$ and $\sigma_{v_2} = \sigma_v/(1-\alpha_v)$. Finally, define $\gamma_v = \beta_v \cdot \sigma/\sigma_v$.

Observe that each recursive call made by Algorithm 1 is associated with a unique internal node v of the dissection tree $\mathcal{DT}(\sigma)$.

Lemma 2. *Each recursive call associated with an internal node v requires a factor M' of magnitude $\Theta^*(2^{\gamma_v n})$.*

Proof. Telescope a product of the ratio σ_p/σ_u for a node u and its parent p along the path from v to the root node. Each such σ_p/σ_u is either α_u or $1-\alpha_u$ depending on whether it is a left branch or right branch—precisely the factor by which n decreases. □

3.2 Choosing the Modulus

The following lemma contains the algorithm that chooses the random modulus.

Lemma 3. *For every $\sigma \in (0,1]$ there exists a randomized algorithm that, given integers n and $b = O(n)$ as input, runs in time polynomial in n and outputs for each internal node $v \in \mathcal{DT}(\sigma)$ random moduli M_v, M'_v such that, for the root node $r \in \mathcal{DT}(\sigma)$, $M_r \geq 2^b$, and furthermore for every internal node v:*

1. *M'_v is of magnitude $\Theta(2^{\gamma_v n})$,*
2. *$M_v = M'_p$, where p is the parent of v,*
3. *M'_v divides M_v, and*
4. *for any fixed integer $1 \leq Z \leq 2^b$, the probability that M'_v divides Z is $O^*(1/M'_v)$.*

Proof. Let $0 < \lambda_1 < \lambda_2 < \cdots < \lambda_k$ be the set of distinct values of γ_v ordered by value, and let $\delta_i = \lambda_i - \lambda_{i-1}$ be their successive differences (where we set $\lambda_0 = 0$ so that $\delta_1 = \lambda_1$). Since $\mathcal{DT}(\sigma)$ depends only σ and not on n, we have $k = O(1)$. For each $1 \leq i \leq k$ independently, let p_i be a uniform random prime from the interval $[2^{\delta_i n}, 2 \cdot 2^{\delta_i n}]$.

For a node v such that $\gamma_v = \lambda_j$, let $M'_v = \prod_{i=1}^{j} p_j$. Condition 1 then holds by construction. The values of M_v are determined for all nodes except the root through condition 2; for the root node r we set $M_r = p_0 M'_r$, where p_0 is a random prime of magnitude $2^{\Theta(n)}$ to make sure that $M_r \geq 2^b$.

To prove condition 3 note that for any node v with parent p, we need to prove that M'_v divides M'_p. Let j_v be such that $\lambda_{j_v} = \gamma_v$ and j_p such that $\lambda_{j_p} = \gamma_p$. Noting that the value of γ_v decreases as one goes down the dissection tree, it then holds that $j_v < j_p$, from which it follows that $M'_v = \prod_{i=1}^{j_v} p_i$ divides $M'_p = \prod_{i=1}^{j_p} p_i$.

Finally, for condition 4, again let j be such that $\lambda_j = \gamma_v$, and observe that in order for Z to divide M'_v it must have all the factors $p_1, p_2, \ldots, p_j$. For each $1 \leq i \leq j$, Z can have at most $\frac{\log_2 Z}{\delta_i n} = O(1)$ different factors between $2^{\delta_i n}$ and $2 \cdot 2^{\delta_i n}$, so by the Prime Number Theorem, the probability that p_i divides Z is at most $O(n 2^{-\delta_i n})$. As the p_i's are chosen independently the probability that Z divides all of $p_1, p_2, \ldots, p_j$ (that is, M'_v) is $O(n^j 2^{-(\delta_1+\delta_2+\ldots+\delta_j)n}) = O(n^k 2^{-\gamma_v n}) = O^*(1/M'_v)$, as desired. □

3.3 Proof of Correctness

We are now ready to prove the correctness of the entire algorithm, assuming preprocessing and isolation has been carried out.

Theorem 5 (restated)**.** *For every $\sigma \in (0,1]$ there is a randomized algorithm that runs in time polynomial in n and chooses a top-level modulus $M \geq 2^n$ so that Algorithm 1 reports a solution of the non-modular instance $(\boldsymbol{a}, t)$ with high probability over the choices of M, assuming that at least one and at most $O(1)$ solutions exist and that $\log t = O(n)$.*

Proof. The modulus M is chosen using Lemma 3, with b set to $\max\{n, \log nt\} = \Theta(n)$. Specifically, it is chosen as M_r for the root node r of $\mathcal{DT}(\sigma)$.

Fix a solution $\boldsymbol{x}^*$ of $(\boldsymbol{a}, t)$, that is, $\sum_{i=1}^{n} a_i x_i^* = t$. (Note that this is an equality over the integers and not a modular congruence.) By assumption such an $\boldsymbol{x}^*$ exists and there are at most $O(1)$ choices.

If $\sigma \geq 1/4$, the top level recursive call executes the Schroeppel–Shamir algorithm and a solution will be discovered. So suppose that $\sigma \in (0, 1/4)$.

For an internal node $v \in \mathcal{DT}(\sigma)$ consider a recursive call associated with v, and let $L_v \subseteq [n]$ (resp. $R_v \subseteq [n]$) be the set of $\alpha_v n_v$ (resp. $(1-\alpha_v)n_v$) indices of the items that are passed to the left (resp. right) recursive subtree of v. Note that these indices are with respect to the top-level instance, and that they do not depend on the choices of s' made in the recursive calls. Let $s'_v \in \{0, \ldots, M'_v - 1\}$ be the choice of s' that could lead to the discovery of $\boldsymbol{x}^*$, in other words $s'_v = \sum_{i \in L_v} a_i x_i^* \bmod M'_v$. Let $I_v = L_v \cup R_v$.

For a leaf node $v \in \mathcal{DT}(\sigma)$ and its parent p, define $I_v = L_p$ if v is a left child of p, and $I_v = R_p$ if v is a right child of p.

We now restrict our attention to the part of the recursion tree associated with the discovery of $\boldsymbol{x}^*$, or in other words, the recursion tree obtained by fixing the value of s' to s'_v in each recursive step, rather than trying all possibilities. This restricted recursion tree is simply $\mathcal{DT}(\sigma)$. Thus the set of items $\boldsymbol{a}_v = (a_i)_{i \in I_v}$ and the target t_v associated with v is well-defined for all $v \in \mathcal{DT}(\sigma)$.

Denote by $B(v)$ the event that $(\boldsymbol{a}_v, t_v, M_v)$ has more than $O^*(2^{n_v}/M_v)$ solutions. Clearly, if $B(v)$ does not happen then there can not be a bailout at node v.[1] We will show that $\cup_{v \in \mathcal{DT}(\sigma)} B(v)$ happens with probability $o(1)$ over the choices of $\{M_v, M'_v\}$ from Lemma 3, which thus implies that $\boldsymbol{x}^*$ is discovered with probability $1 - o(1)$. Because $\mathcal{DT}(\sigma)$ has $O(1)$ nodes, by the union bound it suffices to show that $\Pr[B(v)] = o(1)$ for every $v \in \mathcal{DT}(\sigma)$.

Consider an arbitrary node $v \in \mathcal{DT}(\sigma)$. There are two types of solutions $\boldsymbol{x}_v$ of the instance $(\boldsymbol{a}_v, t_v, M_v)$ associated with v.

First, a vector $\boldsymbol{x}_v \in \{0,1\}^{n_v}$ is a solution if $\sum_{i=1}^{n_v} a_{v,i} x_{v,i} = \sum_{i \in I_v} a_i x_i^*$. (Note that this is an equality over the integers, not a modular congruence.) Because there are at most $O(1)$ solutions to the top-level instance, there are at most $O(1)$ such vectors $\boldsymbol{x}_v$. Indeed, otherwise we would have more than $O(1)$ solutions of the top level instance, a contradiction.

[1] The converse is not true though: it can be that $B(v)$ happens but a bailout happens in one (or both) of the two subtrees of v, causing the recursive call associated with node v to not find all the solutions to $(\boldsymbol{a}_v, t_v, M_v)$ and thereby not bail out.

Second, consider a vector $\boldsymbol{x}_v \in \{0,1\}^{n_v}$ such that $\sum_{i=1}^{n_v} a_{v,i}x_{v,i} \neq \sum_{i\in I_v} a_i x_i^*$ (over the integers). Let $Z = |\sum_{i=1}^{n_v} a_{v,i}x_{v,i} - \sum_{i\in I_v} a_i x_i^*| \neq 0$. Such a vector $\boldsymbol{x}_v$ is a solution of $(\boldsymbol{a}_v, t_v, M_v)$ only if M_v divides Z. Since $\log t = O(n)$ and $1 \leq Z \leq nt$, by Lemma 3, item 4 we have that Z is divisible by M_v with probability $O^*(1/M_v)$.

From the two cases it follows that the expected number of solutions $\boldsymbol{x}_v$ of $(\boldsymbol{a}_v, t_v, M_v)$ is $E = O^*(2^{n_v}/M_v)$. (We remark that the degree in the suppressed polynomial depends on σ but not on n.) Setting the precise bailout threshold to $n \cdot E$, we then have by Markov's inequality that $\Pr[B(v)] = \Pr[\#\text{solutions } \boldsymbol{x}_v > nE] < 1/n = o(1)$, as desired. Since v was arbitrary, we are done. □

References

1. Becker, A., Coron, J.-S., Joux, A.: Improved generic algorithms for hard knapsacks. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 364–385. Springer, Heidelberg (2011)
2. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Computing the Tutte polynomial in vertex-exponential time. In: FOCS, pp. 677–686. IEEE Computer Society (2008)
3. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 719–740. Springer, Heidelberg (2012)
4. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM 21(2), 277–292 (1974)
5. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 235–256. Springer, Heidelberg (2010)
6. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations. The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972)
7. Kaski, P., Koivisto, M., Nederlof, J.: Homomorphic hashing for sparse coefficient extraction. In: Thilikos, D.M., Woeginger, G.J. (eds.) IPEC 2012. LNCS, vol. 7535, pp. 147–158. Springer, Heidelberg (2012)
8. Koivisto, M., Parviainen, P.: A space-time tradeoff for permutation problems. In: Charikar, M. (ed.) SODA, pp. 484–492. SIAM (2010)
9. Lokshtanov, D., Nederlof, J.: Saving space by algebraization. In: Schulman, L.J. (ed.) STOC, pp. 321–330. ACM (2010)
10. Schroeppel, 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)
11. Wang, J.: Space-efficient Las Vegas algorithms for K-SUM. CoRR abs/1303.1016 (2013)
12. Woeginger, G.J.: Open problems around exact algorithms. Discrete Applied Mathematics 156(3), 397–405 (2008)

On the Extension Complexity of Combinatorial Polytopes*

David Avis[1,2,**] and Hans Raj Tiwary[3,***]

[1] GERAD and School of Computer Science, McGill University,
3480 University Street, Montreal, Quebec, Canada H3A 2A7
[2] Graduate School of Informatics, Kyoto University, Sakyo-ku, Yoshida Yoshida,
Kyoto 606-8501, Japan
[3] Department of Mathematics, Université Libre de Bruxelles,
Boulevard du Triomphe, B-1050 Brussels, Belgium
avis@cs.mcgill.ca,
hans.raj.tiwary@ulb.ac.be

Abstract. In this paper we extend recent results of Fiorini et al. on the extension complexity of the cut polytope and related polyhedra. We first describe a lifting argument to show exponential extension complexity for a number of NP-complete problems including subset-sum and three dimensional matching. We then obtain a relationship between the extension complexity of the cut polytope of a graph and that of its graph minors. Using this we are able to show exponential extension complexity for the cut polytope of a large number of graphs, including those used in quantum information and suspensions of cubic planar graphs.

1 Introduction

In formulating optimization problems as linear programs (LP), adding extra variables can greatly reduce the size of the LP [5]. However, it has been shown recently that for some polytopes one cannot obtain polynomial size LPs by adding extra variables [8, 12]. In a recent paper [8], Fiorini et.al. proved such results for the cut polytope, the traveling salesman polytope, and the stable set polytope for the complete graph K_n. In this paper, we extend the results of Fiorini et. al. to several other interesting polytopes. We do not claim novelty of our techniques, in that they have been used - in particular - by Fiorini et. al. Our motivation arises from the fact that there is a strong indication that NP-hard problems require superpolynomial sized linear programs. We make a step in this direction by giving a simple technique that can be used to translate NP-completeness reductions into lower bounds for a number of interesting polytopes.

* Due to space constraints many proofs have been omitted here. A full version containing all proofs is available at http://arxiv.org/abs/1302.2340

** Research supported by the NSERC and JSPS.
*** Research supported by FNRS.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 57–68, 2013.

Cut polytope and related polytopes. The cut polytope arises in many application areas and has been extensively studied. Formal definitions of this polytope and its relatives are given in the next section. A comprehensive compilation of facts about the cut polytope is contained in the book by Deza and Laurent [7]. Optimization over the cut polytope is known as the max cut problem, and was included in Karp's original list of problems that he proved to be NP-hard. For the complete graph with n nodes, a complete list of the facets of the cut polytope $\mathrm{CUT}_n^{\square}$ is known for $n \leq 7$ (see Section 30.6 of [7]), as well as many classes of facet producing valid inequalities. The hypermetric inequalities (see Chapter 28 of [7]) are examples of such a class, and it is known that an exponential number of them are facet inducing. Less is known about classes of facets for the cut polytope of an arbitrary graph, $\mathrm{CUT}^{\square}(G)$. Interest in such polytopes arises because of their application to fundamental problems in physics.

In quantum information theory, the cut polytope arises in relation to Bell inequalities. These inequalities, a generalization of Bell's original inequality [4], were introduced to better understand the nonlocality of quantum physics. Bell inequalities for two parties are inequalities valid for the cut polytope of the complete tripartite graph $\mathrm{K}_{1,n,n}$. Avis, Imai, Ito and Sasaki [2] proposed an operation named *triangular elimination*, which is a combination of zero-lifting and Fourier-Motzkin elimination (see e.g. [14]) using the triangle inequality. They proved that triangular elimination maps facet inducing inequalities of the cut polytope of the complete graph to facet inducing inequalities of the cut polytope of $\mathrm{K}_{1,n,n}$. Therefore a standard description of such polyhedra contains an exponential number of facets.

In [1] the method was extended to obtain facets of $\mathrm{CUT}^{\square}(G)$ for an arbitrary graph G from facets of $\mathrm{CUT}_n^{\square}$. For most, but not all classes of graphs, $\mathrm{CUT}^{\square}(G)$ has an exponential number of facets. An interesting exception are the graphs with no K_5 minor. Results of Seymour for the cut cone, extended by Barahona and Mahjoub to the cut polytope (see Section 27.3.2 of [7]), show that the facets in this case are just projections of triangle inequalities. It follows that the max cut problem for a graph G on n vertices with no K_5 minor can be solved in polynomial time by optimizing over the *semi-metric polytope*, which has $O(n^3)$ facets. Another way of expressing this is to say that in this case $\mathrm{CUT}^{\square}(G)$ has $O(n^3)$ *extension complexity*, a notion that will be discussed next.

Extended formulations and extensions Even for polynomially solvable problems, the associated polytope may have an exponential number of facets. By working in a higher dimensional space it is often possible to decrease the number of constraints. In some cases, a polynomial increase in dimension can yield an exponential decrease in the number of constraints. The previous paragraph contained an example of this.

For NP-hard problems the notion of extended formulations also comes into play. Even though a natural LP formulation of such a problem has exponential size, this does not rule out a polynomial size formulation in higher dimensions.

In a groundbreaking paper, Yannakakis [13] proved that every symmetric LP for the Travelling Salesman Problem (TSP) has exponential size. Here, an LP

is called *symmetric* if every permutation of the cities can be extended to a permutation of all the variables of the LP that preserves the constraints of the LP. This result refuted various claimed proofs of a polynomial time algorithm for the TSP. In 2012 Fiorini et al. [8] proved that the max cut problem also requires exponential size if it is to be solved as an LP. Using this result, they were able to drop the symmetric condition, required by Yannakakis, to get a general super polynomial bound for LP formulations of the TSP.

2 Preliminaries

We briefly review basic notions about the cut polytope and extension complexity used in later sections. Definitions, theorems and other results for the cut polytope stated in this section are from [7], which readers are referred to for more information. We assume that readers are familiar with basic notions in convex polytope theory such as convex polytope, facet, projection and Fourier-Motzkin elimination. Readers are referred to a textbook [14] for details.

Throughout this paper, we use the following notation. For a graph $G = (V, E)$ we denote the edge between two vertices u and v by uv, and the neighbourhood of a vertex v by $\mathrm{N}_G(v)$. We let $[n]$ denote the integers $\{1, 2, ..., n\}$.

2.1 Cut Polytope and Its Relatives

The *cut polytope* of a graph $G = (V, E)$, denoted $\mathrm{CUT}^{\square}(G)$, is the convex hull of the cut vectors $\boldsymbol{\delta}_G(S)$ of G defined by all the subsets $S \subseteq V$ in the $|E|$-dimensional vector space $\mathbb{R}^E$. The cut vector $\boldsymbol{\delta}_G(S)$ of G defined by $S \subseteq V$ is a vector in $\mathbb{R}^E$ whose uv-coordinate is defined as follows: $\delta_{uv}(S) = 1$ if $|S \cap \{u, v\}| = 1$, and $\delta_{uv}(S) = 0$ otherwise, for $uv \in E$. If G is the complete graph K_n, we simply denote $\mathrm{CUT}^{\square}(\mathrm{K}_n)$ by $\mathrm{CUT}^{\square}_n$.

We now describe an important well known general class of valid inequalities for $\mathrm{CUT}^{\square}_n$ (see, e.g. [7], Ch. 28).

Lemma 1. *For any $n \geq 2$, let $b_1, b_2, ..., b_n$ be any set of n integers. The following inequality is valid for $\mathrm{CUT}^{\square}_n$:*

$$\sum_{1 \leq i < j \leq n} b_i b_j x_{ij} \leq \left\lfloor \frac{(\sum_{i=1}^{n} b_i)^2}{4} \right\rfloor \tag{1}$$

The inequality (1) is called *hypermetric* (respectively, of *negative type*) if the integers b_i can be partitioned into two subsets whose sum differs by one (respectively, zero). A simple example of hypermetric inequalities are the triangle inequalities, obtained by setting three of the b_i to be +/- 1 and the others to be zero. The most basic negative type inequality is non-negativity, obtained by setting one b_i to 1, another one to -1, and the others to zero.

For any fixed n there are an infinite number of hypermetric inequalities, but all but a finite number are redundant. This non-trivial fact was proved by Deza, Grishukhin and Laurent (see [7] Section 14.2) and allows us to define the *hypermetric polytope*, which we will refer to again later.

2.2 Extended Formulations and Extensions

An *extended formulation* (EF) of a polytope $P \subseteq \mathbb{R}^d$ is a linear system

$$Ex + Fy = g,\ y \geqslant \mathbf{0} \tag{2}$$

in variables $(x, y) \in \mathbb{R}^{d+r}$, where E, F are real matrices with d, r columns respectively, and g is a column vector, such that $x \in P$ if and only if there exists y such that (2) holds. The *size* of an EF is defined as its number of *inequalities* in the system.

An *extension* of the polytope P is another polytope $Q \subseteq \mathbb{R}^e$ such that P is the image of Q under a linear map. Define the *size* of an extension Q as the number of facets of Q. Furthermore, define the *extension complexity* of P, denoted by $\mathrm{xc}(P)$, as the minimum size of any extension of P.

In this paper we make use of the machinery developed and described in Fiorini et al. [8]. The reader is referred to the original paper for more details and proofs. The main result of Fiorini et al. [8] that we are interested in is the following

Theorem 1 (Lower Bound Theorem). $\mathrm{xc}(\mathrm{CUT}_n^{\square}) \geqslant 2^{\Omega(n)}$.

2.3 Proving Lower Bounds for Extension Complexity

We now note two observations that are useful in translating results from one polytope to another. Let P and Q be two polytopes. Then,

Proposition 1. *If P is a projection of Q then* $\mathrm{xc}(P) \leqslant \mathrm{xc}(Q)$.

Proposition 2. *If P is a face of Q then* $\mathrm{xc}(P) \leqslant \mathrm{xc}(Q)$.

Naturally there are many other cases where the conditions of neither of these propositions apply and yet a lower bounding argument for one polytope can be derived from another. However we would like to point out that these two propositions already seem to be very powerful. In fact, out of the three lower bounds proved by Fiorini et. al. [8] two (for $\mathrm{TSP}(n)$ and $\mathrm{STAB}(n)$) use these propositions, while the lower bound on the cut polytope is obtained by showing a direct embedding of a matrix with high nonnegative rank in the slack matrix of $\mathrm{CUT}_n^{\square}$.

In the next section we will use these propositions to show superpolynomial lower bounds on the extension complexities of polytopes associated with four NP-hard problems.

3 Polytopes for Some NP-Hard Problems

In this section we use the method of Section 2.3 to show super polynomial extension complexity for polytopes related to the following problems: subset sum, 3-dimensional matching and stable set for cubic planar graphs. These proofs are derived by applying this method to standard reductions from 3SAT, which is our starting point.

3SAT. For any given 3SAT formula Φ with n variables in conjunctive normal form define the polytope $\mathrm{SAT}(\Phi)$ as the convex hull of all satisfying assignments. That is, $\mathrm{SAT}(\Phi) := \mathrm{conv}(\{x \in [0,1]^n \mid \Phi(x) = 1\})$. The following theorem and its proof are implicit in [8], who make use of the correlation polytope.

Theorem 2. *For every n there exists a 3SAT formula Φ with $O(n)$ variables and $O(n)$ clauses such that* $\mathrm{xc}(\mathrm{SAT}(\Phi)) \geqslant 2^{\Omega(\sqrt{n})}$.

Subset Sum. The subset sum problem is a special case of the knapsack problem. Given a set of n integers $A = \{a_1, \ldots, a_n\}$ and another integer b, the subset sum problems asks whether any subset of A sums exactly to b. Define the subset sum polytope $\mathrm{SUBSETSUM}(A, b)$ as the convex hull of all characteristic vectors of the subsets of A whose sum is exactly b. That is, $\mathrm{SUBSETSUM}(A, b) := \mathrm{conv}\left(\{x \in [0,1]^n \mid \sum_{i=1}^n a_i x_i = b\}\right)$

The subset sum problem then is asking whether $\mathrm{SUBSETSUM}(A, b)$ is empty for a given set A and integer b. Note that this polytope is a face of the knapsack polytope $\mathrm{KNAPSACK}(A, b) := \mathrm{conv}\left(\{x \in [0,1]^n \mid \sum_{i=1}^n a_i x_i \leqslant b\}\right)$

In this subsection we prove that the subset sum polytope (and hence the knapsack polytope) can have superpolynomial extension complexity.

Theorem 3. *For every 3SAT formula Φ with n variables and m clauses, there exists a set of integers $A(\Phi)$ and integer b with $|A| = 2n + 2m$ such that* $\mathrm{SAT}(\Phi)$ *is the projection of* $\mathrm{SUBSETSUM}(A, b)$.

Proof. Suppose formula Φ is defined in terms of variables $x_1, x_2, ..., x_n$ and clauses $C_1, C_2, ..., C_m$. We use a standard reduction from 3SAT to subset sum (e.g., [6], Section 34.5.5). We define $A(\Phi)$ and b as follows. Every integer in $A(\Phi)$ as well as b is an $(n + m)$-digit number (in base 10). The first n bits correspond to the variables and the last m bits correspond to each of the clauses. $b_j = 1$, if $1 \leqslant j \leqslant n$ and $b_j = 4$, if $n + 1 \leqslant j \leqslant n + m$.

Next we construct $2n$ integers v_i, v'_i for $i \in \{1, \ldots, n\}$, and $2m$ integers s_i, s'_i for $i \in \{1, \ldots, m\}$ as follows: $v_{ij} = 1$, if $j = i$ or $x_i \in C_{j-n}$ and 0, otherwise, $v'_{ij} = 1$, if $j = i$ or $\overline{x}_i \in C_{j-n}$ and 0, otherwise. $s_{ij} = 1$, if $j = n + i$ and 0 otherwise, $s'_{ij} = 2$, if $j = n + i$ and 0 otherwise.

We define the set $A(\Phi) = \{v_1, \ldots, v_n, v'_1, \ldots, v'_n, s_1, \ldots, s_m, s'_1, \ldots, s'_m\}$.

Consider the subset-sum instance with $A(\Phi), b$ as constructed above for any 3SAT instance Φ. Let S be any subset of $A(\Phi)$. If the elements of S sum exactly to b then it is clear that for each $i \in \{1, \ldots, n\}$ exactly one of v_i, v'_i belong to S. Furthermore, setting $x_i = 1$ if $v_i \in S$ or $x_i = 0$ if $v'_i \in S$ satisfies every clause. Thus the characteristic vector of S restricted to $\{v_1, \ldots, v_n\}$ is a satisfying assignment for the corresponding SAT formula.

Also, if Φ is satisfiable then the instance of subset sum thus created has a solution corresponding to each satisfying assignment: Pick v_i if $x_i = 1$ or v'_i if $x_i = 0$ in an assignment. Since the assignment is satisfying, every clause is satisfied and so the sum of digits corresponding to each clause is at least 1. Therefore, for a clause C_j either s_j or s'_j or both can be picked to ensure that

the sum of the corresponding digits is exactly 4. Note that there is unique way to do this.

This shows that every vertex of the subset sum polytope SUBSETSUM$(A(\Phi), b)$ projects to a vertex of SAT(Φ) and every vertex of SAT(Φ) can be lifted to a vertex of SUBSETSUM$(A(\Phi), b)$,. The projection is defined by dropping every coordinate except those corresponding to the numbers v_i in the reduction described above. The lifting is defined by the procedure in the proceeding paragraph. Hence, SAT(Φ) is a projection of SUBSETSUM$(A(\Phi), b)$. □

Combining the preceding two theorems we obtain the following.

Corollary 1. *For every natural number $n \geqslant 1$, there exists an instance A, b of the subset-sum problem with $O(n)$ integers in A such that* xc(SUBSETSUM(A, b)) $\geqslant 2^{\Omega(\sqrt{n})}$.

3d-Matching. Consider a hypergraph $G = ([n], E)$, where E contains triples for some $i, j, k \in [n]$ where i, j, k are distinct. A subset $E' \subseteq E$ is said to be a 3-dimensional matching if all the triples in E' are disjoint. The 3d-matching polytope 3DM(G) is defined as the convex hull of the characteristic vectors of every 3d-matching of G. That is, 3DM(G) := conv$(\{\chi(E') \mid E' \subseteq E$ is a 3d-matching$\})$

The 3d-matching problem asks: given a hypergraph G, does there exist a 3d-matching that covers all vertices? This problem is known to be NP-complete and was one of Karp's 21 problems proved to be NP-complete [9, 11]. Note that this problem can be solved by linear optimization over the polytope 3DM(G) and therefore it is to be expected that 3DM(G) would not have a polynomial size extended formulation.

Now we show that the 3d-matching polytope has superpolynomial extension complexity in the worst case. We prove this using a standard reduction from 3SAT to 3d-Matching used in the NP-completeness proof for the later problem (See [9]). The form of this reduction, which is very widely used, employs a gadget for each variable along with a gadget for each clause. We omit the exact details for the reduction here because we are only interested in the correctness of the reduction and the variable gadget (See Figure 1).

In the reduction, any 3SAT formula Φ is converted to an instance of a 3d-matching by creating a set of hyperedges for every variable (See Figure 1) along with some other hyperedges that does not concern us for our result. The crucial property that we require is the following: any satisfiable assignment of Φ defines some (possibly more than one) 3d-matching. Furthermore, in any maximal matching either only the light hyperedges or only the dark hyperedges are picked,

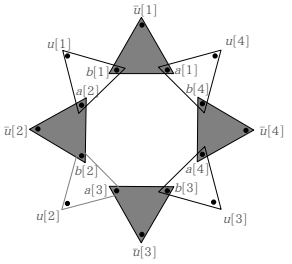


Fig. 1. Gadget for a variable

corresponding to setting the corresponding variable to, say, true or false respectively. Using these facts we can prove the following:

Theorem 4. *Let Φ be an instance of 3SAT and let H be the hypergraph obtained by the reduction above. Then* $\mathrm{SAT}(\Phi)$ *is the projection of a face of* $\mathrm{3DM}(H)$.

Proof. Let the number of hyperedges in the gadget corresponding to a variable x be $2k(x)$. Then, the number of hyperedges picked among these hyperedges in any matching in H is at most $k(x)$. Therefore, if $y_1, \ldots, y_{2k(x)}$ denote the variables corresponding to these hyperedges in the polytope $\mathrm{3DM}(H)$ then $\sum_{i=1}^{2k(x)} y_i \leqslant k(x)$ is a valid inequality for $\mathrm{3DM}(H)$. Consider the face F of $\mathrm{3DM}(H)$ obtained by adding the equality $\sum_{i=1}^{2k(x)} y_i = k(x)$ corresponding to each variable x appearing in Φ.

Any vertex of $\mathrm{3DM}(H)$ lying in F selects either all light hyperedges or all dark hyperedges. Therefore, projecting out all variables except one variable y_i corresponding to any fixed (arbitrarily chosen) light hyperedge for each variable in Φ gives a valid satisfying assignment for Φ and thus a vertex of $\mathrm{SAT}(\Phi)$. Alternatively, any vertex of $\mathrm{SAT}(\Phi)$ can be extended to a vertex of $\mathrm{3DM}(H)$ lying in F easily.

Therefore, $\mathrm{SAT}(\Phi)$ is the projection of F. □

The number of vertices in H is $O(nm)$ where n is the number of variables and m the number of clauses in Φ. Considering only the 3SAT formulae with high extension complexity, we have $m = O(n)$. Therefore, considering only the hypergraphs arising from such 3SAT formulae and using propositions 1 and 2, we have that

Corollary 2. *For every natural number $n \geqslant 1$, there exists a hypergraph H with $O(n)$ vertices such that* $\mathrm{xc}(\mathrm{3DM}(H)) \geqslant 2^{\Omega(n^{1/4})}$.

Stable Set for Cubic Planar Graphs. Now we show that $\mathrm{STAB}(G)$ can have superpolynomial extension complexity even when G is a cubic planar graph. Our starting point is the following result proved by Fiorini et. al. [8].

Theorem 5 ([8]). *For every natural number $n \geqslant 1$ there exists a graph G such that G has $O(n)$ vertices and $O(n)$ edges, and* $\mathrm{xc}(\mathrm{STAB}(G)) \geqslant 2^{\Omega(\sqrt{n})}$.

We start with this graph and convert it into a cubic planar graph G' with $O(n^2)$ vertices and extension complexity at least $2^{\Omega(\sqrt{n})}$.

Making a Graph Planar. For making any graph G planar without reducing the extension complexity of the associated stable set polytope, we use the same gadget used by Garey, Johnson and Stockmeyer [10] in the proof of NP-completeness of finding maximum stable set in planar graph. Start with any planar drawing of G and replace every crossing with the gadget H with 22 vertices shown in Figure 2 to obtain a graph G'. The following theorem shows that $\mathrm{STAB}(G)$ is the projection of a face of $\mathrm{STAB}(G')$.

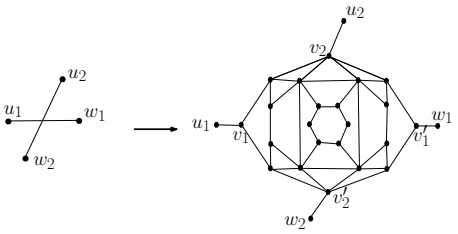


Fig. 2. Gadget to remove a crossing

Table 1. Values of s_{ij}

$i \backslash j$	2	1	0
2	9	8	7
1	9	9	8
0	8	8	7

Using the face $F := \text{STAB}(G') \bigcap_{i=1}^{k} \{x \mid \sum_{j \in V_{H_i}} x_j = 9\}$, where k denotes the number of gadgets itroduced in G and a proof similar to that of Theorem 1, we have the following:

Theorem 6. *Let G be a graph and let G' be obtained from a planar embedding of G by replacing every edge intersection with a gadget shown in Figure 2. Then,* $\text{STAB}(G)$ *is the projection of a face of* $\text{STAB}(G')$.

Since for any graph G with $O(n)$ edges, the number of gadgets introduced $k \leqslant O(n^2)$, we have that the graph G' in the above theorem has at most $O(n^2)$ vertices and edges. Therefore we have a planar graph G' with at most $O(n^2)$ vertices and $O(n^2)$ edges. This together with Theorem 5, Theorem 6 and propositions 1 and 2 yields the following corollary.

Corollary 3. *For every n there exists a planar graph G with $O(n^2)$ vertices and $O(n^2)$ edges such that* $\text{xc}(\text{STAB}(G)) \geqslant 2^{\Omega(\sqrt{n})}$.

Making a Graph Cubic. Suppose we have a graph G and we transform it into another graph G' by performing one of the following operations:

ReduceDegree: Replace a vertex v of G of degree $\delta \geqslant 4$ with a cycle $C_v = (v_1, v_1', \ldots, v_\delta, v_\delta')$ of length 2δ and connect the neighbours of v to alternating vertices $(v_1, v_2, \ldots, v_\delta)$ of the cycle.
RemoveBridge: Replace any degree two vertex v in G by a four cycle v_1, v_2, v_3, v_4. Let u and w be the neighbours of v in G. Then, add the edges (u, v_1) and (v_3, w). Also add the edge (v_2, v_4) in the graph.
RemoveTerminal: Replace any vertex with degree either two or three with a triangle. In case of degree one, attach any one vertex of the triangle to the erstwhile neighbour.

Theorem 7. *Let G be any graph and let G' be obtained by performing any number of operation ReduceDegree, RemoveBridge, or RemoveTerminal described above on G. Then* $\text{STAB}(G)$ *is the projection of a face of* $\text{STAB}(G')$.

Proof. Omitted. □

If G has n vertices and m edges then first applying operation ReduceDegree until every vertex has degree at most 3, and then applying operation RemoveBridge

and RemoveTerminal repeatedly until no vertex of degree 0, 1 or 2 is left, produces a graph that has $O(n+m)$ vertices and $O(n+m)$ edges. Furthermore, any application of the three operations do not make a planar graph non-planar. Combining this fact with Theorem 7, Corollary 3 and propositions 1 and 2, we have

Corollary 4. *For every natural number $n \geqslant 1$ there exists a cubic planar graph G with $O(n)$ vertices and edges such that* $\mathrm{xc}(\mathrm{STAB}(G)) \geqslant 2^{\Omega(n^{1/4})}$.

4 Extended Formulations for $\mathrm{CUT}^{\square}(G)$ and Its Relatives

We use the results described in the previous section to obtain bounds on the extension complexity of the cut polytope of graphs. We begin by reviewing the result in [8] for $\mathrm{CUT}_n^{\square}$ using a direct argument that avoids introducing correlation polytopes. For any integer $n \geq 2$ consider the integers $b_1 = ... = b_{n-1} = 1$ and $b_n = 3 - n$. Let $b = (b_1, b_2, ..., b_n)$ be the corresponding n-vector. Inequality (1) for this b-vector is easily seen to be of negative type and can be written as

$$\sum_{1 \leq i < j \leq n-1} x_{ij} \leq 1 + (n-3) \sum_{i=1}^{n-1} x_{in}. \tag{3}$$

Lemma 2. *Let S be any cut in K_n not containing vertex n and let $\delta(S)$ be its corresponding cut vector. Then the slack of $\delta(S)$ with respect to (3) is $(|S|-1)^2$.*

Let us label a cut S by a binary n-vector a where $a_i = 1$ if and only if $i \in S$. Under the conditions of the lemma we observe that the slack $(|S|-1)^2 = (a^T b - 1)^2$ since we have $a_n = 0$ and $b_1 = ... = b_{n-1} = 1$. Now consider consider any subset T of $\{1, 2, ..., n-1\}$ and set $b_i = 1$ for $i \in T$, $b_n = 3 - |T|$ and $b_i = 0$ otherwise. We form a 2^{n-1} by 2^{n-1} matrix M as follows. Let the rows and columns be indexed by subsets T and S of $\{1, 2, ..., n-1\}$, labelled by the n-vectors a and b as just described. A straight forward application of Lemma 2 shows that $M = M^*(n-1)$. Hence using the fact that the non-negative rank of a matrix is at least as large as that of any of its submatrices, we have that every extended formulation of $\mathrm{CUT}_n^{\square}$ has size $2^{\Omega(n)}$.

Recall the hypermetric polytope, defined in Section 2.1, is the intersection of all hypermetric inequalities. As remarked, nonnegative type inequalities are weaker than hypermetric inequalities and so valid for this polytope. In addition all cut vertices satisfy all hypermetric inequalities. Therefore $M = M^*(n-1)$ is also a submatrix of a slack matrix for the hypermetric polytope on n points. So this polytope also has extension complexity at least $2^{\Omega(n)}$.

Finally let us consider the polytope, which we denote P_n, defined by the inequalities used to define rows of the slack matrix M above. We will show that membership testing for P_n is co-NP-complete.

Theorem 8. *Let P_n be the polytope defined as above, and let $x \in \mathbb{R}^{n(n-1)/2}$. Then it is co-NP-complete to decide if $x \in P_n$.*

Proof. Clearly if $x \notin P_n$ then this can be witnessed by a violated inequality of type (3), so the problem is in co-NP.

To see the hardness we do a reduction from the clique problem: given graph $G = (V, E)$ on n vertices and integer k, does G have a clique of size at least k? Since a graph has a clique of size k if and only if its suspension has a clique of size $k + 1$ we can assume *wlog* that G is a suspension with vertex v_n connected to every other vertex.

Form a vector x as follows: $x_{ij} = 1/k$, if $j = n, x_{ij} = 2/k$, if $j \neq n$ and $ij \in E$ and $x_{ij} = -n^2$otherwise

Fix an integer t, $2 \leq t \leq n$ and consider a b-vector with $b_n = 3 - t$, and with $t - 1$ other values of $b_i = 1$. Without loss of generality we may assume these are lablelled $1, 2, ..., t-1$. Let T be the induced subgraph of G on these vertices. The corresponding non-negative type inequality is:

$$\sum_{1 \leq i < j \leq t-1} x_{ij} \leq 1 + (t-3) \sum_{i=1}^{t-1} x_{in}. \tag{4}$$

Suppose T is a complete subgraph. Then the left hand side minus the right hand side of (4) is $\frac{2(t-1)(t-2)}{2k} - (1 + \frac{(t-3)(t-1)}{k}) = \frac{t-k-1}{k}$. This will be positive if and only if $t \geq k + 1$, in which case x violates (4). On the other hand if T is not a complete subgraph then the left hand side of (4) is always negative and so the inequality is satisfied. Therefore x satisfies all inequalities defining rows of M if an only if G has no clique of size at least k. □

Cut polytope for minors of a graph. A graph H is a *minor* of a graph G if H can be obtained from G by contracting some edges, deleting some edges and isolated vertices, and relabeling. In the introduction we noted that if an n vertex graph G has no K_5-minor then $\mathrm{CUT}^{\square}(G)$ has $O(n^3)$ extension complexity. The following Lemma shows that the extension complexity of a graph G can be bounded from below in terms of its largest clique minor.

Lemma 3. *Let G be a graph and let H be obtained by deleting an edge of G, or deleting a vertex of G, or contracting an edge of G, Then,* $\mathrm{xc}(\mathrm{CUT}^{\square}(G)) \geqslant \mathrm{xc}(\mathrm{CUT}^{\square}(H))$.

Proof. Omitted. □

Therefore, we get the following theorem that can be proved by induction over a sequence of minor-producing steps.

Theorem 9. *Let G be a graph and H be a minor of G. Then,* $\mathrm{xc}(\mathrm{CUT}^{\square}(G)) \geqslant \mathrm{xc}(\mathrm{CUT}^{\square}(H))$.

Using the above theorem together with the result of [8] that the extension complexity of $\mathrm{CUT}^{\square}(K_n)$ is at least $2^{\Omega(n)}$ we get the following result.

Corollary 5. *The extension complexity of* $\mathrm{CUT}^{\square}(G)$ *for a graph G with a K_n minor is at least* $2^{\Omega(n)}$.

Using Theorem 9 and the fact that K_{n+1} is a minor of $K_{1,n,n}$ we can immediately prove that the Bell inequality polytopes mentioned in the introduction have exponential complexity.

Corollary 6. *The extension complexity of* $\mathrm{CUT}^{\square}(K_{1,n,n})$ *is at least* $2^{\Omega(n)}$.

Cut Polytope for K_6 ***minor-free graphs.*** Let $G = (V, E)$ be any graph with $V = \{1, \ldots, n\}$. Consider the suspension G' of G obtained by adding an extra vertex labelled 0 with edges to all vertices V.

Theorem 10. *Let* $G = (V, E)$ *be a graph and let* G' *be a suspension over* G. *Then* $\mathrm{STAB}(G)$ *is the projection of a face of* $\mathrm{CUT}^{\square}(G')$.

Proof. The polytope $\mathrm{STAB}(G)$ is defined over variables x_i corresponding to each of the vertex $i \in V$ whereas the polytope $\mathrm{CUT}^{\square}(G')$ is defined over the variables x_{ij} for $i, j \in \{0, \ldots, n\}$.

Any cut vertex C of $\mathrm{CUT}^{\square}(G')$ defines sets $S, \overline{S}$ such that $x_{ij} = 1$ if and only if $i \in S, j \in \overline{S}$. We may assume that $0 \in \overline{S}$ by interchanging S and $\overline{S}$ if necessary. For every edge $e = (k, l)$ in G consider an inequality $h_e := \{x_{0k} + x_{0l} - x_{kl} \geqslant 0\}$. It is clear that h_e is a valid inequality for $\mathrm{CUT}^{\square}(G')$ for all edges e in G. Furthermore, h_e is tight for a cut vector in G' if and only if either k, l do not lie in the same cut set or k, l both lie in the cut set containing 0. Therefore consider the face $F := \mathrm{CUT}^{\square}(G') \bigcap_{(i,j)\in E} \{x_{0i} + x_{0j} - x_{ij} = 0\}$.

Each vertex in F can be projected to a valid stable set in G by projecting onto the variables $x_{01}, x_{02}, \ldots, x_{0n}$. Furthermore, every stable set S in G can be extended to a cut vector for G' by taking the cut vector corresponding to $S, \overline{S} \cup \{0\}$. Therefore, $\mathrm{STAB}(G)$ is the projection of a face of $\mathrm{CUT}^{\square}(G')$. □

Using this theorem it is easy to show the existence of graphs with a linear number of edges that do not have K_6 as a minor and yet have a high extension complexity. In fact we get a slightly sharper result.

Theorem 11. *For every* $n \geqslant 2$ *there exists a graph* G *which is a suspension of a planar graph and for which* $\mathrm{xc}(\mathrm{CUT}^{\square}(G)) \geqslant 2^{\Omega(n^{1/4})}$.

Proof. Consider a planar graph $G = (V, E)$ with n vertices for which $\mathrm{xc}(\mathrm{STAB}(G)) \geqslant 2^{\Omega(n^{1/4})}$. Corollary 3 guarantees the existence of such a graph for every n. Then the suspension over G has $n+1$ vertices and a linear number of edges. The theorem then follows by applying Theorem 10 together with Propositions 1 and 2. □

The above theorem provides a sharp contrast for the complexity of the cut polytope for graphs in terms of their minors. As noted in the introduction, for any K_5 minor-free graph G with n vertices $\mathrm{CUT}^{\square}(G)$ has an extension of size $O(n^3)$ whereas the above result shows that there are K_6 minor free graphs whose cut polytope has superpolynomial extension complexity.

5 Concluding Remarks

We have a given a simple polyhedral procedure for proving lower bounds on the extension complexity of a polytope. Using this procedure and some standard NP-completeness reductions we were able to prove lower bounds on the extension complexity of various well known combinatorial polytopes. For the cut polytope in particular, we are able to draw a sharp line, in terms of minors, for when this complexity becomes super polynomial.

Nevertheless the procedure is not completely 'automatic' in the sense that any NP-completeness reduction of a certain type, say using gadgets, automatically gives a result on the extension complexity of related polytopes. This would seem to be a very promising line of future research.

References

[1] Avis, D., Imai, H., Ito, T.: Generating facets for the cut polytope of a graph by triangular elimination. Math. Program. 112(2), 303–325 (2008)
[2] Avis, D., Imai, H., Ito, T., Sasaki, Y.: Two-party bell inequalities derived from combinatorics via triangular elimination. J. Phys. A: Math. General 38(50), 10971–10987 (2005)
[3] Barahona, F.: The max-cut problem on graphs not contractible to K_5. Oper. Res. Lett. 2(3), 107–111 (1983)
[4] Bell, J.S.: On the Einstein-Podolsky-Rosen paradox. Physics 1(3), 195–290 (1964)
[5] Conforti, M., Cornuéjols, G., Zambelli, G.: Extended formulations in combinatorial optimization. 4OR 8, 1–48 (2010)
[6] Corman, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press (2009)
[7] Deza, M.M., Laurent, M.: Geometry of cuts and metrics. Algorithms and Combinatorics, vol. 15. Springer (1997)
[8] Fiorini, S., Massar, S., Pokutta, S., Tiwary, H.R., de Wolf, R.: Linear vs. semidefinite extended formulations: exponential separation and strong lower bounds. In: STOC, pp. 95–106 (2012)
[9] Garey, M.R., Johnson, D.S.: Computers and intractability; a guide to the theory of NP-completeness. W.H. Freeman (1979)
[10] Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. Theoret. Comput. Sci. 1, 237–267 (1976)
[11] Karp, R.M.: Reducibility among combinatorial problems. In: Miller, Tatcher (eds.) Complexity of Computer Computations. Plenum (1974)
[12] Rothvoß, T.: Some 0/1 polytopes need exponential size extended formulations. arXiv:1105.0036 (2011)
[13] Yannakakis, M.: Expressing combinatorial optimization problems by linear programs. Journal of Computer and System Sciences 43(3), 441–466 (1991)
[14] Ziegler, G.M.: Lectures on polytopes. Graduate Texts in Mathematics, vol. 152. Springer (1995)

Algorithms for Hub Label Optimization

Maxim Babenko[1], Andrew V. Goldberg[2],
Anupam Gupta[3], and Viswanath Nagarajan[4]

[1] Department of Mechanics and Mathematics, Moscow State Univerity, Yandex
[2] Microsoft Research Silicon Valley
[3] Carnegie Mellon University and Microsoft Research SVC
[4] IBM T.J. Watson Research Center

Abstract. Cohen et al. developed an $O(\log n)$-approximation algorithm for minimizing the total hub label size (ℓ_1 norm). We give $O(\log n)$-approximation algorithms for the problems of minimizing the maximum label (ℓ_∞ norm) and minimizing ℓ_p and ℓ_q norms simultaneously.

1 Introduction

Modern applications, such as computing driving directions and other location-based services, require very fast point-to-point shortest path algorithms. Although Dijkstra's algorithmsolves this problem in near-linear time [14] on directed and in linear time on undirected graphs [16], some applications require sublinear distance queries. This motivates preprocessing-based algorithms, which yield sublinear queries on some graph classes (e.g., [10,13]). In particular, Gavoille et al. [13] introduced *distance labeling* algorithms. These algorithms precompute *labels* for each vertex such that the distance between any two vertices s and t can be computed using only their labels.

A prominent case of this paradigm is *hub labeling (HL)*: the label of v consists of a collection of vertices (the *hubs* of v) with their distances from v. Hub labels satisfy the *cover property*: for any two vertices s and t, there exists a vertex w on the shortest s–t path that belongs to both the label of s and the label of t. Given this information, distance queries are easy to implement: for two vertices v and w, we compute the sums of the v-u and u-w distances over vertices u in the intersection of the labels of v and w, and return the minimum value found.

Cohen at al. [9] gave an $O(\log n)$-approximation algorithm for the smallest-size labeling, where n denotes the number of vertices and the size of the labeling is the sum of the number of hubs in the vertex labels. (This also minimizes the average label size.) The algorithm uses an elegant reduction to the set-cover problem [8]. At each step, the algorithm solves a maximum density subgraph problem, which can be done exactly using parametric flows [12] or by a faster approximation algorithm [15]. HL leads to the fastest implementation of the point-to-point shortest path queries in road networks [1], and works well on some other network types [2]. This motivates further theoretical study of HL.

In this paper we consider approximation algorithms for the optimization problem of producing small labels. Since minimizing the average label size may potentially lead to imbalanced solutions where the label of some vertices are relatively

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 69–80, 2013.

large, a natural objective is to minimize the maximum HL size, which determines the worst-case query time. We give a polynomial time algorithm that finds an $O(\log n)$ approximation of the maximum label size, where n is the size of the graph. Our algorithm is based on reducing the HL problem to a non-standard set covering problem, where the objective is to cover all the elements using sets in such a way that no element is covered too often. Our algorithm is combinatorial, and is based on exponential cost functions, as in [4] and many other contexts.

If we consider a vector whose components correspond to the number of vertex hubs, then the total label size is the $\ell_1$norm of this vector and the maximum label size is the ℓ_∞ norm. This brings a natural generalization of the above problems, that of optimizing the ℓ_p norm of this vector. Our second result is an $O(\log n)$-approximation algorithm for this more general problem. This is also a combinatorial algorithm, where we naturally use degree-p polynomials instead of exponential cost functions, as in [5].

In applications, there are multiple criteria that one would like to be good for—one wants to simultaneously minimize the total label size (i.e., the space needed to store the labels) and the maximum label size (i.e., the worst-case query time). This is a bi-criteria optimization problem, with the optimal solutions on the Pareto-optimal curve. E.g., suppose that there is a labeling with total label size T_1 and maximum label size T_∞ (e.g., a labeling on the Pareto curve). Our third result is a polynomial-time algorithm to find a labeling with the total label size $O(T_1 \log n)$ and the maximum label size $O(T_\infty \log n)$. In fact, our techniques easily extend to a more general result: a logarithmic approximation to the problem of maintaining k moments of the label sizes. Specifically, given any set $P = \{p_1, p_2, \ldots, p_k\}$, and values T_i such that there exists a labeling whose ℓ_{p_i} norm is at most T_i for each $p_i \in P$, we can find a labeling whose ℓ_{p_i} norm is at most $O(k \log n) \cdot T_i$ for all $p_i \in P$.

1.1 Related Work

There is much work on minimizing multiple norms of a vector. For some problems, one can find a single solution that is simultaneously good against the best solution for each ℓ_p norm individually. E.g., Azar et al. [7] considered the restricted-assignment machine scheduling problem, and gave a solution which 2-approximates the best solution for each ℓ_p norm; this was extended and improved by [6,3]. In some settings, it is possible to get better bounds for ℓ_p norm minimization for different values of p: e.g., [5] show p-competitive online algorithms for minimizing the ℓ_p norms of machine loads. However, the low-load set covering problem has a hardness of $\Omega(\log n)$ for all p, which means new techniques would be needed to get better approximations for HL.

2 Definitions and Notation

In the HL problem, we are given a graph $G = (V, E)$ with a distinguished shortest path P_{ij} between each pair of vertices $i, j \in \binom{V}{2}$. A *hub labeling* (HL) is

an assignment of labels $L_i \subseteq V$ for each vertex $i \in V$ such that for any $i, j \in V$, we have some vertex $u \in L_i \cap L_j$ that lies on the path P_{ij}. For the purposes of this paper, the graph can be directed or undirected, and the path P_{ij} can be an arbitrary path—we will not use the fact that it is a shortest i-j path. If we consider the vector $\mathbf{L} = (L_1, L_2, \ldots, L_n)$, then we are interested in finding labelings with small ℓ_p norms: $\|\mathbf{L}\|_p := (\sum_{i=1}^{n} L_i^p)^{1/p}$ and $\|\mathbf{L}\|_\infty := \max_{i \in V} L_i$. We assume $p \in [1, \log n]$, since $\ell_{\log n}$ approximates all higher ℓ_p norms to within constant factors.

We will reduce HL to a low-load set-covering problem (LSC), which is defined as follows. As in the usual set cover problem, we are given a set system $(U, \mathcal{F})$, where $\mathcal{F} = \{S_1, S_2, \ldots\}$ is a collection of subsets of the universe U with N elements. A sub-collection $\mathcal{C} \subseteq \mathcal{F}$ is a *set cover* if $\cup_{S \in \mathcal{C}} S = U$, every element of U is contained in some set in $\mathcal{C}$. The elements in U are either *relevant* (denoted by $R \subseteq U$) or *irrelevant* (those in $U \setminus R$): for any relevant element $e \in R$, let $A_{\mathcal{C}}(e) = \#\{S \in \mathcal{C} \mid e \in S\}$ be the *load* of element e under this set cover $\mathcal{C}$, the number of sets in $\mathcal{C}$ that contain e. (Imagine the irrelevant elements to always have load 0.) For any $p \in [1, \infty)$, the ℓ_p norm of the loads is $\|A_{\mathcal{C}}\|_p = (\sum_{e \in R} A_{\mathcal{C}}(e)^p)^{1/p}$; the ℓ_∞ norm is $\|A_{\mathcal{C}}\|_\infty = \max_{e \in R} A_{\mathcal{C}}(e)$. To reiterate: we have to cover all the elements, relevant or otherwise, but we count the load only for the relevant elements.

Our algorithms use approximate max-density oracles. A *max-density oracle* takes costs c_e for relevant elements $e \in R$ and a set $X \subseteq U$ of elements already covered, and outputs a set $S \in \mathcal{F}$ that minimizes $\frac{\sum_{e \in S \cap R} c_e}{|S \setminus X|}$. In Section 3.2 we show how to implement the oracle for LSC instances arising from HL.

3 Application to Shortest Path Labels

Our motivating application is HL ℓ_p norm optimization. We show a reduction from the label optimization problem to LSC and an implementation of an approximate max-density oracle for the corresponding LSC problem.

3.1 From Labels to Set Covers

We model an instance $\mathcal{I}$ of HL as the following instance $\mathcal{I}'$ of LSC.

- The elements are all $\{i, j\}$ pairs, and all vertices — i.e., $U := \binom{V}{2} \cup V$. The elements of V are relevant and the elements of $\binom{V}{2}$ are irrelevant.
- For each vertex $x \in V$, let Q_x be the set of pairs $\{i, j\} \in \binom{V}{2}$ such that $x \in P_{ij}$. For any set of pairs $Q \subseteq \binom{V}{2}$, let $V(Q)$ denote the vertices that lie in at least one of these pairs—i.e., $V(Q) = \cup_{\{i,j\} \in Q} \{i, j\}$. Now for each $x \in V$, for each $Q \subseteq Q_x$, add the set $Q \cup V(Q)$ to the collection $\mathcal{F}$. Note that this may give us exponentially many sets.
- The problem is to compute a set cover $\mathcal{C} \subseteq \mathcal{F}$ minimizing the ℓ_p norm $\|A_{\mathcal{C}}\|_p$ (which only involves the load on relevant elements, i.e. vertices).

Lemma 1. *Minimizing the ℓ_p norm of an instance $\mathcal{I}$ of HL is equivalent to minimizing the ℓ_p norm of the corresponding instance $\mathcal{I}'$ of LSC.*

Proof. Given a solution for $\mathcal{I}$, construct a solution for $\mathcal{I}'$ as follows. For each $x \in V$, take $S_x \subseteq \binom{V}{2}$ to be the pairs that x "covers" in this solution—i.e., pairs $\{i, j\}$ such that x lies in $L_i \cap L_j$ and also on the path P_{ij}. Picking the sets $S_x \cup V(S_x)$, one for each x, we get a cover $\mathcal{C}$ for $\mathcal{I}'$. If the label of i does not contain x, then $i \notin V(S_x)$. Thus only the vertices x in the label of i can contribute to $A_{\mathcal{C}}(i)$, and therefore $A_{\mathcal{C}}(i) \leq |L_i|$.

Given a solution $\mathcal{C}$ to $\mathcal{I}'$, we construct a label L for $\mathcal{I}$ as follows. If $\mathcal{I}'$ contains a set $(Q \cup V(Q))$ for some $Q \subseteq Q_x$, then add x to L_i for all $i \in V(Q)$. Since all pairs in U are covered by $\mathcal{C}$, for each pair $\{i, j\}$, the labels L_i and L_j intersect at some vertex of P_{ij}. For every vertex i, we add x to L_i only if there is a set $(Q \cup V(Q)) \in \mathcal{C}$ with $Q \subseteq Q_x$ that covers i (i.e. $i \in V(Q)$). Thus $|L_i| \leq A_{\mathcal{C}}(i)$. □

3.2 Max-Density Oracle

In this section we show how to construct approximate max-density oracles for the LSC instances $\mathcal{I}'$ obtained via a reduction from HL. Recall that there are costs $c_i \geq 0$ for all $i \in V$ (relevant elements). Let us divide U into U_P (the pairs in V), and U_V (the vertices in V); recall that U_V are the relevant elements. For any set $S \subseteq U$, we now use $S_P := S \cap U_P$ and $S_V := S \cap U_V$.

Lemma 2. *Solving* $\min_{S \in \mathcal{F}} \frac{\sum_{v \in S_V} c_v}{|S_P \setminus X_P|}$ *gives a 3-approximate max-density oracle.*

Proof. For any $T \subseteq U$, we have $T = T_P \cup T_V$. By the structure of our sets, if some pair $\{i, j\} \in T_P$ then it must be the case that $\{i, j\} \subseteq T_V$. Hence, $V(T_P) \subseteq T_V$. Moreover, $|T_P \cup V(T_P)| \leq 3|T_P|$, since each pair in T_P can contribute at most both its endpoints to $V(T_P)$. Combining these facts, we get $|T_P| \leq |T| \leq 3|T_P|$. This implies the lemma. □

Recall that the Cohen et al. [9] algorithm approximates the ℓ_1 norm of the labels. Their algorithm uses a subroutine for the *maximum density subgraph* problem. We use a subroutine for the *weighted* variant of the problem: given a non-negative cost function $c : V \to \mathcal{R}_+$, the *density* of a graph H is $\mu(H) = \frac{|E(H)|}{c(V(H))}$. Note that $\mu(H)$ is undefined if $c(V(H)) = 0$. In this case if $|E(H)| = 0$ we define $\mu(H) = 0$, and otherwise $\mu(H) = \infty$. The *maximum density subgraph* problem is to find a vertex-induced subgraph of maximum density. This can be solved exactly using network flow [12]. In the full version, we describe a generalization of the faster 2-approximation algorithm [15] for the unit-weight maximum density subgraph to the weighted case.

Fix $v \in V$ and a set $X \subseteq U$ of covered elements. We define the v-center graph G_v as follows. The vertex set of G_v is V. Two vertices i, j are connected in G_v iff the pair $\{i, j\} \notin X$ and $P_{i,j} \ni v$, i.e., if v covers the pair.

Lemma 3. *We can reduce* $\min_{S \in \mathcal{F}} \frac{\sum_{v \in S_V} c_v}{|S_P \setminus X_P|}$ *to n weighted maximum density subgraph problems.*

Proof. Let S' be some set of vertices. Suppose that we add a vertex v to the labels of all vertices in S'. Then the edges in the subgraph of G_v induced by S' correspond to previously uncovered $\{i, j\}$ pairs that become covered. So the maximum density subgraph of G_v yields the set that minimizes the desired expression over the sets that correspond to v. The result follows by minimizing over all v (i.e., solving n weighted maximum density subgraph problems). □

A ρ-approximate solution to the weighted maximum subgraph problem gives a ρ-approximate minimum. Note that the results of this section extend to directed graphs. In this case the center graphs will become bipartite, as in [9].

4 The ℓ_∞ Case: Minimizing the Maximum Load

In this section we investigate the problem of finding LSC $\mathcal{C}$ that approximately minimizes the maximum load of any relevant element in U. Recall that we have to cover the irrelevant elements (those in $U \setminus R$), even though we do not care about the load on them. Suppose that we know the optimal load $u = \|A_{\mathcal{C}^*}\|_\infty$; we can enumerate over all the possible values of u. We show a combinatorial greedy-like algorithm that achieves $\rho = O(\log N)$, where $N = |U|$.

Since the family $\mathcal{F}$ may consist of an exponential number of sets, and it may not be possible to look over all sets to find the best one, we assume that we have an α-approximate *max-density oracle* through which we access the set system.

Theorem 1. *There exists a combinatorial algorithm that makes at most n calls to an α-approximate max-density oracle, and finds a set cover $\mathcal{C}$ with element loads $A_{\mathcal{C}}(e) \leq O(\alpha \log N) \cdot \|A_{\mathcal{C}^*}\|_\infty$.*

The algorithm is a multiplicative-weights algorithm. It proceeds in *rounds*, where one set is added in each round. Let $\varepsilon = 1/(8\alpha)$. Let $A_t(e)$ be the number of times a relevant element $e \in R$ has been covered at the beginning of round t; at the very beginning of the process we have $A_1(e) = 0$ for all $e \in R$. Define the round-t cost of elements $e \in R$ as $c_t(e) := (1+\varepsilon)^{A_t(e)/u} \cdot \left((1+\varepsilon)^{1/u} - 1\right)$; so the round-$t$ cost of set S is $c_t(S) := \sum_{e \in S \cap R} (1+\varepsilon)^{A_t(e)/u} \cdot \left((1+\varepsilon)^{1/u} - 1\right)$.

Note the sum is only over the relevant elements in S. The algorithm is simple: consider the beginning of round t, when $t-1$ sets have already been picked, and let X_t be the elements already covered at this time. (Hence $X_1 = \emptyset$.) If all elements have not yet been covered (i.e., if $X_t \neq U$), use the α-approximate max-density oracle with costs $c_t(S)$ and the set X_t to obtain the next set.

For the analysis, define the potential at the beginning of round t to be $\Phi(t) := \sum_{e \in R} (1+\varepsilon)^{A_t(e)/u}$. The potential at the beginning of round 1 is $\Phi(1) = N$.

Lemma 4. *If we pick set S in round t, then $\Phi(t+1) - \Phi(t) = c_t(S)$.*

Proof. By the definition of the potential,

$$\Phi(t+1) - \Phi(t) = \sum_{e \in R} (1+\varepsilon)^{A_{t+1}(e)/u} - \sum_{e \in R} (1+\varepsilon)^{A_t(e)/u}$$

$$= \sum_{e \in S \cap R} (1+\varepsilon)^{(A_t(e)+1)/u} - (1+\varepsilon)^{A_t(e)/u} = \sum_{e \in S \cap R} (1+\varepsilon)^{A_t(e)/u} \left((1+\varepsilon)^{1/u} - 1 \right)$$

which is $c_t(S)$ by the definition of the round-t cost. □

Lemma 5. *At any round t, there exists a set S with cost-to-coverage ratio $\frac{c_t(S)}{|S \setminus X_t|} \leq \frac{\Phi(t)}{|U \setminus X_t|} \cdot 2\varepsilon$.*

Proof. The round-t cost of all the sets in the set cover $\mathcal{C}^*$ is

$$\sum_{S \in \mathcal{C}^*} c_t(S) = \sum_{S \in \mathcal{C}^*} \sum_{e \in S \cap R} (1+\varepsilon)^{A_t(e)/u} \cdot \left((1+\varepsilon)^{1/u} - 1 \right) \tag{1}$$

$$= \sum_{e \in R} \left[(1+\varepsilon)^{A_t(e)/u} \cdot \left((1+\varepsilon)^{1/u} - 1 \right) \cdot \sum_{S \in \mathcal{C}^* : e \in S} 1 \right]$$

$$\leq \sum_{e \in R} (1+\varepsilon)^{A_t(e)/u} \cdot 2\varepsilon/u \cdot u \quad \leq \quad \Phi(t) \cdot 2\varepsilon. \tag{2}$$

(The equality in (1) uses the definition of $c_t(S)$, and (2) used $u \geq 1$ and $\varepsilon \leq 1/4$ and hence $(1+\varepsilon)^{1/u} \leq 1 + (\varepsilon/u)(1 + \varepsilon + \varepsilon^2 + \cdots) \leq 1 + 2\varepsilon/u$.) Since $|U \setminus X_t|$ universe elements are not yet covered, and we could have chosen all the sets in $\mathcal{C}^*$ to cover these remaining elements at cost $\sum_{S \in \mathcal{C}^*} c_t(S)$, there exists some set whose cost-to-coverage-ratio is at most $\Phi(t) \cdot 2\varepsilon / (|U \setminus X_t|)$. □

Let us partition the rounds into phases: phase i begins in round t if the number of uncovered elements is at most $N/2^i$ for the first time at the beginning of round t. Hence phase 0 begins with round 1 (when the number of uncovered elements is $N/2^0 = N$) and ends at the point we have covered half the elements, etc. Note that some phases contain no rounds at all.

Lemma 6. *If rounds a and b are the first and last rounds of some phase i, then $\Phi(b+1) \leq 2 \cdot \Phi(a)$.*

Proof. Consider the beginning of some round $t \in \{a, a+1, \ldots, b\}$ in phase i. By Lemma 5 there exists a set with cost-to-coverage-ratio is at most $\Phi(t) \cdot 2\varepsilon / (|U \setminus X_t|)$. Moreover, the α-approximate max-density oracle finds a set whose cost-to-coverage ratio at most α times as much; i.e., at most $2\alpha\varepsilon \frac{\Phi(t)}{|U \setminus X_t|} \leq \frac{1}{4} \frac{\Phi(t)}{|U \setminus X_t|}$, using the definition of $\varepsilon = 1/(8\alpha)$.

Since the potential $\Phi(t)$ is non-decreasing, and $|U \setminus X_t| \geq N/2^{i+1}$ in this phase, this last expression is at most $\frac{1}{4} \frac{\Phi(b+1)}{N/2^{i+1}}$. Moreover, we cover at most $N/2^i$ elements in this phase, so the total cost incurred is at most $\frac{1}{2}\Phi(b+1)$. By Lemma 4, the total cost incurred in the phase equals the change in potential, so $\Phi(b+1) - \Phi(a) \leq (1/2) \cdot \Phi(b+1)$. This proves the lemma. □

Proof of Theorem 1: We claim the potential at the end of the algorithm is at most N^2. Indeed, using Lemma 6, the potential at most doubles in each phase, whereas the number of yet-covered elements at least halves. Hence, at the end

of phase $\log_2 N$, there are strictly less than $N/2^{\log_2 N} = 1$ elements (i.e., zero elements) remaining; the potential is at most $2^{\log_2 N} \cdot \Phi(1) = N^2$.

Suppose the last round in which we pick a set is $f-1$. Then for the final potential to be at most N^2, it must be the case that for each $e \in U$, $(1+\varepsilon)^{A_f(e)/u} \leq N^2$. This means that $A_f(e) \leq u \log_{1+\varepsilon}(N^2) = O(\log N)u/\varepsilon = O(\alpha \log N)u$. □

5 Simultaneous ℓ_1 and ℓ_∞ Norm Approximation

We can extend the results of Section 4 to find a set cover that simultaneously has small maximum load and small average load. In this case, suppose we are given non-negative values T and u such that there exists a set cover $\mathcal{C}^*$ with $\sum_{e \in R} A_{\mathcal{C}^*}(e) \leq T$, and also $A_{\mathcal{C}^*}(e) \leq u$ for all relevant elements $e \in R$. We want to find a cover $\mathcal{C}$ such that $\|A_{\mathcal{C}}\|_1 \leq T \cdot O(\alpha \log N)$, and $\|A_{\mathcal{C}}\|_\infty \leq u \cdot O(\alpha \log N)$.

For the algorithm, we use definitions of $c_t(e)$, $c_t(S)$, $\Phi(t)$, etc., from Section 4, but redefine ε to be $\frac{1}{24\alpha}$. We define the d-cost as $d_e = 1$ for $e \in R$; so $d(S) := |S \cap R|$ for any set S. The algorithm changes as follows: in round t, we now use the α-approximate density oracle to pick a set S (approximately) minimizing the "combined" ratio

$$\frac{c_t(S) + \varepsilon(\Phi(t)/T) \cdot d(S)}{|S \setminus X_t|}. \tag{3}$$

For the analysis, consider the beginning of round t, and call a set S *t-light* if $\frac{d(S)}{|S \setminus X_t|} \leq \frac{2T}{|U \setminus X_t|}$, and *$t$-heavy* otherwise. Let $\mathcal{C}_h^*$ denote the t-heavy sets in $\mathcal{C}^*$, and $\mathcal{C}_l^* := \mathcal{C}^* \setminus \mathcal{C}_h^*$ the light sets.

Lemma 7. *The number of elements from $U \setminus X_t$ covered by t-heavy sets in $\mathcal{C}^*$ is at most $\frac{1}{2}|U \setminus X_t|$.*

Proof. The fraction of the remaining elements that any t-heavy set S covers is $\frac{|S \setminus X_t|}{|U \setminus X_t|} \leq \frac{d(S)}{2T}$. Hence, the total fraction of remaining elements that t-heavy sets in $\mathcal{C}^*$ cover is $\sum_{S \in \mathcal{C}_h^*} \frac{|S \setminus X_t|}{|U \setminus X_t|} \leq \sum_{S \in \mathcal{C}_h^*} \frac{d(S)}{2T} \leq 1/2$. The last inequality is because $\sum_{S \in \mathcal{C}_h^*} d(S) \leq \sum_{S \in \mathcal{C}^*} |S \cap R| \leq T$. □

We can now modify Lemma 5 to say the following:

Lemma 8. *At any round t, there exists a t-light set S with cost-to-coverage ratio $\frac{c_t(S)}{|S \setminus X_t|} \leq \frac{\Phi(t)}{|U \setminus X_t|} \cdot 4\varepsilon$.*

Proof. Let $z := |U \setminus X_t|$ denote the number of universe elements not yet covered. Choosing all t-light sets in $\mathcal{C}^*$ at cost $\sum_{S \in \mathcal{C}_l^*} c_t(S)$, we would cover at least $z/2$ elements (by Lemma 7), hence there exists some set whose cost-to-coverage-ratio is at most $\frac{2}{z} \cdot \sum_{S \in \mathcal{C}_l^*} c_t(S) \leq \frac{2}{z} \cdot \sum_{S \in \mathcal{C}^*} c_t(S) \leq \frac{2}{z} \cdot \Phi(t) \cdot 2\varepsilon$. using the calculations as in Lemma 5. □

By Lemma 8, we infer there exists a set S whose combined cost-per-coverage is

$$\frac{c_t(S) + \varepsilon(\Phi(t)/T) \cdot d(S)}{|S \setminus X_t|} \leq \frac{\Phi(t) \cdot 4\varepsilon + \varepsilon(\Phi(t)/T) \cdot 2T}{|U \setminus X_t|} = \frac{\Phi(t) \cdot 6\varepsilon}{|U \setminus X_t|}.$$

Our α-approximate density oracle finds a set S_t with combined-cost-per-coverage at most $\alpha \cdot 6\varepsilon \cdot \Phi(t)/|U \setminus X_t| = \frac{1}{4}\Phi(t)/|U \setminus X_t|$. Since the combined cost is a sum of non-negative quantities, we get

$$\frac{c_t(S_t)}{|S_t \setminus X_t|} \leq \frac{1}{4}\frac{\Phi(t)}{|U \setminus X_t|} \qquad \text{and} \qquad \frac{d(S_t)}{|S_t \setminus X_t|} \leq \frac{1}{4\varepsilon}\frac{T}{|U \setminus X_t|}. \tag{4}$$

The bound on $c_t(S_t)/|S_t \setminus X_t|$ can be used in the same fashion as in Section 4 to show that the potential function at most doubles during a phase, and there are at most $\log_2 N$ phases, so the maximum load is at most $O(\alpha \log N) \cdot u$. Moreover, the d-cost incurred in any phase i is at most $(N/2^i) \cdot \frac{1}{4\varepsilon} \cdot \frac{T}{N/2^{i+1}} = \frac{1}{2\varepsilon} \cdot T$. Summing over all $\log_2 N$ phases, and using $\varepsilon = 1/(24\alpha)$, we get the total d-cost is $\sum_{S \in \mathcal{C}} |S \cap R| = \|A_{\mathcal{C}}\|_1 \leq (12\alpha \log_2 N) \cdot T$.

Application to Shortest-Path Labelings. To use this result for shortest-path labelings, we would again use the same reduction and the same max-distance oracle as in Section 3. We can try all the polynomially many guesses for u and T.

6 Minimizing ℓ_p Norms

We now turn to approximating ℓ_p norms of the element loads for $p \in [1, \log N]$. Specifically, we want a set cover $\mathcal{C}$ with the ℓ_p-norm of the loads $\|A_{\mathcal{C}}\|_p$ only logarithmically larger than $\|A_{\mathcal{C}^*}\|_p$ for any other set cover $\mathcal{C}^*$. (Recall that the vectors $A_{\mathcal{C}}$ and $A_{\mathcal{C}^*}$ are only defined on the relevant elements.) The round-t costs are now $c_{p,t}(e) := (A_t(e) + 1)^p - A_t(e)^p$ for $e \in R$; so

$$c_{p,t}(S) \quad := \quad \sum_{e \in S \cap R} ((A_t(e) + 1)^p - A_t(e)^p),$$

(the sum being only over the relevant items), and the algorithm picks a set that (approximately) minimizes $\frac{c_{p,t}(S)}{|S \setminus X_t|}$. Again, X_t is the set of elements covered prior to round t. $A_t(e)$ is the load of element e at the beginning of round t. To make the analysis easier, we set $A_1(e) = p$ for all relevant elements $e \in R$. The potential function is now a polynomial: $\Phi_p(t) := \sum_{e \in R} A_t(e)^p$. It immediately follows that if we pick set S_t in round t, $\Phi_p(t+1) - \Phi_p(t) = c_{p,t}(S_t)$. Initially $\Phi_p(1) = |R| \cdot p^p$; we will have to deal with this issue.

Lemma 9. *For $b := (e - 1)$ and any t, $\sum_{S \in \mathcal{C}^*} c_{p,t}(S) \leq b \cdot p \cdot \Phi_p(t)^{(p-1)/p} \cdot \|A_{\mathcal{C}^*}\|_p$.*

Proof. By the definition of $c_{p,t}(\cdot)$, we know that

$$\sum_{S \in \mathcal{C}^*} c_{p,t}(S) = \sum_{S \in \mathcal{C}^*} \sum_{e \in S \cap R} ((A_t(e) + 1)^p - A_t(e)^p) \leq \sum_{S \in \mathcal{C}^*} \sum_{e \in S \cap R} (\mathrm{e} - 1)\, p\, A_t(e)^{p-1},$$

(which follows from Fact 1 below and the observation that $A_t(e) \geq A_1(e) \geq p$)

$$\begin{aligned}
&= (\mathrm{e}-1)\, p \sum_{e \in R} \left[A_t(e)^{p-1} \sum_{S \in \mathcal{C}^* : e \in S} 1 \right] = (\mathrm{e}-1)\, p \sum_{e \in R} A_t(e)^{p-1} A_{\mathcal{C}^*}(e) \\
&\leq (\mathrm{e}-1)\, p \, \|A_t\|_p^{p-1} \, \|A_{\mathcal{C}^*}\|_p \qquad \text{(by Hölder's inequality)} \\
&\leq (\mathrm{e}-1)\, p \cdot \Phi(t)^{(p-1)/p} \cdot \|A_{\mathcal{C}^*}\|_p
\end{aligned}$$

Note that we used the fact that we initialized $A_1(e) \geq p$. □

Fact 1 *For any real $r \geq 1$ and $x \geq r$, $(x+1)^r - x^r \leq (e-1)\, r\, x^{r-1}$.*

Proof. Observe that $(x+1)^r - x^r = x^r((1+x^{-1})^r - 1)$, thus

$$x^r \sum_{j=1}^{\infty} \binom{r}{j} x^{-j} \leq x^r \sum_{j=1}^{\infty} \frac{r^j}{j!} x^{-j} \leq x^r \frac{r}{x} \sum_{j=1}^{\infty} \frac{(r/x)^{j-1}}{j!} \leq r\, x^{r-1} \sum_{j=1}^{\infty} \frac{1}{j!} \leq (\mathrm{e}-1)\, r\, x^{r-1},$$

where we used the inequality $x \geq r$. □

Corollary 1. *At any time t, there exists a set S with $\frac{c_{p,t}(S)}{|S \setminus X_t|} \leq \frac{2b \cdot p \cdot \Phi_p(t)^{(p-1)/p} \cdot \|A_{\mathcal{C}^*}\|_p}{|U \setminus X_t|}$. Hence the max-density algorithm picks a set with cost-to-coverage ratio at most α times that.*

Proof. If there exists a set in $\mathcal{C}^*$ that satisfies the above property, we are done. Hence imagine that no set in $\mathcal{C}^*$ satisfies it. Then

$$\sum_{S \in \mathcal{C}^*} \frac{|S \setminus X_t|}{|U \setminus X_t|} \leq \sum_{S \in \mathcal{C}^*} \frac{c_{p,t}(S)}{2b \cdot p \cdot \Phi_p(t)^{(p-1)/p} \cdot \|A_{\mathcal{C}^*}\|_p} \leq \frac{1}{2},$$

where the last inequality is from Lemma 9. But since all elements in U need to be covered by $\mathcal{C}^*$, this quantity is at least 1, a contradiction. □

Lemma 10. *For $\mathcal{C}$ produced by the algorithm, $\|A_{\mathcal{C}}\|_p \leq O(\alpha \log N) \cdot \|A_{\mathcal{C}^*}\|_p$.*

Proof. We define phase $i \in \{0, 1, \ldots, \log_2 N\}$ to consist of rounds t where $|U \setminus X_t| \in (\frac{N}{2^{i+1}}, \frac{N}{2^i}]$. Let $\beta := 2\,\alpha\, b\, p$. Let t^* denote the last round where $\Phi_p(t^*)^{1/p} \leq 4\beta \|A_{\mathcal{C}^*}\|_p$, and let i^* denote the phase containing t^*. Note the starting potential was $\Phi_p(1) = |R| \cdot p^p \leq |R|\, \beta^p \leq \beta^p \|A_{\mathcal{C}^*}\|_p^p$; hence $t^* \geq 1$.

Consider any phase $i \geq i^*$, and let I (resp., F) denote the values of Φ_p at the start (resp., end) of phase i. By our choice of t^* and hence of i^*, it follows that $F^{1/p} \geq \Phi_p(t^*+1)^{1/p} > 4\beta \|A_{\mathcal{C}^*}\|_p$. Moreover, the cost-to-coverage ratio of sets picked in phase i is at most $\frac{\beta \cdot F^{(p-1)/p} \cdot \|A_{\mathcal{C}^*}\|_p}{N/2^{i+1}}$ (using Corollary 1), and at most $N/2^i$ elements are covered in this phase. Consequently the total cost incurred during phase i is at most $2\beta \cdot F^{(p-1)/p} \cdot \|A_{\mathcal{C}^*}\|_p$; moreover, this total cost equals the increase in potential, $F - I$. We can rewrite the resulting inequality as:

$$I^{\frac{1}{p}} \geq F^{\frac{1}{p}} \left(1 - \frac{2\beta \|A_{\mathcal{C}^*}\|_p}{F^{\frac{1}{p}}}\right)^{\frac{1}{p}} \;\geq\; F^{\frac{1}{p}} \cdot e^{\frac{-4\beta \|A_{\mathcal{C}^*}\|_p}{p \cdot F^{\frac{1}{p}}}} \;\geq\; F^{\frac{1}{p}} \cdot \left(1 - \frac{4\beta \|A_{\mathcal{C}^*}\|_p}{p \cdot F^{\frac{1}{p}}}\right).$$

The second inequality above uses $1 - x \geq e^{-2x}$ for $0 \leq x \leq 1/2$; the final inequality is by $e^y \geq 1 + y$ for all y. This gives us that $F^{1/p} - I^{1/p} \leq \frac{4\beta}{p} \cdot \|A_{\mathcal{C}^*}\|_p$ for any phase $i \geq i^*$. Now summing over all such phases $i \geq i^*$ (there are at most $\log_2 N$ of them), we obtain $\Phi_p(\mathsf{final})^{1/p} - \Phi_p(t^*)^{1/p} \leq \frac{4\beta}{p} \cdot \log_2 N \cdot \|A_{\mathcal{C}^*}\|_p$. This uses the fact that round t^* lies in phase i^* and $\Phi_p(\cdot)$ is monotone non-decreasing. Finally, using $\Phi_p(t^*) \leq 4\beta \|A_{\mathcal{C}^*}\|_p$ and that $\beta = 2\alpha\, b\, p$, we have $\Phi_p(\mathsf{final})^{1/p} \leq (\frac{4\beta}{p} \log_2 N + 4\beta) \cdot \|A_{\mathcal{C}^*}\|_p = (4\alpha\, b\, (\log_2 N + p)) \cdot \|A_{\mathcal{C}^*}\|_p$. Since $p \leq \log N$, this completes the proof. □

Note that minimizing ℓ_∞ is within constant factors of minimizing $\ell_{\log N}$, so the result subsumes that of Section 4. Moreover, this algorithm does not require us to enumerate over guesses of the optimum load u.

7 Multiple Norms Simultaneously

The approach of the previous section naturally extends to give solutions that are good with respect to multiple ℓ_p norms; we now show how to handle two norms. Specifically, given $p, q \in [1, \log N]$, we want to find a cover $\mathcal{C}$ with $\|A_{\mathcal{C}}\|_p \leq O(\alpha \log N)\|A_{\mathcal{C}^*}\|_p$ and $\|A_{\mathcal{C}}\|_q \leq O(\alpha \log N)\|A_{\mathcal{C}^*}\|_q$, where $\mathcal{C}^*$ is some intended "optimal" cover. We assume we know values P, Q such that $\|A_{\mathcal{C}^*}\|_p \approx P$ and $\|A_{\mathcal{C}^*}\|_q \approx Q$.

We define the round-t costs $c_{p,t}$ and $c_{q,t}$ as in (6), and the potentials $\Phi_p(t)$ and $\Phi_q(t)$ as in (6). We initialize the loads $A_1(e)$ to $\max\{p, q\} \leq \log N$, and run the algorithm where picking the set S minimizing the "combined" ratio:

$$\frac{1}{|S \setminus X_t|} \cdot \left(\frac{c_{p,t}(S)}{p \cdot \Phi_p(t)^{(p-1)/p} \cdot P} + \frac{c_{q,t}(S)}{q \cdot \Phi_q(t)^{(q-1)/q} \cdot Q} \right) \tag{5}$$

Lemma 11. *At any time t, there exists a set S such that for both $r \in \{p, q\}$,*

$$\frac{c_{r,t}(S)}{|S \setminus X_t|} \leq \frac{3b \cdot r \cdot \Phi_r(t)^{(r-1)/r} \cdot \|A_{\mathcal{C}^*}\|_r}{|U \setminus X_t|}.$$

Proof. Suppose each set $S \in \mathcal{C}^*$ fails the inequality corresponding to either p or q or both, and say $\mathcal{C}^*_p, \mathcal{C}^*_q \subseteq \mathcal{C}^*$ denote the corresponding sets. Then

$$\sum_{r \in \{p,q\}} \sum_{S \in \mathcal{C}^*_r} \frac{|S \setminus X_t|}{|U \setminus X_t|} \leq \sum_{r \in \{p,q\}} \sum_{S \in \mathcal{C}^*_r} \frac{c_{r,t}(S)}{3b \cdot r \cdot \Phi_r(t)^{(r-1)/r} \cdot \|A_{\mathcal{C}^*}\|_r} \leq \frac{1}{3} + \frac{1}{3},$$

where the last inequality uses Lemma 9. But since all elements in U are covered by $\mathcal{C}^*$, this should be at least 1, a contradiction. □

Hence at each step t, there exists some set where the combined ratio objective function (5) for the algorithm has value at most $\frac{6b}{|U \setminus X_t|}$. Thus the algorithm will pick set S_t with objective function value at most α times greater, which guarantees a set S_t with $\frac{c_{r,t}(S_t)}{|S \setminus X_t|} \leq \alpha \cdot \frac{6b \cdot r \cdot \Phi_r(t)^{(r-1)/r} \cdot \|A_{\mathcal{C}^*}\|_r}{|U \setminus X_t|}$ for both $r \in \{p, q\}$.

Finally, the analysis from Lemma 10 carries over virtually unchanged for both p, q, the only difference being the definition $\beta := 6b\alpha r$ instead of 6.

The above algorithm extends to finding LSC that is within an $O(\alpha k \log N)$ factor of k different targets with respect to k different ℓ_p norms $p_1, p_2, \ldots, p_k$.

7.1 Non-existence of Simultaneous Optimality

In this section we construct a family of graphs for which no labeling can be simultaneously near-optimal for the total label size T^* and the maximum label size A^*. For any labeling with the total size T and the maximum size M, either T is polynomially bigger than T^* or M is polynomially bigger than M^*.

For a parameter k, the (undirected) graph has three sets of vertices, $A = \{a_1, a_2, \ldots, a_k\}$, $B = \{b_1, b_2, \ldots b_{k^2}\}$, and $C = \{c_{ij} \mid i \in [k^2], j \in [k]\}$, of size k, k^2, and k^3, respectively. Every vertex in A is connected to all vertices in B. Vertices in C are partitioned into k^2 groups of size k each. For every i, the partition $C_i = \{c_{ij} \mid j \in [k]\}$ corresponds to the vertex $b_i \in B$. Each vertex in B is connected to all k vertices in its group in C. There are no other edges in the graph. All edges have length 1, except for the edges from a_1 to B, which are of length $1 - \varepsilon$. The total number of vertices of the graph is $n = \Theta(k^3)$.

Observe that the shortest paths in the graph are as follows:

- For vertices $a, a' \in A$, and every $b \in B$, the path a, b, a' is a shortest a-a' path. For any vertex $a \in A, b \in B$, the edge (a, b) is a shortest path. For any $a \in A$ and $c \in C_i$, the path a, b_i, c is the unique shortest path.
- For vertices $b, b' \in B$, the path b, a_1, b' is the unique shortest path. For any i, the unique shortest path between vertex $b_i \in B$ and any $c \in C_i$ is the edge (b_i, c); for any $c \in C_{i'} \neq C_i$, the path $b_i, a_1, b_{i'}, c$ is the unique shortest path.
- For vertices c, c' in the same group C_i, the path c, b_i, c' is the unique shortest path. For $c \in C_i$ and $c' \in C_j \neq C_i$, the path c, b_i, a_1, b_j, c' is the unique shortest path.

An $O(k^3)$-size labeling is as follows. Each vertex is in its own label. In addition, every vertex in A has all vertices in B in its label. Every vertex in B contains a_1 in its label. Every vertex $c \in C$ has a_1 in its label; moreover, if c belongs to group C_i it has the corresponding vertex $b_i \in B$ its label. The total label size is $k \cdot (k^2 + 1) + 2 \cdot k^2 + 3 \cdot k^3 = O(k^3)$. In this labeling, the vertices in A have labels of size $k^2 + 1$.

There is a different labeling with the maximum label size $O(k)$. Each vertex is in its own label. In addition, every vertex in the graph has all vertices of A in its label. Moreover, if c belongs to group C_i it has the corresponding vertex $b_i \in B$ its label. The total size of this labeling is $\Omega(k^4)$.

Now consider a labeling L with the total size T and the maximum size M. For a vertex $a \in A$, consider shortest paths to all vertices in C. The number of vertices $c \in C$ for which an a–c shortest path contains a vertex of $L(a)$ different from a is at most kM. Therefore labels of $k^3 - kM$ vertices in C must contain a. Thus $T \geq k(k^3 - kM)$, or $T + k^2 M \geq k^4$. Hence, if $T = o(k^4)$, then $M = \Omega(k^2)$,

and if $M = o(k^2)$, then $T = \Omega(k^4)$. Therefore T or M is a factor $\Omega(n^{1/3})$ away from the corresponding optima.

Acknowledgments. We thank Amos Fiat and Haim Kaplan for interesting discussions. A. Gupta thanks Microsoft Research Silicon Valley, IBM T.J. Watson Research Center, and the IEOR Department at Columbia University for their generous hospitality.

References

1. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 230–241. Springer, Heidelberg (2011)
2. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: Hierarchical Hub Labelings for Shortest Paths. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 24–35. Springer, Heidelberg (2012)
3. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: A unified approach to scheduling on unrelated parallel machines. JACM 56(5), 28–31 (2009)
4. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM 44(3), 486–504 (1997)
5. Awerbuch, B., Azar, Y., Grove, E.F., Kao, M.-Y., Krishnan, P., Vitter, J.S.: Load balancing in the lp norm. In: FOCS, pp. 383–391 (1995)
6. Azar, Y., Epstein, A.: Convex programming for scheduling unrelated parallel machines. In: STOC 2005, pp. 331–337 (2005)
7. Azar, Y., Epstein, L., Richter, Y., Woeginger, G.J.: All-norm approximation algorithms. J. Algorithms 52(2), 120–133 (2004)
8. Chvátal, V.: A Greedy Heuristic for the Set-Covering Problem. Mathematics of Operations Research 4(3), 233–235 (1979)
9. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and Distance Queries via 2-hop Labels. SIAM J. Comput. 32 (2003)
10. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering Route Planning Algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009)
11. Feige, U.: A threshold of $\ln n$ for approximating set cover. J. ACM 45(4), 634–652 (1998)
12. Gallo, G., Grigoriadis, M.D., Tarjan, R.E.: A Fast Parametric Maximum Flow Algorithm and Applications. SIAM J. Comput. 18, 30–55 (1989)
13. Gavoille, C., Peleg, D., Pérennes, S., Raz, R.: Distance Labeling in Graphs. Journal of Algorithms 53, 85–112 (2004)
14. Goldberg, A.V.: A Practical Shortest Path Algorithm with Linear Expected Time. SIAM Journal on Computing 37, 1637–1655 (2008)
15. Kortsarz, G., Peleg, D.: Generating Sparse 2-Spanners. J.Alg. 17, 222–236 (1994)
16. Thorup, M.: Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time. JACM 46, 362–394 (1999)

Improved Approximation Algorithms for (Budgeted) Node-Weighted Steiner Problems

MohammadHossein Bateni[1], MohammadTaghi Hajiaghayi[2,⋆], and Vahid Liaghat[2,⋆]

[1] Google Research, 76 Ninth Avenue, New York, NY 10011
[2] Computer Science Department, Univ of Maryland, A.V.W. Bldg., College Park, MD 20742

Abstract. Moss and Rabani [13] study constrained node-weighted Steiner tree problems with two independent weight values associated with each node, namely, cost and prize (or penalty). They give an $O(\log n)$-approximation algorithm for the prize-collecting node-weighted Steiner tree problem (PCST)—where the goal is to minimize the cost of a tree plus the penalty of vertices not covered by the tree. They use the algorithm for PCST to obtain a bicriteria $(2, O(\log n))$-approximation algorithm for the Budgeted node-weighted Steiner tree problem—where the goal is to maximize the prize of a tree with a given budget for its cost. Their solution may cost up to twice the budget, but collects a factor $\Omega(\frac{1}{\log n})$ of the optimal prize. We improve these results from at least two aspects.

Our first main result is a primal-dual $O(\log h)$-approximation algorithm for a more general problem, prize-collecting node-weighted Steiner forest (PCSF), where we have h demands each requesting the connectivity of a pair of vertices. Our algorithm can be seen as a greedy algorithm which reduces the number of demands by choosing a structure with minimum cost-to-reduction ratio. This natural style of argument (also used by Klein and Ravi [11] and Guha et al. [9]) leads to a much simpler algorithm than that of Moss and Rabani [13] for PCST.

Our second main contribution is for the Budgeted node-weighted Steiner tree problem, which is also an improvement to Moss and Rabani [13] and Guha et al. [9]. In the unrooted case, we improve upon an $O(\log^2 n)$-approximation of [9], and present an $O(\log n)$-approximation algorithm without any budget violation. For the rooted case, where a specified vertex has to appear in the solution tree, we improve the bicriteria result of [13] to a bicriteria approximation ratio of $(1+\epsilon, O(\log n)/\epsilon^2)$ for any positive (possibly subconstant) ϵ. That is, for any permissible budget violation $1+\epsilon$, we present an algorithm achieving a tradeoff in the guarantee for prize. Indeed, we show that this is almost tight for the natural linear-programming relaxation used by us as well as in [13].

1 Introduction

In the rapidly evolving world of telecommunications and internet, design of fast and efficient networks is of utmost importance. It is not surprising, therefore, that the field of network design has continued to be an active area of research since its inception

⋆ Supported in part by NSF CAREER award 1053605, NSF grant CCF-1161626, ONR YIP award N000141110662, DARPA/AFOSR grant FA9550-12-1-0423, and a University of Maryland Research and Scholarship Award (RASA).

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 81–92, 2013.

several decades ago. These problems have applications not only in designing computer and telecommunications networks, but are also essential for other areas such as VLSI design and computational geometry [3]. Besides their appeals in these applications, basic network design problems (such as Steiner Tree, TSP, and their variants) have been the testbed for new ideas and have been instrumental in development of new techniques in the field of approximation algorithms.

In parallel to the study by Moss and Rabani [13], this work focuses on graph-theoretic problems in which two (independent) nonnegative weight functions are associated with the vertices, namely cost $c(v)$ and prize (or penalty) $\pi(v)$ for each vertex v of the given graph $G(V, E)$. The goal is to find a connected subgraph H of G that optimizes a certain objective. We now summarize the four different problems, already introduced in the literature. In the *Net Worth* problem (NW), the goal is to maximize the prize of H minus its cost[1]. It can be proved that this natural problem does not admit any finite approximation algorithm (see the full version of this work). A similar, yet better-known objective is that of minimizing the cost of the subgraph plus the penalty of nodes outside of it (which is called *Prize-Collecting Steiner Tree* (PCST) in the literature). Two other problems arise if one restricts the range of either cost or prize in the desired solution. In particular, the *Quota* problem tries to find the minimum-cost tree among those with a total prize surpassing a given value, whereas the *Budgeted* problem deals with maximizing the prize with a given maximum budget for the cost. The rooted variants ask, in addition, that a certain root vertex be included in the solution. In the k-MST problem, the goal is to find a minimum-cost tree with at least k vertices. In the k-STEINER TREE problem, given a set of terminals, the goal is to find a minimum-cost tree spanning at least k terminals. We show the following reductions missing from the literature.

Theorem 1. *Let α, $0 < \alpha < 1$, be a constant. The following statements are equivalent (both for edge-weighted and node-weighted variants):*

i There is an α-approximation algorithm for the rooted k-MST problem.
ii There is an α-approximation algorithm for the unrooted k-MST problem.
iii There is an α-approximation algorithm for the k-STEINER TREE problem.

Proof. Here we present the equivalence of (ii) and (iii) (see the full version of this work for that of (i) and (ii)). We note that one way is clear by definition. To prove that (iii) implies (ii), we give a cost-preserving reduction from k-STEINER TREE to k-MST. Let $< G = (V, E), T, k >$ be an instance of k-STEINER TREE with the set of terminals $T \subseteq V$. Let $n = |V|$. For every terminal $v_t \in T$, add n vertices at distance zero of v_t. Let $k' = kn + k$ and consider the solution to k'-MST on the new graph. Any subtree with at most $k-1$ terminals have at most $(k-1)n+n-1 = kn-1$ vertices. Therefore an optimal solution covers at least k terminals. Hence the reduction preserves the cost of optimal solution. □

These results improve the approximation ratio for k-Steiner tree. Previously, a 4-approximation algorithm was proved by [14] and a 5-approximation algorithm was due to [4] who had also conjectured the presence of a $2 + \epsilon$-approximation algorithm.

[1] The prize or cost of a subgraph is defined as the total prize or cost of its vertices, respectively

The equivalence of k-Steiner tree and k-MST combined with the 2-approximation result of Garg [7] leads to a 2-approximation algorithm for k-Steiner tree.

A more tractable version of the prize-collecting variant is the edge-weighted case in which the costs (but not the prizes) are associated with edges rather than nodes. The best known approximation ratio for the edge-weighted Steiner tree problem is 1.39 due to Byrka et al. [5]. For the earlier work on edge-weighted variant we refer the reader to the references of [5]. In this paper, unless otherwise specified all our graphs are node-weighted and undirected.

1.1 Contributions and Techniques

Approximation Algorithm for PCSF. Klein and Ravi [11] were the first to give an $O(\log h)$-approximation algorithm for the SF problem. Later, Guha et al. [9] improved the analysis of [11] by showing that the approximation ratio of the algorithm of [11] is w.r.t. the fractional optimal solution for the ST problem. The ST problem is a special case of SF where all demands share an endpoint. Very recently and independently of our work, Chekuri et al. [2] give an algorithm with an approximation ratio of $O(\log n)$ w.r.t. to the fractional solution for SF and higher connectivity problems. This immediately provides a reduction from PCSF to the SF problem: one can fractionally solve the LP for PCSF and pay the penalty of every demand for which the fractional solution pays at least half its penalty. Hence, the remaining demands can be (fractionally) satisfied by paying at most twice the optimal solution. Therefore, one can make a new instance of SF with only the remaining demands and get a solution within $O(\log n)$ factor of the optimal solution using the SF algorithm.

We start off by presenting a simple primal-dual $O(\log h)$-approximation algorithm for the node-weighted prize-collecting Steiner forest (PCSF) problem where h is the number of connectivity demands—see Theorem 2. Compared to the PCST algorithm given by Moss and Rabani [13] and Konemann et al. [12], our algorithm for PCSF solves a more general problem and it has a simpler analysis. A reader familiar with the moat-growing framework[2] may recall that algorithms in this framework (e.g., that of Moss and Rabani [13] or Könemann et al. [12]) consist of a *growth phase* and a *pruning phase*. A moat is a set of dual variables corresponding to a laminar set of vertices containing *terminals*—vertices with a positive penalty. The algorithm grows the moats by increasing the dual variables and adding other vertices gradually to guarantee feasibility. In the edge-weighted Steiner tree problem, when two moats collide on an edge, the algorithm buys the path connecting the moats and merges the moats. Roughly speaking, the algorithm stops growing a moat when either it reaches the root, or its total growth reaches the total prize of terminals inside it. This process is not quite enough to obtain a good approximation ratio. At the end of the algorithm we may have paid too much for connecting unnecessary terminals. Thus as a final step one needs to prune the solution in a certain way to obtain the tight approximation ratio of $2 - \frac{1}{n}$.

In the node-weighted problem, one obstacle is that (polynomially) many moats may collide on a vertex. Handling the proper growth of the moats and the process to merge them proves to be very sophisticated. This may have been the reason that for more than

[2] Introduced by Agrawal, Klein, and Ravi (AKR) [1] and Goemans and Williamson (GW) [8].

a decade no one noticed the flaw in the algorithm of Moss and Rabani [13][3]. Indeed the recently proposed algorithm by Könemann et al. [12] is even more sophisticated. In our algorithm, not only do we completely discard the pruning phase, but we also never merge the moats (thus intuitively, a moat forms a disk centered at a terminal). In fact, our algorithm can be thought of as a simple greedy algorithm. Our algorithm runs in iterations, and in each iteration several disks are grown simultaneously on different endpoints of the demands. The growth stops at the largest possible radius where there are no "overlaps" and no disk has run out of "penalty." If the disks corresponding to several endpoints hit each other, a set of paths connecting them is added to the solution and all but one representative endpoint are removed for the next iteration. However, if a disk is running out of penalty, the terminal at its center is removed for the next iteration. The cost incurred at each iteration is a fraction of OPT, proportional to the fraction of endpoints removed, hence the logarithmic term in the guarantee.

Although our primal-dual approach is different from the approach known for SF [11,9], we indeed use the same style of argument to analyze our algorithm. The crux of these algorithms is to reduce the number of components of the solution by using a structure with minimum cost-to-reduction ratio. Besides the simplicity of this trend, it is important that by avoiding the pruning phase, these algorithms may lead to progress in related settings such as streaming and online settings. The moat-growing approach of Konemann et al. [12], however, allows a stronger lagrangian-preserving guarantee [4] for PCST. This property is shown to be quite important for solving various problems such as k-MST and k-Steiner tree (see e.g. [4,10]).

Approximation Algorithms for the Budgeted Problem. Using their algorithm for PCST, Moss and Rabani developed a bicriteria[5] approximation algorithm for the Budgeted problem, one that achieves an approximation factor $O(\log n)$ on prize while violating the budget constraint by no more than factor two [13]. We present in Theorem 3 a modified pruning procedure that improves the bicriteria bound to $(1+\epsilon, O(\log n)/\epsilon^2)$; in other words, if the algorithm is allowed to violate the budget constraint by only a factor $1+\epsilon$ (for any positive ϵ), the approximation guarantee on the prize will be $O(\log n)/\epsilon^2$. In fact, we also show using the natural linear-programming relaxation (used in [13] as well), that it is not possible to improve these bounds significantly—see the full version. In particular, there are instances for which the fractional solution is OPT/ϵ, however, no solution of cost at most $1+\epsilon$ times the budget has prize more than $O(\mathrm{OPT})$. Our integrality-gap construction fails if the instance is not rooted. Indeed, in that case, we show how to obtain an $O(\log n)$-approximation algorithm with no budget violations—see Theorem 4. This improves the $O(\log^2 n)$-approximation algorithm of Guha et al. [9].[6] To get over the integrality gap of the LP formulation, we prove several

[3] In private correspondence the authors of the original work have admitted that their algorithm is flawed and that it cannot be fixed easily.

[4] Let T denote the sets of vertices purchased by the algorithm of [12]. It is guaranteed that $c(T) + \log(n)\pi(V \backslash T) \leq \log(n)\mathrm{OPT}$.

[5] An (α, β)-bicriteria approximation algorithm for the Budgeted problem finds a tree with total prize at least $\frac{1}{\beta}$ fraction of that of optimal solution and total cost at most α factor of the budget.

[6] The $O(\log^2 n)$-approximation algorithm can be derived from the results in [9] with some efforts, not as explicitly as cited by Moss and Rabani [13].

structural properties for near-optimal solutions. By restricting the solution to one with these properties, we use a bicriteria approximation algorithm as a black box to find a near-optimal solution. Finally we use a generalization of the trimming method of [9] to avoid violating the budget.

2 The Prize-Collecting Steiner Forest Problem

The starting point of the algorithm of Moss and Rabani [13] is a standard LP relaxation for the rooted version. For the Quota and Budgeted problems they show that any (fractional) feasible solution can be approximated by a convex combination of sets of nodes connected (integrally) to the root. Given the support of such a convex combination, it follows from an averaging argument that a proper set can be found. Thus the problem comes down to finding the support of the convex combination. They show that given a black-box algorithm which solves the PCST problem with the approximation factor $O(\log n)$, one can obtain the support in polynomial time.

The main result of this section is a very simple, and maybe more elegant algorithm for the classical problem of PCSF (and thus PCST). As mentioned before, using moats and having a pruning phase lead to the main difficulty in the analysis of previous algorithms. These seem to be a necessary evil for achieving a tight constant approximation factor for the edge-weighted variant. Surprisingly, we show neither is needed in the node-weighted variant. Instead of moats, we use dual *disks* which are centered on a *single* terminal and we do not need a pruning phase.

2.1 Preliminaries

Consider a graph $G = (V, E)$ with a node-weight function $c : V \to \mathbb{R}_{\geq 0}$. For a subset $S \subseteq V$, let $c(S) := \sum_{v \in S} c(v)$. In the *Steiner Forest* problem, given a set of demands $\mathcal{L} = \langle (s_1, t_1), \ldots, (s_h, t_h) \rangle$, the goal is to find a set of vertices X such that for every demand $i \in [h]$, s_i and t_i are connected in $G[X]$. The vertices s_i and t_i are denoted as the *endpoints* of the demand i. In PCSF a penalty (prize) $\pi_i \in \mathbb{R}_{\geq 0}$ is associated with every demand $i \in [h]$. If the endpoints of a demand are not connected in the solution, we need to pay the penalty of the demand. The objective cost of a solution $X \subseteq V$ is

$$\mathbf{PCSF}(X) = c(X) + \sum_{i \in [h] : i \text{ is not satisfied}} \pi_i.$$

A *terminal* is a vertex which is an endpoint of a demand. Let $\mathcal{T}$ denote the set of terminals. We may assume that the cost of a terminal is zero. We also assume the endpoints of all demands are different[7] (thus $|\mathcal{T}| = 2h$). For a pair of vertices u and v and a cost function c, let $d^c(u, v)$ denote the length of the shortest path with respect to c connecting u and v, including the cost of endpoints.

For a set of vertices S let $\delta(S)$ denote the set of vertices that are not in S but have neighbors in S. A set S *separates* a demand i if exactly one of s_i and t_i is in S. Let

[7] Both assumptions are without loss of generality. For every demand (s_i, t_i), attach a new vertex s^i of cost zero to s_i and similarly attach a new vertex t^i of cost zero to t_i. Now interpret i as the demand between s^i and t^i. The optimal cost does not change.

$\mathcal{S}_i$ denote the collection of sets separating the demand i and let $\mathcal{S} = \bigcup_i \mathcal{S}_i$. For a set S, define the penalty of S as half of the total penalty of demands separated by S, i.e., $\pi_{\mathcal{L}}(S) = \frac{1}{2}\sum_{i:S\in\mathcal{S}_i} \pi_i$. We may drop the index $\mathcal{L}$ when there is no ambiguity. The PCSF problem can be formulated as the following standard integer program (IP):

$$\begin{aligned}
&\text{Minimize} && \sum_{v\in V\setminus\mathcal{T}} c(v)\mathbf{x}(v) + \sum_{S\in\mathcal{S}} \pi(S)\mathbf{z}(S)\\
&\forall i\in[h], S\in\mathcal{S}_i && \sum_{v\in\delta(S)} \mathbf{x}(v) + \sum_{R|S\subseteq R\in\mathcal{S}_i} \mathbf{z}(R) \geq 1\\
& && \mathbf{x}(v), \mathbf{z}(S) \in \{0,1\}
\end{aligned}$$

Given a solution $X \subseteq V$ to the PCSF problem one can easily make a feasible solution $\mathbf{x}$ to the IP with the same objective value as $\mathbf{PCSF}(X)$: since the cost of a terminal is zero, we assume $\mathcal{T} \subseteq X$. For every vertex $v \in X$ set $\mathbf{x}(v) = 1$ and for every connected component CC of $G[X]$ set $\mathbf{z}(V\setminus CC) = 1$. It is also easy to verify since the cost of a terminal is zero, any (integral) feasible solution $\mathbf{x}$ corresponds to a solution $X \subseteq V$ for the PCSF problem with (at most) the same cost. One may relax the IP by allowing assignment of fractional values to the variables. Let OPT denote the objective value of the optimal solution for the relaxed linear program (LP). The following is the dual program $\mathcal{D}$ corresponding to the relaxed LP.

$$\begin{aligned}
&\text{Maximize} && \sum_{S\in\mathcal{S}} \mathbf{y}(S) && (\mathcal{D})\\
&\forall v\in V && \sum_{S\in\mathcal{S}:v\in\delta(S)} \mathbf{y}(S) \leq c(v)\\
&\forall S\in\mathcal{S} && \sum_{S'\subseteq S} \sum_{i:S,S'\in\mathcal{S}_i} \mathbf{y}_i(S') \leq \pi(S)\\
& && \mathbf{y}_i(S) \geq 0, \mathbf{y}(S) = \sum_{i:S\in\mathcal{S}_i} \mathbf{y}_i(S)
\end{aligned}$$

In the case of Steiner tree, the dual variables are defined w.r.t. a set S. However, in Steiner forest, the dual variables are in the form $\mathbf{y}_i(S)$, i.e., they are defined based on a demand as well. This has been one source of the complexity of previous primal-dual algorithms for Steiner forest problems. Interestingly, in our approach, we only need to work with a *simplified dual* constructed as follows.

Cores and Simplified Duals Let c and $\mathcal{L}$ denote a node-weight function and a set of demands, respectively. Let Z_c denote the set of vertices with zero cost. We note that the terminals are in Z_c. A set $C \subseteq V$ is a *core* if C is a connected component of $G[Z_c]$ *and* contains a terminal (i.e., an endpoint of a demand in $\mathcal{L}$). Let $\overline{\mathcal{S}(c,\mathcal{L})}$ be the collection of sets separating one core from the other cores, i.e., a set S is in $\overline{\mathcal{S}(c,\mathcal{L})}$ if S contains a core but has no intersection with other cores. For a set $S \in \overline{\mathcal{S}(c,\mathcal{L})}$, let $\mathbf{core}(S)$ denote the core inside S. Note that $\pi_{\mathcal{L}}(S) = \pi_{\mathcal{L}}(\mathbf{core}(S))$. A simplified dual w.r.t. c and $\mathcal{L}$ is the following program $\overline{\mathcal{D}(c,\mathcal{L})}$.

$$\text{Maximize} \quad \sum_{S \in \mathcal{S}} \mathbf{y}(S) \qquad (\overline{\mathcal{D}(c,\mathcal{L})})$$

$$\forall v \in V \quad \sum_{S \in \overline{\mathcal{S}(c,\mathcal{L})}: v \in \delta(S)} \mathbf{y}(S) \leq c(v) \qquad \text{(C1)}$$

$$\forall S \in \overline{\mathcal{S}(c,\mathcal{L})} \quad \sum_{S' : \mathbf{core}(S) \subseteq S' \subseteq S} \mathbf{y}(S') \leq \pi_{\mathcal{L}}(S) \qquad \text{(C2)}$$

$$\mathbf{y}(S) \geq 0$$

Observe that $\overline{\mathcal{S}(c,\mathcal{L})} \subseteq \mathcal{S}$. Indeed $\overline{\mathcal{D}(c,\mathcal{L})}$ is the same as $\mathcal{D}$ with only (much) fewer variables. Thus the program $\overline{\mathcal{D}(c,\mathcal{L})}$ is only more restricted than $\mathcal{D}$. In the rest of the paper, unless specified otherwise, by a dual we mean a simplified dual. When clear from the context, we may omit the indices c and $\mathcal{L}$.

Disks Consider a Dual Vector $\mathbf{y}$ Initialized to Zero. A *disk of radius R centered at a terminal t* is the dual vector obtained from the following process: Initialize the set S to the core containing t. Increase $\mathbf{y}(S)$ until for a vertex u the dual constraint C1 becomes tight. Add u to S and repeat with the new S. Stop the process when the total growth (i.e., sum of the dual variables) reaches R. A disk is *valid* if $\mathbf{y}$ is feasible. In what follows, by a disk we mean a valid disk unless specified otherwise.

A vertex v is *inside* the disk if $d^c(t,v)$ is strictly less than R. The *continent* of a disk is the set of vertices inside the disk. Further, we say a vertex v is on the *boundary* of a disk if it is not inside the disk but has a neighbor u such that $d^c(t,u) \leq R$. Note that u is not necessary inside the disk. The following facts about a disk of radius R centered at a terminal t can be derived from the definition:

Fact 1. *The (dual) objective value of the disk is exactly R.*

Fact 2. *For every vertex inside the disk, the dual constraint C1 is tight.*

Fact 3. *If a set S does not include the center, then $\mathbf{y}(S) = 0$. Further, if S is not a subset of the continent, then $\mathbf{y}(S) = 0$.*

Let $\mathbf{y}_1, \ldots, \mathbf{y}_k$ denote a set of disks. The *union* of the disks is simply a dual vector $\mathbf{y}$ such that $\mathbf{y}(S) = \sum_i \mathbf{y}_i(S)$ for every set $S \subseteq \mathcal{S}$. A set of disks are *non-overlapping* if their union is a feasible dual solution (i.e., both set of constraints C1 and C2 hold). If a vertex v is inside a disk, the corresponding dual constraint is tight. Thus for any set S such that $v \in \delta(S)$, the dual variable $\mathbf{y}(S)$ cannot be increased. On the other hand since the distance between v and the center is *strictly* less than the radius, there exists a set containing v with positive dual value. This observation leads to the following.

Proposition 1. *Let $\mathbf{y}$ be the union of a set of non-overlapping disks $\mathbf{y}_1, \ldots, \mathbf{y}_k$. A vertex inside a disk cannot be on the boundary of another disk.*

Proposition 1 implies that in the union of a set of non-overlapping disks, the continents are pairwise far from each other. This intuition leads to the following .[8]

[8] Due to the lack of space, we have omitted some of the proofs throughout the paper. We defer the reader to the full version of this paper for the omitted proofs.

Lemma 1. *Suppose T' is a subset of terminals such that the distance between every pair of them is non-zero. Let R denote the maximum radius such that the $|T'|$ disks of radius R centered at terminals in T' are non-overlapping. Consider the union of such disks. Either (i) the constraint C2 is tight for a continent; or (ii) the constraint C1 is tight for a vertex on the boundary of multiple disks.*

The final tool we need for the analysis of the algorithm states a precise relation between the dual variables and the distance of a vertex on the boundary.

Lemma 2. *Let v be a vertex on the boundary of a disk $\mathbf{y}$ of radius R centered at a terminal t. We have $\sum_{S|v\in\delta(S)} \mathbf{y}(S) = R - (d^c(t,v) - c(v))$.*

2.2 An Algorithm for the PCSF Problem

The algorithm finds the solution X iteratively. Let X_i denote the set of vertices bought after iteration i where X_0 is the set of terminals. For every i, the *modified cost function* c_i is a copy of c induced by setting the cost of vertices in X_{i-1} to zero, i.e., $c_i = c[X_{i-1} \to 0]$. At iteration i there is a set of *active* demands $\mathcal{L}_i \subseteq \mathcal{L}$ and the dual program $\mathcal{D}_i = \overline{\mathcal{D}(c_i, \mathcal{L}_i)}$. The program $\mathcal{D}_i$ is the simplified dual program w.r.t. the modified cost function and the active demands. Note that $\mathcal{D}_i$ is stricter than $\mathcal{D}$, thus the objective value of a feasible solution to $\mathcal{D}_i$ is a lower bound for OPT. The algorithm guarantees that for every $i < j$, $X_i \subseteq X_j$ and $\mathcal{L}_i$ is a superset of $\mathcal{L}_j$.

The algorithm is as follows (see Algorithm 1). We initialize $X_0 = \mathcal{T}$, $c_1 = c$, and $\mathcal{L}_1 = \mathcal{L}$. At iteration i, consider the cores formed w.r.t. c_i and $\mathcal{L}_i$. Let T_i denote a set which has exactly one terminal in each core (so the number of cores is $|T_i|$). The algorithm finds the maximum radius R_i such that the $|T_i|$ disks of radius R_i centered at each terminal in T_i are non-overlapping w.r.t. $\mathcal{D}_i$. By Lemma 1 either the constraint C2 is tight for a continent S; or the constraint C1 is tight for a vertex v on the boundary of multiple disks. In the former, deactivate every demand with exactly one endpoint in $\mathbf{core}(S)$; pay the penalty of such demands and continue to the next iteration with the remaining active demands. In the latter, let L_v denote the centers of the disks whose boundaries contain v. For every terminal $\tau \in L_v$ buy the shortest path w.r.t. c_i connecting v to τ (and so to the core containing τ). Deactivate a demand if its endpoints are now connected in the solution and continue to the next iteration. The algorithm stops when there is no active demand remaining; in which case it returns the final set of vertices bought by the algorithm.

We bound the objective cost of the algorithm in each iteration separately. The following theorem shows that the fraction of OPT we incur at each iteration is proportional to the reduction in the number of cores after the iteration.

Theorem 2. *The approximation ratio of Algorithm 1 is at most $2H_{2h}$ where H_{2h} is the $(2h)^{th}$ harmonic number.*

Proof. Observe that at each iteration, a core is a connected component of the solution which contains an endpoint of at least one active demand. We distinguish between two types of iterations: In Type I, Line 8 of Algorithm 1 is executed while in Type II, Line 15 is executed.

Algorithm 1. The Prize-Collecting Steiner Forest Algorithm

Input: A graph $G = (V, E)$, a set of demands $\mathcal{L}$ with penalties, and a cost function c.

1: Initialize $X_0 = \mathcal{T}$, $\mathcal{L}_1 = \mathcal{L}$, $c_1 = c$, and $i = 1$.
2: **while** $|\mathcal{L}_i| > 0$ **do**
3: Set $c_i = c[X_{i-1} \to 0]$ and construct the dual program $\mathcal{D}_i$ with respect to c_i and $\mathcal{L}_i$.
4: Construct T_i by choosing an arbitrary terminal from each core.
5: Let R_i be the maximum radius such that putting a disk of radius R_i centered at every terminal in T_i is feasible w.r.t. $\mathcal{D}_i$.
6: **if** the constraint C2 is tight for a continent S **then**
7: Set $X_i = X_{i-1}$.
8: Set $\mathcal{L}_{i+1} = \mathcal{L}_i \setminus \{j \in [h] |$ *either* $s_j \in \mathbf{core}(S)$ *or* $t_j \in \mathbf{core}(S)\}$.
9: **else**
10: Find a vertex v on the boundary of multiple disks for which constraint C1 is tight.
11: Let L_v denote the centers of the disks whose boundaries contain v.
12: Initialize $X_i = X_{i-1}$.
13: **for all** $\tau \in L_v$ **do**
14: Add the shortest path (w.r.t. c_i) between τ and v to X_i.
15: Set $\mathcal{L}_{i+1} = \mathcal{L}_i \setminus \{j \in [h] | d^{c_{i+1}}(s_j, t_j) = 0\}$.
16: $i = i + 1$.
17: Output X_{i-1}.

Observe that a demand is deactivated either at Line 8 or at Line 15. In the latter, the endpoints of a demand are indeed connected in the solution. Thus we only need to pay the penalty of a demand if it is deactivated in an iteration of Type I. Recall that at Line 8, the penalty of $\mathbf{core}(S)$ is half the total penalty of demands cut by S. Thus the total penalty we incur at that line is exactly $2\pi_{\mathcal{L}_i}(S)$

We now break the total objective cost of the algorithm into a payment P_i for each iteration i as follows:

$$P_i = \begin{cases} 2\pi_{\mathcal{L}_i}(S) & \text{for Type I iterations executing Line 8 with the continent } S; \\ c(X_i) - c(X_{i-1}) & \text{for Type II iterations.} \end{cases}$$

Recall that $|T_i|$ is the number of cores at iteration i. Observe that by Fact 1, at iteration i the total dual vector has value $R_i|T_i|$. By the weak duality $R_i \leq \frac{\text{OPT}}{|T_i|}$. For every $i \geq 1$, let $h_i = |T_i| - |T_{i+1}|$ denote the reduction in the number of cores after the iteration i.

Claim. $P_i \leq 2h_iR_i$ for every iteration i.

Proof. Fix an iteration i. Let $\mathbf{y}$ denote the union of disks of radius R_i centered at T_i. We distinguish between the two types of the iteration:

- *Type I.* At Line 8, by deactivating all the demands crossing a core, we essentially remove that core. Thus in such an iteration $h_i = 1$. The objective cost of the iteration is $2\pi_{\mathcal{L}_i}(S)$. On the other hand, the constraint C1 is tight for S, i.e., $\sum_{S' \subseteq S} \mathbf{y}(S) = \pi_{\mathcal{L}_i}(S)$. By Facts 1 and 3, the radius R_i equals $\sum_{S' \subseteq S} \mathbf{y}(S)$. Therefore the objective cost is at most $2h_iR_i$
- *Type II.* At line 15, we connect $|L_v|$ cores to each other, thus reducing the number of cores in the next iteration by at least $h_i \geq |L_v| - 1$. Recall that by Lemma 1,

$|L_v| \geq 2$ and hence $h_i \geq 1$. The total cost of connecting terminals in L_v to v is bounded by $c_i(v)$ plus for every $\tau \in L_v$, the cost of the path connecting τ to v excluding $c_i(v)$. Thus $P_i \leq c_i(v) + \sum_{\tau \in L_v} (d^{c_i}(\tau, v) - c_i(v))$. Now we write the equation in Lemma 2 for every disk centered at a terminal in L_v:

$$\begin{aligned} |L_v| R_i &= \sum_{\tau \in L_v} [d^{c_i}(\tau, v) - c_i(v) + \sum_{S | v \in \delta(S), \tau \in S} \mathbf{y}(S)] \\ &= \sum_{\tau \in L_v} [d^{c_i}(\tau, v) - c_i(v)] + c_i(v) \geq P_i, \end{aligned}$$

where the last equality follows since the constraint C1 is tight for v. Since the disks are non-overlapping, by Fact 3, $\mathbf{y}(S)$ is positive only if it contains a single terminal of L_v. This completes the proof since $P_i \leq |L_v| R_i \leq (h_i + 1) R_i \leq 2 h_i R_i$. □

Let X be the final solution of the algorithm. Note that $|T_{i+1}| = |T_i| - h_i$ and $|T_1| \leq |\mathcal{T}|$. A simple calculation shows

$$\mathbf{PCSF}(X) \leq \sum_i P_i \leq \sum_i 2 h_i R_i \leq 2\mathrm{OPT} \sum_i \frac{h_i}{|T_i|} \leq 2\mathrm{OPT} \cdot H_{|\mathcal{T}|}.$$ □

3 The Budgeted Steiner Tree Problem

In this section we consider the Budgeted problem in the node-weighted Steiner tree setting. Recall that for a vertex $v \in V$, we denote the prize and the cost of the vertex by $\pi(v)$ and $c(v)$, respectively. First we generalize the trimming process of Guha et al. [9] which reduces the budget violation of a solution while preserving the prize-to-cost ratio. We use this process to obtain a bicriteria approximation algorithm for the rooted version in Section 3.1. Next, in Section 3.2 we consider the unrooted version. By providing a structural property of near-optimal solutions, we propose an algorithm which achieves a logarithmic approximation factor without violating the budget constraint; improving on the previous result of Guha et al. [9] which obtains an $O(\log^2 n)$-approximation algorithm without violation.

In what follows, for a rooted tree T we assume a *subtree* rooted at a vertex v consists of all vertices whose path to the root of T passes through v. The set of *strict subtrees* of T consists of all subtrees other than T itself. Further, the set of *immediate subtrees* of T are the subtrees rooted at the children of the root of T.

3.1 The Rooted Budgeted Problem

For a budget value B and a vertex r, a graph is *B-proper for the vertex r* if the cost of reaching any vertex from r is at most B. The following lemma shows a bicriteria trimming method (proof in the full version).

Lemma 3. *Let T be a subtree rooted at r with the prize-to-cost ratio γ. Suppose the underlying graph is B-proper for r and for $\epsilon \in (0, 1]$ the cost of the tree is at least $\frac{\epsilon B}{2}$. One can find a tree T^* containing r with the prize-to-cost ratio at least $\frac{\epsilon}{4}\gamma$ such that $\frac{\epsilon}{2} B \leq c(T^*) \leq (1 + \epsilon) B$.*

Moss and Rabani [13] give an $O(\log n)$-approximation algorithm for the Budgeted problem which may violate the budget by a factor of two. Using Lemma 3 one can trim such a solution to achieve a trade-off between the violation of budget and the approximation factor (proof in the full version) .

Theorem 3. *For every $\epsilon \in (0, 1]$ one can find a subtree $T \subseteq G$ in polynomial time such that $c(T) \leq (1 + \epsilon)B$ and the total prize of T is $\Omega(\frac{\epsilon^2}{\log n})$ fraction of OPT.*

3.2 The Unrooted Budgeted Problem

We prove a stronger variant of Lemma 3 for the unrooted version. We show that if no single vertex is too expensive, one does not need to violate the budget at all. The analysis is similar to that of Lemma 3.

Lemma 4. *Let T be a tree with the prize to cost ratio γ. Suppose $c(T) \geq \frac{B}{2}$ and the cost of every vertex of the tree is at most $\frac{B}{2}$ for a real number B. One can find a subtree $T^* \subseteq T$ with the prize to cost ratio at least $\frac{\gamma}{4}$ such that $\frac{B}{4} \leq c(T^*) \leq B$.*

One may use arguments similar to that of Theorem 3 to derive an $O(\log n)$-approximation algorithm from Lemma 4 when the cost of a vertex is not too big. On the other hand if the cost of a vertex is more than half the budget, we can guess that vertex and try to solve the problem with the remaining budget. However, one obstacle is that this process may need to be repeated, i.e., the cost of another vertex may be more than half the remaining budget. Thus we may need to continue guessing many vertices in which case connecting them in an optimal manner would not be an easy task. The following theorem shows indeed guessing one vertex is sufficient if one is willing to lose an extra factor of two in the approximation guarantee.

Theorem 4. *The unrooted budgeted problem admits an $O(\log n)$-approximation algorithm which does not violate the budget constraint.*

Proof. We define two classes of subtrees: the *flat* trees and the *saddled* trees. A tree is flat if the cost of every vertex of the tree is at most $\frac{B}{2}$. For a tree T, let x be the vertex of T with the largest cost. The tree T is saddled if $c(x) > \frac{B}{2}$ and the cost of every other vertex of the tree is at most $\frac{B-c(x)}{2}$. Let T_f^* denote the optimal flat tree, i.e., a flat tree with the maximum prize among all the flat trees with the total cost at most B. Similarly, let T_s^* denote the optimal saddled tree.

The proof is described in two parts. First we show the prize of the best solution between T_f^* and T_s^* is indeed in a constant factor of OPT(see the following claim, proof in the full version). Next, we show by restricting the optimum to any of the two classes, an $O(\log(n))$-approximation solution can be found in polynomial time. Therefore this would give us the desired approximation algorithm.

Claim. Either $\pi(T_f^*) \geq \frac{\text{OPT}}{2}$ or $\pi(T_s^*) \geq \frac{\text{OPT}}{2}$.

Now we only need to restrict the algorithm to flat trees and saddled trees. Indeed we can reduce the case of saddled trees to flat trees. We simply guess the maximum-cost vertex x (by iterating over all vertices). We form a new instance of the problem by

reducing the budget to $B - c(x)$ and the cost of x to zero. The cost of every other vertex in T_s^* is at most half the remaining budget, thus we need to look for the best flat tree in the new instance. Therefore it only remains to find an approximation solution when restricted to flat trees.

We use Lemma 4 to find the desired solution for flat trees. A vertex with cost more than half the budget cannot be in a flat tree, thus we remove all such vertices. We may guess a vertex of the best solution and by using the algorithm of Moss and Rabani [13] we can find an $O(\log n)$-approximation solution which may use twice the budget. Let T be the resulting tree with the total prize P. If $c(T) \leq B$ we are done. Otherwise by Lemma 4 we can trim T to obtain a tree with the cost at most B and the prize at least $\frac{P}{32}$ which completes the proof. □

References

1. Agrawal, A., Klein, P., Ravi, R.: When trees collide: an approximation algorithm for the generalized Steiner problem on networks. In: STOC (1991)
2. Chekuri, C., Ene, A., Vakilian, A.: Prize-collecting survivable network design in node-weighted graphs. In: Gupta, A., Jansen, K., Rolim, J., Servedio, R. (eds.) APPROX/RANDOM 2012. LNCS, vol. 7408, pp. 98–109. Springer, Heidelberg (2012)
3. Cheng, X., Li, Y., Du, D.-Z., Ngo, H.Q.: Steiner trees in industry. In: Handbook of Combinatorial Optimization (2005)
4. Chudak, F.A., Roughgarden, T., Williamson, D.P.: Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. Mathematical Programming 100, 411–421 (2004)
5. Erlebach, T., Grant, T., Kammer, F.: Maximising lifetime for fault-tolerant target coverage in sensor networks. In: SPAA (2011)
6. Feigenbaum, J., Papadimitriou, C.H., Shenker, S.: Sharing the cost of multicast transmissions. Journal of Computer and System Sciences 63, 21–41 (2001)
7. Garg, N.: Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In: STOC (2005)
8. Goemans, M., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM J. on Computing 24, 296–317 (1992)
9. Guha, S., Moss, A., Naor, J(S.), Schieber, B.: Efficient recovery from power outage. In: STOC (1999)
10. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-Median problems using the primal-dual schema and Lagrangian relaxation. J. ACM
11. Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted Steiner trees. J. Algorithms 19(1), 104–115 (1995)
12. Könemann, J., Sadeghian, S., Sanita, L.: An LMP $O(\log n)$-approximation algorithm for node weighted prize collecting Steiner tree (unpublished, 2013)
13. Moss, A., Rabani, Y.: Approximation algorithms for constrained node weighted Steiner tree problems. SIAM J. Comput. 37(2), 460–481 (2007)
14. Ravi, R., Sundaram, R., Marathe, M.V., Rosenkrantz, D.J., Ravi, S.S.: Spanning trees - short or small. SIAM J. Discrete Math. 9(2), 178–200 (1996)

Search-Space Size in Contraction Hierarchies

Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner

Karlsruhe Institute of Technology
firstname.lastname@kit.edu

Abstract. Contraction hierarchies are a speed-up technique to improve the performance of shortest-path computations, which works very well in practice. Despite convincing practical results, there is still a lack of theoretical explanation for this behavior.

In this paper, we develop a theoretical framework for studying search space sizes in contraction hierarchies. We prove the first bounds on the size of search spaces that depend solely on structural parameters of the input graph, that is, they are independent of the edge lengths. To achieve this, we establish a connection with the well-studied elimination game. Our bounds apply to graphs with treewidth k, and to any minor-closed class of graphs that admits small separators. For trees, we show that the maximum search space size can be minimized efficiently, and the average size can be approximated efficiently within a factor of 2.

We show that, under a worst-case assumption on the edge lengths, our bounds are comparable to the recent results of Abraham et al. [1], whose analysis depends also on the edge lengths. As a side result, we link their notion of highway dimension (a parameter that is conjectured to be small, but is unknown for all practical instances) with the notion of pathwidth. This is the first relation of highway dimension with a well-known graph parameter.

1 Introduction

Contraction hierarchies were introduced by Geisberger et al. [7], who evaluated their performance experimentally. Given a directed graph $G = (V, E)$ and a vertex $v \in V$, *contraction of* v means (i) removing v, and (ii) inserting a shortcut uw of length $\operatorname{dist}_G(u, w)$ for each unique shortest path (u, v, w) in G. Given an order $\alpha\colon V \to \{1, \dots, n\}$ of the vertices of $G = (V, E)$, a *contraction hierarchy* $\bar{G}_\alpha = (\bar{G}^\wedge_\alpha, \bar{G}^\vee_\alpha)$ of G is obtained by iteratively contracting the vertices in the order specified by α. Let E' denote the set of shortcuts that is created in this process. Then $\bar{G}^\wedge_\alpha = (V, E^\wedge_\alpha)$ and $\bar{G}^\vee_\alpha = (V, E^\vee_\alpha)$, where $E^\wedge_\alpha = \{uv \in E \cup E' \mid \alpha(u) < \alpha(v)\}$ and $E^\vee_\alpha = \{uv \in E \cup E' \mid \alpha(v) < \alpha(u)\}$.

The correctness of shortest path computation relies on the following proposition, which is due to Geisberger et al. [7]. It immediately implies that distance queries can be performed by a bidirectional query, searching $\bar{G}^\wedge_\alpha$ and $\bar{G}^\vee_\alpha$.

Proposition 1. $\operatorname{dist}_G(s, t) = \min_{v \in V} \operatorname{dist}^\wedge_{\bar{G}}(s, v) + \operatorname{dist}^\vee_{\bar{G}}(v, t)$ *for all* $s, t \in V$.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 93–104, 2013.

In the following, we refer to such a contraction hierarchy as an *algorithmic contraction hierarchy.* Obviously, the contraction hierarchy depends strongly on the ordering α. Finding a good node ordering that allows fast shortest-path computations thus is an important problem. Practical implementations, such as the one by Geisberger et al. [7] employ heuristics for which no provable guarantees are known. Previous theoretical expositions rather focus on minimizing the size of the contraction hiearchy [4,11]. In particular, it is known that minimizing the size of a contraction hierarchy is NP-complete. The only work providing provable performance guarantees for shortest-path computations in contraction hierarches, we are aware of, is the work of Abraham et al. [1,2]. They introduce the notion of highway dimension, a parameter that is conjectured to be small in real-world road networks, and prove sublinear query times under this assumption. However, the highway dimension of real-world instances is unknown, and may change as the length function changes. By contrast, we use separator decompositions and focus on providing bounds that rely on purely structural parameters of the graph, such as bounded treewidth or excluding a fixed minor. Our algorithms thus apply to a larger class of graphs, and are not dependent on the length function.

We note that theoretical results with better query times [17,6] exist, some of them even using similar techniques. They are, however, far from being practical. By contrast, our theoretical bounds apply to a widely used speed-up technique. It is also worth noting that recursive graph separation has been used as a heuristic in practical approaches [16], although, without providing theoretical guarantees.

Contribution and Outline. We develop a theoretical framework for studying search-space sizes in contraction hierarchies. The iterative definition of an algorithmic contraction hierarchy is difficult to work with. In Section 2 we derive a global description of the contraction hierarchy associated with a node ordering.

Afterwards, in Section 3, we establish a connection between contraction hiearchies and two classical problems that have been widely studied. Namely, so-called filled graphs, which were introduced by Parter [12] in his analysis of Gaussian elimination, and elimination trees, which were introduced by Schreiber [15] for Gaussian elimination on sparse matrices. For trees, this implies an efficient algorithm for minimizing the maximum search space and a 2-approximation for the average search space. This contrasts hardness results for other speed-up techniques, such as arcflags, where optimal preprocessing for trees is NP-complete [3].

In Section 4, we show that nested dissection, a technique for finding elimination trees of small height, can be applied to construct orders α with provable bounds on the maximum search space size. For graphs of treewidth k and for graphs that admit small separators and exclude a fixed minor, we obtain maximum search space size $O(k \log n)$ and $O(\sqrt{n})$, respectively.

Finally, we compare our results with the results of Abraham et al. [1,2] in Section 5. If the length function is such that the highway dimension is maximal, then our results are comparable to theirs. However, our approach neither requires small maximum degree, nor does it depend on the diameter of the graph, and thus applies to a larger class of graphs.

2 A Formal Model of Contraction Hierarchies

In this section, we develop a theoretical model of contraction hierarchies that is simpler to work with than algorithmic contraction hierarchies. Let $G = (V, E)$ be a directed graph and let α be an ordering of V. Let $P_\alpha(s,t) = \{v \in V \mid \alpha(v) \geq \min\{\alpha(s), \alpha(t)\}$ and $\text{dist}_G(s,v) + \text{dist}_G(v,t) = \text{dist}_G(s,t) < \infty\}$, i.e., $P_\alpha(s,t)$ contains the vertices that lie on a shortest path from s to t and lie above at least one of s and t. The following theorem provides a global characterization of the algorithmic contraction hierarchy.

Theorem 1. *Let $G = (V, A)$ be a weighted digraph and let α be an order of its vertices. The arcs $A^\wedge_\alpha$ and $A^\vee_\alpha$ of $\bar{G}^\wedge_\alpha$ and $\bar{G}^\vee_\alpha$ are*

$$\begin{aligned} A^\wedge_\alpha &= \{uv \in A \mid \alpha(u) < \alpha(v)\} \cup \{uv \mid \alpha(u) < \alpha(v) \text{ and } P_\alpha(u,v) = \{u,v\}\} \\ A^\vee_\alpha &= \{uv \in A \mid \alpha(u) > \alpha(v)\} \cup \{uv \mid \alpha(u) > \alpha(v) \text{ and } P_\alpha(u,v) = \{u,v\}\}. \end{aligned}$$

The length of a shortcut uv in $\bar{G}^\wedge_\alpha$ or $\bar{G}^\vee_\alpha$ is $\text{dist}_G(u,v)$.

Not only does this theorem shed some light on the structure of algorithmic contraction hierarchies, but also suggests an alternative definition. Consider an arc st of G that is no unique shortest path. Then, removing st from G does not change any distances. If st is a unique shortest path, then $P_\alpha(s,t) = \{s,t\}$ anyway. Thus the following definition works equally well. For a weighted digraph $G = (V, E)$ and an order α of its vertices, we define $G_\alpha = (G^\wedge_\alpha, G^\vee_\alpha)$, where $G^\wedge_\alpha = (V, A^\wedge_\alpha)$ and $G^\vee_\alpha = (V, A^\vee_\alpha)$ by

$$\begin{aligned} A^\wedge_\alpha &= \{uv \mid \alpha(u) < \alpha(v) \text{ and } P_\alpha(u,v) = \{u,v\}\} \\ A^\vee_\alpha &= \{uv \mid \alpha(u) > \alpha(v) \text{ and } P_\alpha(u,v) = \{u,v\}\}. \end{aligned}$$

As in Theorem 1, we set $\text{len}^\alpha_G(uv) = \text{dist}_G(u,v)$. We call the pair $G_\alpha = (G^\wedge_\alpha, G^\vee_\alpha)$ a *formal contraction hierarchy.*

We remark that it immediately follows from this definition that if H is the digraph obtained from G by reversing all arcs, $H^\wedge_\alpha = G^\vee_\alpha$ and $H^\vee_\alpha = G^\wedge_\alpha$ hold. This allows us to prove statements about G_α by only considering $G^\wedge_\alpha$ since the analogous statement for $G^\vee_\alpha$ follows by reversing all arcs.

We now carry over several useful concepts from the algorithmic definition. A *shortcut* is an arc uw of G_α that is not contained in G, or for which $\text{len}_G(uw) > \text{dist}_G(u,w)$. Note that the latter type of shortcuts are included only to model the possible overwriting of arclengths in algorithmic contraction hierarchies. Given a shortcut uw in $A^\wedge$ or $A^\vee$, we are able to recover the vertex v that would have caused the insertion of uw in the corresponding algorithmic contraction hierarchy. Namely, let $S = \{v \in V \setminus \{u,w\} \mid \text{dist}_G(u,v) + \text{dist}_G(v,w) = \text{dist}_G(u,w)\}$. Note that $S \neq \emptyset$, for otherwise uw would be no shortcut. Note further that $\alpha(v) < \alpha(u)$ and $\alpha(v) < \alpha(w)$ for all $v \in S$ since otherwise we would have $v \in P_\alpha(u,w) = \{u,w\}$. We call the vertex $v \in S$ with $\alpha(v)$ maximal the *supporting vertex of uw*. It is easy to show that exactly the contraction of v causes the insertion of uw in the algorithmic contraction hierarchy $\bar{G}_\alpha$.

Lemma 1. *Let uw be a shortcut of G_α and let v be its supporting vertex. Then G_α contains $uv \in A^\vee_\alpha$ and $vw \in A^\wedge_\alpha$.*

We call the arcs, whose existence is guaranteed by Lemma 1, the *supporting arcs* of uw, and write $\sup(uw) = (uv, vw)$. Observe that if uv is a supporting arc of uw, then $\alpha(v) < \alpha(w)$, and thus chains of supporting arcs in $G^\wedge_\alpha$ and $G^\vee_\alpha$ are acyclic and do not descend indefinitely. This allows us to perform induction on the depth of the nested shortcuts below a given arc uw. We define the *shortcut depth* $\mathrm{scd}(uw)$ of an arc of G_α by $\mathrm{scd}(uw) = 1$ if uw is no shortcut, and by $\mathrm{scd}(uw) = \mathrm{scd}(uv) + \mathrm{scd}(vw)$ if uw is a shortcut with $\sup(uw) = (uv, vw)$. It is readily seen that, for a shortcut uw with $\sup(uw) = (uv, vw)$, we have $\mathrm{len}^\alpha_G(uw) = \mathrm{len}^\alpha_G(uv) + \mathrm{len}^\alpha_G(vw)$. Hence G_α still possesses the most essential properties of the algorithmic contraction hierarchy $\bar{G}_\alpha$.

A close look at the proof of Proposition 1 in Section 1 reveals that it merely depends on the fact that each arc uw with $P_\alpha(u, w) = \{u, w\}$ is contained in $\bar{G}_\alpha$ and has length $\mathrm{len}^\alpha_{\bar{G}}(uw) = \mathrm{dist}_G(u, w)$. Thus Proposition 1 also holds for formal contraction hierarchies, implying that a bidirectional variant of Dijkstra's algorithm on the contraction hierarchy G_α can be used to compute shortest paths in G.

To measure the performance of such computations, we define the search space of a query as $S(s, G^\wedge_\alpha) = \{u \in V \mid \mathrm{dist}^\wedge_G(s, u) < \infty\}$ and $R(t, G^\vee_\alpha) = \{u \in V \mid \mathrm{dist}^\vee_G(u, t) < \infty\}$. Clearly, the shortest-path query from s to t in G_α settles at most the vertices in $S(s, G^\wedge_\alpha) \cup R(t, G^\vee_\alpha)$. To maximize the performance of query algorithms, one is interested in an ordering α that minimizes $\max_{s,t \in V} |S(s, G^\wedge_\alpha)| + |R(t, G^\vee_\alpha)|$. To simplify the analysis, we rather concentrate on minimizing the *maximum search space size* $S_{\max}(G_\alpha) = \max\{|S(v, G^\wedge_\alpha)|, |R(v, G^\vee_\alpha)|\}$. Note that $2 \cdot S_{\max}(G_\alpha)$ is an upper bound on the number of vertices that is settled in any query, and thus bounding $S_{\max}(G_\alpha)$ gives a guarantee on the query performance in terms of the number of settled nodes. We denote the minimum maximum search space size by $S_{\max}(G) = \min_\alpha S_{\max}(G_\alpha)$. Similarly, we define the *average search space size* $S_{\mathrm{avg}}(G_\alpha) = 1/n^2 \cdot \sum_{s,t \in V} |S(s, G^\wedge_\alpha)| + |R(t, G^\vee_\alpha)|$.

There is still one downside of formal contraction hierarchies: Practical implementations of contraction hierarchies do not compute an actual formal (or even algorithmic) contraction hierarchy. Instead of inserting a shortcut only when it is strictly necessary, fast heuristics are used to quickly exclude the necessity of a shortcut in many cases. In some cases, this results in the addition of shortcuts that are not necessary. Thus bounding $S_{\max}(G_\alpha)$ may not have any practical implications since the additional shortcuts might increase the search space arbitrarily. To overcome this downside, we introduce one final type of contraction hierarchies, which also allows for additional shortcuts, yet preserve the properties that have turned out to be key to contraction hierarchies.

A *weak contraction hierarchy* H_α of a weighted digraph $G = (V, A)$ is a pair $(H^\wedge_\alpha, H^\vee_\alpha)$ of digraphs $H^\wedge_\alpha = (V, B^\wedge_\alpha)$ and $H^\vee_\alpha = (V, B^\vee_\alpha)$, such that the following conditions are satisfied.

(w1) $G_\alpha \subseteq H_\alpha$

(w2) $\alpha(u) < \alpha(v)$ for each $uv \in B^\wedge_\alpha$ and each $vu \in B^\vee_\alpha$

(w3) If uw is an arc of H_α that is not contained in G, then there is at least one pair of arcs $uv \in B_\alpha^\vee$ and $vw \in B_\alpha^\wedge$.

In the remainder of this section, we indicate how to extend our previous findings for contraction hierarchies to weak contraction hierarchies and investigate the relationship between different weak contraction hierarchies for the same ordering α. For this purpose, we fix a weighted digraph G and an ordering α of its vertices. As usual, we denote its formal contraction hierarchy by $G_\alpha = (G_\alpha^\wedge, G_\alpha^\vee)$. Additionally, we fix a weak contraction hierarchy $H_\alpha = (H_\alpha^\wedge, H_\alpha^\vee)$, whose arcs we denote by $B_\alpha^\wedge$ and $B_\alpha^\vee$, as above.

The notions of shortcuts and shortcut depth carry over literally to H_α. It follows immediately from (w1) and (w3) that for each shortcut uw in H_α, there is a pair of supporting arcs $uv \in B_\alpha^\vee$ and $vw \in B_\alpha^\wedge$. Although distances and arc lengths are only of secondary importance in the remaining sections, we still want to point out that it is not hard to give a "correct" definition of arc lengths on H, such that the following lemma holds true.

Lemma 2. *Let H_α be a weak contraction hierarchy.*
(a) $\operatorname{len}_H^\alpha(uw) \geq \operatorname{dist}_G(u, w)$ *for all arcs uw of H_α.*
(b) $\operatorname{len}_H^\alpha(uw) = \operatorname{dist}_G(u, w)$ *for all arcs uw of H_α with $P_\alpha(u, w) = \{u, w\}$.*

As indicated above, the proof of Proposition 1 relies on exactly the containment of G_α and the properties guaranteed by Lemma 2, and it thus holds also for any weak contraction hierarchy. In particular, the same shortest-path algorithm works for weak contraction hierarchies.

In view of property (w1) it is clear that G_α is the smallest weak contraction hierarchy. Moreover, if H_α and K_α are weak contraction hierarchies, then $(H_\alpha^\wedge \cup K_\alpha^\wedge, H_\alpha^\vee \cup K_\alpha^\vee)$ is a weak contraction hierarchy. Thus, there exists a unique maximal weak contraction hierarchy, which we denote by M_α. It is not difficult to see that $S(u, G_\alpha^\wedge) \subseteq S(u, H_\alpha^\wedge) \subseteq S(u, M_\alpha^\wedge)$, and symmetrically $R(u, G_\alpha^\vee) \subseteq R(u, H_\alpha^\vee) \subseteq R(u, M_\alpha^\vee)$ for all weak contraction hierarchies H_α. In particular, this shows that $S_{\max}(G_\alpha) \leq S_{\max}(H_\alpha) \leq S_{\max}(M_\alpha)$ for all weak contraction hierarchies H_α. Thus, we will concentrate on bounding $S_{\max}(M_\alpha)$ in the following sections. Before we do so, we give a more explicit description of M_α.

Lemma 3. *A weak contraction hierarchy H_α is maximal if and only if H_α satisfies the following properties.*
(i) Each arc of G is contained in H_α.
(ii) For any two arcs $uv \in B_\alpha^\vee$ and $vw \in B_\alpha^\wedge$, H_α also contains uw.

Note that this immediately implies an efficient way to construct the arcs of M_α by inserting shortcuts between each pair of neighbors during the contraction. In particular, the structure of M_α is independent of the weights on G.

Finally, we note that the query performance does not solely depend on the number of vertices in the search space, but also on the number of arcs. For a search space $S(u, M_\alpha)$, this number is certainly bounded by $|S(u, M_\alpha)|^2$. Moreover, the size $|M_\alpha|$ has a crude upper bound in terms of $S_{\max}(M_\alpha)$.

Lemma 4. *For any n-vertex directed graph with an ordering α of its vertices, we have $|M_\alpha| \leq 2n \cdot S_{\max}(M_\alpha)$.*

3 Contraction Hierarchies and Filled Graphs

In this section, we establish a link between contraction hierarchies and the more well-studied graph elimination game, which was introduced by Parter [12]. Let $G = (V, E)$ be an undirected graph and let α be an ordering of its vertices. We consider the so-called *elimination game* played on G. Beginning at $G^1 = G$, one removes in each step $i = 1, \dots, n$ the vertex $v_i = \alpha^{-1}(i)$ and its incident edges from G^i. Afterwards, the graph G^{i+1} is obtained from G^i by inserting *fill edges*, such that the neighbors of v_i form a clique. Denote by F^i the set of edges inserted in step i, and let $F = \bigcup_{i=1}^{n} F_i$. The filled graph G^α is now commonly defined to be the undirected graph with edge set $E \cup F$. For our purposes it is more convenient to define the filled graph as the according directed graph with all arcs pointing upwards with respect to α. That is, the *filled graph* $G^\alpha = (V, A^\alpha)$ is defined by $A^\alpha = \{uv \mid \{u, v\} \in E \cup F \text{ and } \alpha(u) < \alpha(v)\}$. Note that the only difference from the construction of M_α is that M_α is constructed from a digraph, whereas the elimination game is played on an undirected graph. In what follows, we denote for a digraph G by ${}^*G$ the underlying undirected graph. The following theorem immediately follows from the construction of M_α and G^α.

Theorem 2. *Let G be a directed graph with an ordering α of its vertices. Let further $\overleftarrow{M}_\alpha^\vee$ denote $M_\alpha^\vee$ with reversed arcs. Then $M_\alpha^\wedge, \overleftarrow{M}_\alpha^\vee \subseteq {}^*G^\alpha$. Moreover, if G contains for each arc uv also the opposite arc vu, then $M_\alpha^\wedge = \overleftarrow{M}_\alpha^\vee = {}^*G^\alpha$.*

Theorem 2 has many far-reaching consequences, and we will only explore a few of them in this paper. It turns out that the definition of M_α is nothing essentially new, and has indeed already been defined and studied by Rose and Tarjan [13]. However, much of the work on filled graphs is primarily concerned with the problem of minimizing the number of arcs in G^α; see the survey by Heggernes [10]. Minimizing the fill-in corresponds to minimizing the number of shortcuts in a contraction hierarchy, and hence its space requirements. We rather focus on the implications of Theorem 2 regarding search spaces and their size.

Corollary 1. *Let $G = (V, A)$ be a weighted digraph with vertex ordering α. Then $S(u, M_\alpha^\wedge), R(u, M_\alpha^\vee) \subseteq S(u, {}^*G^\alpha)$. In particular $S_{\max}(G_\alpha) \leq S_{\max}({}^*G^\alpha)$.*

An analogous statement holds for the average search space size S_{avg}. Our next goal is an alternative description of $S_{\max}({}^*G^\alpha)$ known as the height of the elimination tree of ${}^*G^\alpha$. For this purpose consider again an undirected graph G and a filled graph G^α. Associated with G^α is the so-called *elimination tree* $T(G^\alpha)$ of G. The elimination tree $T(G^\alpha)$ has vertex set V but contains for each $u \in V$ only the arc uv of G^α with minimal $\alpha(v)$. Again the usual definition of $T(G^\alpha)$ is undirected, but it is natural to choose $\alpha^{-1}(n)$ as the root. With this choice the usual definition coincides with our definition. The height of $T(G^\alpha)$ with respect to this root is the *elimination tree height*, denoted by $\text{ht}(G^\alpha)$. The following lemmas relate the search space size in G^α with $\text{ht}(G^\alpha)$.

Lemma 5. *Let G be a connected graph, α an order of V and let $T = T(G^\alpha)$. Then $S_{\max}(T) = 1 + \text{ht}(G^\alpha)$.*

Proof. Let r denote the root of T, and denote by $p(u)$ the vertices lying on the path from u to r. Then $\mathrm{ht}(G^\alpha) = \max_{u \in V} |p(u)| - 1$ and it therefore suffices to show $S(u,T) = p(u)$ for all $u \in V$. This is trivially satisfied, since due to the connectivity of G, each vertex $u \in V$ distinct from r is the source of precisely one arc uv. □

The crucial property of $T(G^\alpha)$ is that if u and v are two vertices connected by a path in p in the filled graph G^α of G, then there is a path p' from u to v in $T(G^\alpha)$. It obviously suffices to prove this statement when p is an arc, as the general case then follows by induction.

Lemma 6. *If uv is an arc of the filled graph G^α, then there exists a path with source u and target v in $T(G^\alpha)$.*

Proof. Denote by $p(u)$ the unique path in $T(G^\alpha)$ from u to the root r. We show by descending induction on $\alpha(u)$ that we have $p(v) \subseteq p(u)$ for all arcs uv of G^α. Note that this implies our claim, for it then follows that $p(u) = q \cdot p(v)$, where q is a path from u to v.

If $\alpha(u) = n - 1$, then $\alpha(v) = n$, and hence $v = r$. The claim holds trivially.

If $\alpha(u) < n - 1$, let uw be the unique arc of $T(G^\alpha)$ with source u. By the definition of $T(G^\alpha)$, we have $\alpha(w) \leq \alpha(v)$, and $p(u) = uw \cdot p(w)$. If $v = w$, we are done. Otherwise, G^α contains the arc wv as both v and w are neighbors of u at the time of its removal during the elimination game. The induction hypothesis therefore implies $p(v) \subseteq p(w)$, and hence $p(v) \subseteq p(u)$. □

This allows us to finally relate search spaces in G^α and $T(G^\alpha)$.

Corollary 2. *Let $G = (V, E)$ be a graph, α an order on V and $T = T(G^\alpha)$ the corresponding elimination tree. Then $S(u,T) = S(u,G^\alpha)$ for all $u \in V$.*

Corollary 1 and 2 immediately imply the following.

Corollary 3. *For any connected weighted digraph G with vertex ordering α,*

$$S_{\max}(M_\alpha) \leq \mathrm{ht}({}^*G^\alpha) + 1\,.$$

Despite its innocent appearance, the above corollary is central to our analysis of search spaces in contraction hierarchies, for it enables us to translate upper bounds on $\mathrm{ht}({}^*G^\alpha)$ into upper bounds on $S_{\max}(G)$. Upper bounds from the literature are not seldomly accompanied by algorithms to determine orders α so that $\mathrm{ht}({}^*G^\alpha)$ is at most the upper bound at hand. Without any further modifications these algorithms may be used to compute contraction orders α with good upper bounds on $S_{\max}(G^\alpha)$.

As a first application of the above result let us consider contraction hierarchies of undirected trees T. In this case, shortest paths are unique and each possible shortcut is thus present in the contraction hierarchy T_α. Hence, T_α, the maximal weak contraction hierarchy M_α and the filled graph T^α coincide. Moreover, Schäffer [14] has given a linear-time algorithm to compute optimal elimination orders for trees, and we may thus conclude that the problem of minimizing $S_{\max}(T_\alpha)$ is solvable in linear time. The techniques of the next section can also be used to obtain a 2-approximation for $S_{\mathrm{avg}}(T_\alpha)$; we omit the details.

4 Contraction Orders from Nested Dissection

Next we show how to compute orders that yield small search spaces for several classes of graphs. The main idea is to exploit Corollary 3, which relates search space sizes and elimination tree heights. One way to construct orderings that give guarantees on the elimination tree height is the use of *nested dissection*, which goes back to George [8].

Let $0 < b < 1$ and let $f(n)$ be a monotonically increasing function. A *(b,f)-balanced separator decomposition* of an undirected n-vertex graph $G = (V, E)$ is a rooted tree $\mathcal{T} = (\mathcal{X}, \mathcal{E})$ whose nodes $X \in \mathcal{X}$ are disjoint subsets of V and that is recursively defined as follows. If $n \leq n_0$ for some fixed constant n_0, then $\mathcal{T}$ consists of a single node $X = V$. If $n > n_0$, then a (b, f)-balanced separator decomposition of G consists of a root $X \subseteq V$ of size at most $f(n)$ whose removal separates G into at least two subgraphs $G_1, \ldots, G_d$ with at most bn vertices, each. The children of X in $\mathcal{T}$ are the roots of (b, f)-balanced separator decompositions of $G_1, \ldots, G_d$. For clarity, we will always refer to the vertices of $\mathcal{T}$ as *nodes*. We use $\mathcal{T}_X$ to denote the subtree of $\mathcal{T}$ rooted at a node X, and by G_X the connected subgraph of G induced by the vertices contained in $\mathcal{T}_X$. For a vertex $u \in V$, we denote the unique node X of $\mathcal{T}$ with $u \in X$ by X_u. A node X of $\mathcal{T}$ has *level* $\text{level}(X) = i$ if the unique simple path from X to the root of $\mathcal{T}$ has length i.

Remark 1. Let G be an undirected graph and let $\mathcal{T}$ be a (b, f)-balanced separator decomposition of G. If $\{u, v\}$ is an edge of G with $\text{level}(X_u) \geq \text{level}(X_v)$, then X_v is an ancestor of X_u.

Proof. Consider the lowest common ancestor X of X_u and X_v in $\mathcal{T}$. If $X \neq X_v$, then X_u and X_v lie in distinct subtrees of $\mathcal{T}_X$. However, by construction of $\mathcal{T}$, this means that X separates X_u from X_v, contradicting the existence of the edge $\{u, v\}$. □

Given a (b, f)-balanced separator decomposition of G, we determine an associated *(b, f)-balanced nested dissection order* $\alpha = \alpha(\mathcal{T})$ on the vertices of G by performing a post-order traversal of $\mathcal{T}$, where the vertices of each node are visited in an arbitrary order. It follows immediately from Remark 1 and the construction of α that for any edge $\{u, v\}$ of G with $\alpha(u) < \alpha(v)$ the node X_v is an ancestor of X_u. This property remains valid also for the corresponding filled graph G^α.

Lemma 7. *Let $G = (V, E)$ be an undirected graph and $\alpha = \alpha(\mathcal{T})$ a nested dissection order associated with a given (b, f)-balanced separator decomposition $\mathcal{T} = (\mathcal{X}, \mathcal{E})$ of G. Then X_v is an ancestor of X_u for any arc uv of the filled graph G^α with $\alpha(u) < \alpha(v)$.*

Proof. It suffices to that that X_v is an ancestor of X_u for each edge $\{u, v\}$ with $\alpha(u) < \alpha(v)$ of G^i, where G^i is the graph before the ith step in the elimination game. Observe that the above discussion establishes exactly this property for $G = G^1$. We show that G^{i+1} satisfies the property if G^i does.

Consider the vertex $v_i = \alpha^{-1}(i)$ that is removed in the ith step of the elimination game. Let $\{u, v\}$ be a fill-edge with $\alpha(u) < \alpha(v)$ that is inserted in this step. Then G^i contains edges $\{v_i, u\}$ and $\{v_i, v\}$. By the induction hypothesis, we have that both X_u and X_v are ancestors of X_{v_i} in $\mathcal{T}$ since $\alpha(v_i)$ is minimal among the vertices in G^i. Note that this implies that either X_u is an ancestor of X_v or vice versa. Our assumption $\alpha(u) < \alpha(v)$ and the construction of α imply that X_u is an ancestor of X_v. This finishes the proof. □

To simplify notation, we denote by $\mathcal{T}(u)$ the union of all nodes that lie on the unique simple path from X_u to the root of $\mathcal{T}$. The above lemma has immediate implications in terms of search spaces and elimination tree height. An easy induction over the length of a path in G^α yields the following.

Corollary 4. *Let $G = (V, E)$ be an undirected graph, $\mathcal{T} = (\mathcal{X}, \mathcal{E})$ an (b, f)-balanced separator decomposition of G, and let $\alpha = \alpha(\mathcal{T})$ be an associated nested dissection order. Then (i) $S(u, G^\alpha) \subseteq \mathcal{T}(u)$, and (ii) $\mathrm{ht}(G^\alpha) \leq |\mathcal{T}(u)|$.*

In particular, Corollary 1 and 4 together imply the following theorem.

Theorem 3. *Let $G = (V, E)$ be a weighted digraph and $\alpha = \alpha(\mathcal{T})$ a nested dissection order of a (b, f)-balanced separator decomposition of ${}^*G$. Then we have $|S(u, M_\alpha^\wedge)|, |R(u, M_\alpha^\vee)| \leq |\mathcal{T}(u)|$.*

In order to find upper bounds on $S_{\max}(M_\alpha)$, it remains to bound $|\mathcal{T}(u)|$. This issue cannot be handled simultaneously for all families of graphs, but needs special treatment depending on the properties of G. We will first study a rather general setting, and afterwards specialize to graphs that exclude a fixed minor.

It is not hard to see that $|\mathcal{T}(u)| \leq n_0 + h \cdot f(n)$, where h is the height of $\mathcal{T}$. Further, it follows from the balance of the decomposition that $h \leq \log_{1/b} n$. In particular, for graphs with treewidth at most k, the $(1/2, 1)$-balanced separator decomposition for trees facilitates a $(1/2, k+1)$-balanced separator decomposition with $n_0 = k + 1$. We have the following theorem.

Theorem 4. *Let G be a weighted digraph of treewidth at most k. There exists an order α, such that $S_{\max}(M_\alpha) \leq (k+1)(1+\log n)$ and $|M_\alpha| \leq 2n(k+1)(1+\log n)$.*

If the separator size is not fixed, but depends on the graph size, better bounds can be achieved. For example for minor-closed graph classes that admit $(b, a\sqrt{n})$-balanced separators, we have $\mathrm{ht}(G^\alpha) \leq n_0 + \sum_{i=0}^{\infty} a\sqrt{b^i n} = n_0 + a/(1-\sqrt{b})\sqrt{n} = O(\sqrt{n})$. According to Lemma 4, this yields a contraction hierarchy of size $O(n^{3/2})$. However, a more sophisticated analysis due to Gilbert and Tarjan [9] proves that the number of fill arcs is $O(n \log n)$. The next theorem summarizes this discussion.

Theorem 5. *Let $\mathcal{C}$ be a minor-closed graph class with balanced $O(\sqrt{n})$-separators. Any $G \in \mathcal{C}$ admits an order α with $S_{\max}(M_\alpha) = O(\sqrt{n})$ and $|M_\alpha| = O(n \log n)$.*

5 Comparison with Highway Dimension

In this section, we compare our bounds with the ones obtained by Abraham et al. [1,2]. Their results employ the highway dimension, a notion that, unlike the graph parameters we use, also depends on the edge lengths. We show that their bounds are comparable to ours if the edge lengths are sufficiently ill-behaved.

First, we briefly recall the definition of highway dimension. Let $G = (V, E)$ be a weighted undirected graph. Given a vertex $u \in V$, the set of vertices v of distance at most ε from u is called the *ball of radius* ε around u, and is denoted by $B_\varepsilon(u)$. We say that a vertex u covers a shortest path p if p contains u. The *highway dimension* $\mathrm{hd}(G)$ of G is the smallest integer d, such that all the shortest paths p in G of length $\varepsilon < \mathrm{len}(p) \leq 2\varepsilon$ that intersect a given ball of radius 2ε can be covered by at most d vertices. Abraham et al. [1] prove that for a n-vertex weighted graph G with highway dimension d and diameter D and maximum degree Δ, there exists an ordering α, such that the size of G^α is $O(nd \log D)$, and such that distance queries can be answered in time $O((\Delta + d \log D) \cdot d \log D)$.

In the remainder of this section, we proceed as follows. We consider the edge lengths of G that maximize $\mathrm{hd}(G)$. For this particular choice of lengths, we construct a $(2/3, \mathrm{hd}(k))$-balanced separator decomposition of G, which then provides the link to nested dissection orders. We note that the same proof can be adapted to the slightly different definition of highway dimension in [2].

Lemma 8. *Let $G = (V, E)$ be a connected graph, let k be the maximum highway dimension over all possible edge lengths on G, and let $H \subseteq G$ be a connected subgraph with $|V(H)| \geq 2k + 2$. Then H can be separated into at least two connected components of size at most $\lceil |V(H)|/2 \rceil$ by removing at most k vertices.*

Proof. Let H be a connected subgraph of G with h vertices and let $H_1 \subseteq H$ be a connected subgraph with $\lfloor h/2 \rfloor$ vertices. Denote the vertex sets of H and H_1 by V_H and V_1, respectively. Define lengths $\mathrm{len}\colon E \to \mathbb{R}^+$ by setting the length of an edge to 1 if it has exactly one endpoint in V_1, and to $\varepsilon = 3/h$, otherwise. Observe that the length of a simple path in H_1 is at most $h\varepsilon/2 \leq 3/2$.

Consider a ball B with radius $3/2$ around any vertex u of H_1. Then $V_1 \subseteq B$. By our choice of k, and the definition of highway dimension, there exists a set S of at most k vertices, such that each shortest path in G that intersects H_1 and has length between $3/4$ and $3/2$ contains at least one element of S. We claim that the removal of $S \cap V_H$ separates H into at least two connected components with at most $\lceil h/2 \rceil$ vertices, each.

Consider any path with source $s \in V_1$ and target $t \in V \setminus V_1$. This path necessarily contains an edge $\{u, v\}$ with $u \in V_1$ and $v \in V \setminus V_1$. By the choice of edge lengths, this edge is a shortest path of length 1, and thus one of its endpoints is in S. Hence S separates H_1 from $H \setminus V_1$. It remains to verify that the connected components of $H \setminus S$ contain at most $\lceil h/2 \rceil$ vertices, each, and that there are at least two such components. The former claim follows immediately from our choice of H_1. For the latter claim, note that $|V_1| \geq k + 1$ and $|V_H \setminus V_1| \geq k + 1$ imply $|V_1 \setminus S| \geq |V_1| - k \geq 1$ and $|V_H \setminus (V_1 \cup S)| \geq |V_H \setminus V_1| - k \geq 1$. □

This lemma allows us to separate each subgraph of $n' \geq 2k+2$ vertices into at least two connected components with at most $\lceil n'/2 \rceil \leq \frac{2}{3}n'$ vertices by removing at most k vertices. We have the following corollary.

Corollary 5. *Let $G = (V, E)$ be a connected undirected graph with maximum highway dimension k. Then G admits a $(2/3, k)$-balanced separator decomposition, whose leaves have size at most $2k+1$.*

A simple calculation shows that $\mathrm{ht}(G) \leq 2k+1+k \cdot \log_{3/2} n$. It is known for the pathwidth $\mathrm{pw}(G)$ that $\mathrm{pw}(G) \leq \mathrm{ht}(G)$ [5].

Theorem 6. *Let G be a weighted undirected graph. There exist edge lengths on G, such that $\mathrm{hd}(G) \geq (\mathrm{pw}(G)-1)/(\log_{3/2} n + 2)$.*

To our knowledge, this is a novel and unanticipated relation between highway dimension and more commonly used graph parameters. Moreover, Corollary 5 allows a comparison of our results with those of Abraham et al. [1].

Theorem 7. *Let G be an undirected graph with diameter D and maximum degree Δ. Let β denote the order constructed by Abraham et al. [1]. There exist edge lengths on G and a nested dissection order α, such that*
(a) $|M_\alpha| \leq O(\log n / \log D)|G_\beta|$, and
(b) the worst-case running time of distance queries in M_α is at most a factor of $O(\log^2(n)/\log^2(D))$ greater than that in G_β.

Proof. Choose the edge lengths such that G attains its maximum highway dimension k. Recall from [1], that their optimal order β results in a contraction hierarchy G_β that has $m_\beta = O(nk\log(D))$ arcs and on which a distance query has worst-case running time $T^\beta_{\text{query}} = O((\Delta + k\log D) \cdot k \log D)$.

By virtue of Theorem 4 and Corollary 5, there exists a nested-dissection ordering α, such that $S_{\max}(M_\alpha) = O(2k+1+k\log n) = O(k\log n)$. Using Lemma 4, we have $|M_\alpha| = O(nk\log n)$, which immediately implies (a). For (b), we use that Dijkstra's algorithm relaxes at most $S_{\max}(M_\alpha)^2$ edges. □

We note that, for graphs that bear some resemblance to road networks, it seems quite likely that $\Theta(\log n) = \Theta(\log D)$. It is remarkable that the results are so close, given that Abraham et al. bound both the vertices and arcs in the search space, while our crude bound on the number of arcs is simply the square of the number of vertices in the search space. Any improvement on this bound would immediately imply faster query times. It is moreover worth noting that our machinery neither requires small maximum degree nor small diameter.

Conclusion. We have developed a theoretical framework for studying search spaces in contraction hierarchies. Our main contributions are a global description of contraction hierarchies and the connection to elimination games. Using nested dissection, we are able to compute contraction orders with sublinear search spaces for large classes of graphs. Under a worst-case assumption on the highway dimension, our results, even though our constructions ignore edge lengths, are comparable to those of Abraham et al. [1]. Our main open questions are: (i) Are there stronger bounds on the number of arcs in search spaces? (ii) Is there an efficient approximation for the maximum or average search space size?

References

1. Abraham, I., Delling, D., Fiat, A., Goldberg, A.V., Werneck, R.F.: VC-Dimension and Shortest Path Algorithms. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 690–699. Springer, Heidelberg (2011)
2. Abraham, I., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In: Proc. 21st Annual ACM–SIAM Symp. Disc. Alg. (SODA 2010), pp. 782–793. SIAM (2010)
3. Bauer, R., Baum, M., Rutter, I., Wagner, D.: On the Complexity of Partitioning Graphs for Arc-Flags. In: Proc. 12th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2012). OpenAccess Series in Informatics (OASIcs), pp. 71–82 (2012)
4. Bauer, R., Columbus, T., Katz, B., Krug, M., Wagner, D.: Preprocessing Speed-Up Techniques Is Hard. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 359–370. Springer, Heidelberg (2010)
5. Bodlaender, H.L., Gilbert, J.R., Kloks, T., Hafsteinsson, H.: Approximating treewidth, pathwidth, and minimum elimination tree height. In: Schmidt, G., Berghammer, R. (eds.) WG 1991. LNCS, vol. 570, pp. 1–12. Springer, Heidelberg (1992)
6. Fakcharoenphol, J., Rao, S.: Negative weight edges, shortest paths, near linear time. In: Proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS 2001), pp. 232–241 (2001)
7. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact Routing in Large Road Networks Using Contraction Hierarchies. Transport. Sci. 46(3), 388–404 (2012)
8. George, A.: Nested dissection of a regular finite element mesh. SIAM Journal on Numerical Analysis 10(2), 345–363 (1973)
9. Gilbert, J.R., Tarjan, R.E.: The analysis of a nested dissection algorithm. Numerische Mathematik 50(4), 377–404 (1986)
10. Heggernes, P.: Minimal triangulations of graphs: A survey. Discrete Mathematics 306(3), 297–317 (2006)
11. Milosavljević, N.: On optimal preprocessing for contraction hierarchies. In: Proc. 5th Internat. Workshop Comput. Transport. Sci., IWCTS 2012 (2012)
12. Parter, S.V.: The use of linear graphs in gaussian elimination. SIAM Review 3(2), 119–130 (1961)
13. Rose, D.J., Tarjan, R.E.: Algorithmic aspects of vertex elimination on directed graphs. SIAM Journal on Applied Mathematics 34(1), 176–197 (1978)
14. Schäffer, A.A.: Optimal node ranking of trees in linear time. Information Processing Letters 33(2), 91–96 (1989)
15. Schreiber, R.: A new implementation of sparse Gaussian elimination. ACM Transactions on Mathematical Software (TOMS) 8(3), 256–276 (1982)
16. Schulz, F., Wagner, D., Zaroliagis, C.D.: Using multi-level graphs for timetable information in railway systems. In: Mount, D.M., Stein, C. (eds.) ALENEX 2002. LNCS, vol. 2409, pp. 43–59. Springer, Heidelberg (2002)
17. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. Journal of the ACM 51(6), 993–1024 (2004)

Time-Efficient Quantum Walks for 3-Distinctness*

Aleksandrs Belovs[1], Andrew M. Childs[2,4], Stacey Jeffery[3,4], Robin Kothari[3,4], and Frédéric Magniez[5]

[1] Faculty of Computing, University of Latvia
[2] Department of Combinatorics & Optimization, University of Waterloo, Canada
[3] David R. Cheriton School of Computer Science, University of Waterloo, Canada
[4] Institute for Quantum Computing, University of Waterloo, Canada
[5] CNRS, LIAFA, Univ Paris Diderot, Sorbonne Paris-Cité, France

Abstract. We present two quantum walk algorithms for 3-Distinctness. Both algorithms have time complexity $\tilde{O}(n^{5/7})$, improving the previous $\tilde{O}(n^{3/4})$ and matching the best known upper bound for query complexity (obtained via learning graphs) up to log factors. The first algorithm is based on a connection between quantum walks and electric networks. The second algorithm uses an extension of the quantum walk search framework that facilitates quantum walks with nested updates.

1 Introduction

Element Distinctness is a basic computational problem. Given a sequence $\chi = \chi_1, \ldots, \chi_n$ of n integers, the task is to decide if those elements are pairwise distinct. This problem is closely related to Collision, a fundamental problem in cryptanalysis. Given a 2-to-1 function $f : [n] \to [n]$, the aim is to find $a \neq b$ such that $f(a) = f(b)$. One of the best (classical and quantum) algorithms is to run Element Distinctness on f restricted to a random subset of size $\sqrt{n}$.

In the quantum setting, Element Distinctness has received a lot of attention. The first non-trivial algorithm used $\tilde{O}(n^{3/4})$ time [1]. The optimal $\tilde{O}(n^{2/3})$ algorithm is due to Ambainis [2], who introduced an approach based on quantum walks that has become a major tool for quantum query algorithms. The optimality of this algorithm follows from a query lower bound for Collision [3]. In the query model, access to the input χ is provided by an oracle whose answer to query $i \in [n]$ is χ_i. This model is the quantum analog of classical decision tree complexity: the only resource measured is the number of queries to the input.

Quantum query complexity has been a very successful model for studying the power of quantum computation. In particular, bounded-error quantum query

* This paper is a merge of two submitted papers, whose full versions are available at `http://arxiv.org/abs/1302.3143` and `http://arxiv.org/abs/1302.7316`. Support for this work was provided by European Social Fund within the project "Support for Doctoral Studies at University of Latvia", the European Commission IST STREP project 25596 (QCS), NSERC, the Ontario Ministry of Research and Innovation, the US ARO, and the French ANR Blanc project ANR-12-BS02-005 (RDAM).

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 105–122, 2013.

complexity has been exactly characterized in terms of a semidefinite program, the general adversary bound [4, 5]. To design quantum query algorithms, it suffices to exhibit a solution to this semidefinite program. However, this turns out to be difficult in general, as the minimization form of the general adversary bound has exponentially many constraints. Belovs [6] recently introduced the model of learning graphs, which can be viewed as the minimization form of the general adversary bound with additional structure imposed on the form of the solution. This additional structure makes learning graphs much easier to reason about. The learning graph model has already been used to improve the query complexity of many graph problems [6–9] as well as k-Distinctness [10].

One shortcoming of learning graphs is that these upper bounds do not lead explicitly to efficient algorithms in terms of time complexity. Although the study of query complexity is interesting on its own, it is relevant in practice only when a query lower bound is close to the best known time complexity.

Recently, [11] reproduced several known learning graph upper bounds via explicit algorithms in an extension of the quantum walk search framework of [12]. This work produced a new quantum algorithmic tool, quantum walks with nested checking. Algorithms constructed in the framework of [11] can be interpreted as quantum analogs of randomized algorithms, so they are simple to design and analyze for any notion of cost, including time as well as query complexity. This framework has interpreted all known learning graphs as quantum walks, except the very recent *adaptive learning graphs* for k-Distinctness [10].

In k-Distinctness, the problem is to decide if there are k copies of the same element in the input, with $k = 2$ being Element Distinctness. The best lower bound for k-Distinctness is the Element Distinctness lower bound $\Omega(n^{2/3})$, whereas the best query upper bound is $O(n^{1-2^{k-2}/(2^k-1)}) = o(n^{3/4})$ [10], achieved using learning graphs, improving the previous bound of $O(n^{k/(k+1)})$ [2]. However, the best known time complexity remained $\tilde{O}(n^{k/(k+1)})$. We improve this upper bound for the case with $k = 3$ using two distinct approaches.

For the first approach, described in Sections 2 and 3, we use a connection between quantum walks and electric networks. Hitting and commute times of random walks are closely connected to the effective resistance of associated networks of resistors. We develop a similar connection for the case of quantum walks. For any initial distribution over the vertices of a graph, we prove that a quantum walk can detect the presence of a marked element in $O(\sqrt{WR})$ steps, where W is the total weight of the graph and R is the effective resistance. This generalizes a result of Szegedy that only applies if the initial distribution is stationary.

The second approach, described in Sections 4 and 5, uses quantum walks with nested updates. The basic idea is conceptually simple: we walk on sets of 2-collisions and look for a set containing a 2-collision that is part of a 3-collision. We check if a set has this property by searching for an index that evaluates to the same value as one of the 2-collisions in the set. However, to move to a new set of 2-collisions, we need to use a quantum walk subroutine for finding 2-collisions as part of our update step. This simple idea is surprisingly difficult to implement and leads us to develop a new extension of the quantum walk search framework.

2 Quantum Walks and Electric Networks

2.1 Random Walks and Electric Networks

Let $G = (V, E)$ be a simple undirected graph with each edge assigned a *weight* $w_e \geq 0$. Let $W = \sum_{e \in E} w_e$ be the *total weight*. Consider the following *random walk* on G: If the walk is at a vertex $u \in V$, proceed to a vertex v with probability proportional to w_{uv}, i.e., $w_{uv}/(\sum_{ux \in E} w_{ux})$. The random walk has a *stationary probability distribution* $\pi = (\pi_u)$ given by $\pi_u = \sum_{uv \in E} w_{uv}/(2W)$.

Let $\sigma = (\sigma_u)$ be an *initial probability distribution* on the vertices of the graph, and let $M \subseteq V$ be some set of *marked vertices*. We are interested in the *hitting time* $H_{\sigma,M}$ of the random walk: the expected number of steps of the random walk required to reach a vertex in M when the initial vertex is sampled from σ. If σ is concentrated in a vertex $s \in V$, or M consists of a single element $t \in V$, we replace σ by s or M by t. The *commute time* between vertices s and t is defined as $H_{s,t} + H_{t,s}$. We assume G and σ are known, and the task is to determine whether M is non-empty.

Assume M is non-empty, and define a *flow* on G from σ to M as a real-valued function p_e on the (oriented) edges of the graph satisfying the following conditions. First, $p_{uv} = -p_{vu}$. Next, for each non-marked vertex u,

$$\sigma_u = \sum_{uv \in E} p_{uv}. \tag{1}$$

That is, σ_u units of the flow are injected into u, traverse through the graph, and are removed from marked vertices. Define the *energy* of the flow as

$$\sum_{e \in E} \frac{p_e^2}{w_e}. \tag{2}$$

Clearly, the value of (2) does not depend on the orientation of each e. The *effective resistance* $R_{\sigma,M}$ is the minimal energy of a flow from σ to M. For R, as for H, we also replace σ and M by the corresponding singletons. The resistance $R_{\sigma,M}$ equals the energy dissipated by the electric flow where the edges have conductance w_e and σ_u units of the current are injected into each u and collected in M [13]. The following two results can be easily obtained from the results in Ref. [14]:

Theorem 1. *If G, w, W are as above, s, t are two vertices of G, $M \subseteq V$, and π is the stationary distribution on G, then*

(a) the commute time between s and t equals $2WR_{s,t}$ and
(b) the hitting time $H_{\pi,M}$ equals $2WR_{\pi,M}$.

We show a quadratic improvement in the quantum case: If G and σ are known in advance and the superposition $\sum_{u \in V} \sqrt{\sigma_u}|u\rangle$ is given, the presence of a marked vertex in G can be determined in $O(\sqrt{WR})$ steps of a quantum walk. By combining this result with the second statement of Theorem 1, we obtain the main result of the paper by Szegedy [15].

2.2 Tools from Quantum Computing

Although we use the language of electric networks to state our results, we use spectral properties of unitary operators in the algorithms.

Lemma 1 (Effective Spectral Gap Lemma [5]). *Let Π_A and Π_B be two orthogonal projectors in the same vector space, and $R_A = 2\Pi_A - I$ and $R_B = 2\Pi_B - I$ be the reflections about their images. Assume P_Θ, where $\Theta \geq 0$, is the orthogonal projector on the span of the eigenvectors of $R_B R_A$ with eigenvalues $\mathsf{e}^{\mathsf{i}\theta}$ such that $|\theta| \leq \Theta$. Then, for any vector w in the kernel of Π_A, we have*

$$\|P_\Theta \Pi_B w\| \leq \frac{\Theta}{2}\|w\|.$$

Theorem 2 (Phase Estimation [16], [17]). *Assume a unitary U is given as a black box. There exists a quantum algorithm that, given an eigenvector of U with eigenvalue $\mathsf{e}^{\mathsf{i}\phi}$, outputs a real number w such that $|w-\phi| \leq \delta$ with probability at least $9/10$. Moreover, the algorithm uses $O(1/\delta)$ controlled applications of U and $\frac{1}{\delta}\operatorname{polylog}(1/\delta)$ other elementary operations.*

2.3 Szegedy-Type Quantum Walk

In this section, we construct a quantum counterpart of the random walk described in Section 2.1. The quantum walk differs slightly from the quantum walk of Szegedy. The framework of the algorithm goes back to [18], and Lemma 1 is used to analyze its complexity. We assume the notations of Section 2.1 throughout the section.

It is customary to consider quantum walks on bipartite graphs. Let $G = (V, E)$ be a bipartite graph with parts A and B. Also, we assume the support of σ is contained in A. These are not very restrictive assumptions: If either of them fails, consider the bipartite graph G' with the vertex set $V' = V \times \{0, 1\}$, the edge set $E' = \{(u,0)(v,1), (u,1)(v,0) : uv \in E\}$, edge weights $w'_{(u,0)(v,1)} = w'_{(u,1)(v,0)} = w_{uv}$, the initial distribution $\sigma'_{(u,0)} = \sigma_u$, and the set of marked vertices $M' = M \times \{0, 1\}$.

We assume the quantum walk starts in the state $|\varsigma\rangle = \sum_{u\in V} \sqrt{\sigma_u}\,|u\rangle$ that is known in advance. Also, we assume there is an upper bound R known on the effective resistance from σ to M for all potential sets M of marked vertices.

Now we define the vector space of the quantum walk. Let S be the *support* of σ, i.e., the set of vertices u such that $\sigma_u \neq 0$. The vectors $\{|u\rangle : u \in S\} \cup \{|e\rangle : e \in E\}$ form the computational basis of the vector space of the quantum walk. Let $\mathcal{H}_u$ denote the *local space* of u, i.e., the space spanned by all $|uv\rangle$ for $uv \in E$ and, additionally, $|u\rangle$ if $u \in S$. We have that $\bigoplus_{u\in A} \mathcal{H}_u$ equals the whole space of the quantum walk, and $\bigoplus_{u\in B} \mathcal{H}_u$ equals the subspace spanned by $|e\rangle$ for $e \in E$.

The *step of the quantum walk* is defined as $R_B R_A$ where $R_A = \bigoplus_{u\in A} D_u$ and $R_B = \bigoplus_{u\in B} D_u$ are the direct sums of the *diffusion* operations. Each D_u is a reflection operation in $\mathcal{H}_u$. All D_u in R_A or R_B commute, which makes them easy to implement in parallel. They are as follows:

- If a vertex u is marked, then D_u is the identity, i.e., the reflection about $\mathcal{H}_u$;
- If u is not marked, then D_u is the reflection about the orthogonal complement of $|\psi_u\rangle$ in $\mathcal{H}_u$, where

$$|\psi_u\rangle = \sqrt{\frac{\sigma_u}{C_1 R}}\,|u\rangle + \sum_{uv\in E} \sqrt{w_{uv}}\,|uv\rangle \tag{3}$$

for some large constant $C_1 > 0$ we choose later. The vector $|\psi_u\rangle$ is not necessarily normalized. This also holds for $u \notin S$: then the first term in (3) disappears.

Theorem 3. *The presence of a marked vertex can be detected with bounded error in $O(\sqrt{RW})$ steps of the quantum walk.*

Proof. Similarly to the Szegedy algorithm, we may assume S is disjoint from M. We perform phase estimation on $R_B R_A$ starting in $|\varsigma\rangle$ with precision $1/(C\sqrt{RW})$ for some constant C. We accept iff the phase is 1. The complexity estimate follows from Theorem 2. Let us prove the correctness. We start with the positive case. Let p_e be a flow from σ to M with energy at most R. First, using the Cauchy-Schwarz inequality and the fact that S is disjoint from M, we get

$$RW \geq \left(\sum_{e\in E} \frac{p_e^2}{w_e}\right)\left(\sum_{e\in E} w_e\right) \geq \sum_{e\in E} |p_e| \geq 1. \tag{4}$$

Now, we construct an eigenvalue-1 eigenvector

$$|\phi\rangle = \sqrt{C_1 R}\sum_{u\in S} \sqrt{\sigma_u}|u\rangle - \sum_{e\in E} \frac{p_e}{\sqrt{w_e}}|e\rangle$$

of $R_B R_A$ having large overlap with $|\varsigma\rangle$ (assume the orientation of each edge e is from A to B.) Indeed, by (1), $|\phi\rangle$ is orthogonal to all $|\psi_u\rangle$, so it is invariant under the action of both R_A and R_B. Moreover, $\||\phi\rangle\|^2 = C_1 R + \sum_{e\in E} p_e^2/w_e$ and $\langle\phi|\varsigma\rangle = \sqrt{C_1 R}$. Since we assumed $R \geq \sum_{e\in E} p_e^2/w_e$, we get that the normalized vector satisfies

$$\frac{\langle\phi|\varsigma\rangle}{\||\phi\rangle\|} \geq \sqrt{\frac{C_1}{1+C_1}}. \tag{5}$$

Now consider the negative case. Define

$$|w\rangle = \sqrt{C_1 R}\left(\sum_{u\in S} \sqrt{\frac{\sigma_u}{C_1 R}}\,|u\rangle + \sum_{e\in E} \sqrt{w_e}\,|e\rangle\right).$$

Let Π_A and Π_B be the projectors on the invariant subspaces of R_A and R_B, respectively. Since $S \subseteq A$, we have $\Pi_A|w\rangle = 0$ and $\Pi_B|w\rangle = |\varsigma\rangle$. Also

$$\||w\rangle\|^2 = \sum_{u\in S} \sigma_u + C_1 R \sum_{e\in E} w_e = 1 + C_1 RW,$$

hence, by Lemma 1, we have that, if

$$\Theta = \frac{1}{C_2\sqrt{1 + C_1 RW}}$$

for some constant $C_2 > 0$, then the overlap of $|\varsigma\rangle$ with the eigenvectors of $R_B R_A$ with phase less than Θ is at most $1/(2C_2)$. Comparing this with (5), we find that it is sufficient to execute phase estimation with precision Θ if C_1 and C_2 are large enough. Also, assuming $C_1 \geq 1$, we get $\Theta = \Omega(1/\sqrt{RW})$ by (4). □

3 Application to 3-Distinctness

In this section, we describe a quantum algorithm for 3-distinctness having time complexity $\tilde{O}(n^{5/7})$. This is a different algorithm from Ref. [10], and is based on ideas from Ref. [19].

Theorem 4. *The 3-distinctness problem can be solved by a quantum algorithm in time $\tilde{O}(n^{5/7})$ using quantum random access quantum memory (QRAQM) of size $\tilde{O}(n^{5/7})$.*

Recall that Ambainis's algorithm consists of two phases: the setup phase that prepares the uniform superposition, and the quantum walk itself. Our algorithm also consists of these two phases. In our case, the analysis of the quantum walk is quite simple, and can be easily generalized to any k. However, the setup phase is hard to generalize. The case of $k = 3$ has a relatively simple *ad hoc* solution (see the full version of the paper).

Technicalities. We start the section with some notations and algorithmic primitives we need for our algorithm. For more detail on the implementation of these primitives, refer to the paper by Ambainis [2]. Although this paper does not exactly give the primitives we need, it is straightforward to apply the necessary modifications; we omit the details here.

We are given a string $\chi \in [q]^n$. A subset $J \subseteq [n]$ of size ℓ is called an *ℓ-collision* iff $\chi_i = \chi_j$ for all $i, j \in J$. In the k-distinctness problem, the task is to determine whether the given input contains a k-collision. Inputs with a k-collision are called *positive*; the remaining inputs are called *negative*.

Without loss of generality, we may assume that any positive input contains exactly one k-collision [2]. Also, we may assume there are $\Omega(n)$ $(k-1)$-collisions by extending the input with dummy elements.

For a subset $S \subseteq [n]$ and $i \in [k]$, let S_i denote the set of $j \in S$ such that $|\{j' \in S : \chi_{j'} = \chi_j\}| = i$. Denote $r_i = |S_i|/i$, and call $\tau = (r_1, \ldots, r_k)$ the *type* of S.

Our main technical tool is a dynamical quantum data structure that maintains a subset $S \subseteq [n]$ and the values χ_j for $j \in S$. We use notation $|S\rangle_{\mathsf{D}}$ to denote a register containing the data structure for a particular choice of $S \subseteq [n]$.

The data structure can perform several operations in polylogarithmic time. The initial state of the data structure is $|\emptyset\rangle_{\mathsf{D}}$. The update operation adds or

removes an element: $|S\rangle_{\mathsf{D}}|j\rangle|\chi_j\rangle \mapsto |S\triangle\{j\}\rangle_{\mathsf{D}}|j\rangle|0\rangle$, where that $\triangle$ denotes for the symmetric difference. The data structure can perform several query operations. It can give the type τ of S. For integers $i \in [k]$ and $\ell \in [|S_i|]$, it returns the ℓth element of S_i according to some internal ordering. Given an element $j \in [n]$, it detects whether j is in S, and if it is, returns the pair (i, ℓ) such that j is the ℓth element of S_i. Given $a \in [q]$, it returns $i \in [k]$ such that a is a value in S_i or says there is no such i.

The data structure is coherence-friendly, i.e., a subset S has the same representation $|S\rangle_{\mathsf{D}}$ independent of the sequence of update operations that results in this subset. It has an exponentially small error probability of failing that can be ignored. The data structure can be implemented using quantum random access quantum memory (QRAQM) in the terminology of Ref. [20].

The quantum walk part of the algorithm is given in the following theorem. For the $k = 3$, there exists a setup procedure that prepares the state $|\varsigma\rangle$ defined in the statement of the theorem with $r_1 = r_2 = n^{4/7}$ in time $\tilde{O}(n^{5/7})$. Together with the theorem this gives an $\tilde{O}(n^{5/7})$-time quantum algorithm for 3-distinctness.

Theorem 5. *Let $r_1, \ldots, r_{k-1} = o(n)$ be positive integers, let $\chi \in [q]^n$ be an input for the k-distinctness problem, and let V_0 be the set of all $S \subseteq [n]$ having type $(r_1, \ldots, r_{k-1}, 0)$. Given the uniform superposition $|\varsigma\rangle = \frac{1}{\sqrt{|V_0|}} \sum_{S \in V_0} |S\rangle$, the k-distinctness problem can be solved in $\tilde{O}(n/\sqrt{\min\{r_1, \ldots, r_{k-1}\}})$ quantum time.*

Proof. We may assume that any input contains at most one k-collision and $\Omega(n)$ $(k-1)$-collisions. Define $r_k = 0$, and the type τ_i as $(r_1, \ldots, r_{i-1}, r_i + 1, r_{i+1}, \ldots, r_k)$ for $i \in \{0, 1, \ldots, k\}$. Let V_i be the set of all $S \subseteq [n]$ having type τ_i (consistent with our previous notation for V_0). Denote $V = \bigcup_i V_i$. Also, for $i \in [k]$, define the set Z_i of *dead-ends* consisting of vertices of the form (S, j) for $S \in V_{i-1}$ and $j \in [n]$ such that $S \triangle \{j\} \notin V$. Again, $Z = \bigcup_i Z_i$.

The vertex set of G is $V \cup Z$. Each $S \in V \setminus V_k$ is connected to n vertices, one for each $j \in [n]$: if $S \triangle \{j\} \in V$, the vertex corresponding to j is $S \triangle \{j\}$; otherwise, it is $(S, j) \in Z$. A vertex $S \in V_k$ is connected to k vertices in V_{k-1} differing from S in one element. Each $(S, j) \in Z$ is only connected to S. The weight of each edge is 1. A vertex is marked if and only if it is contained in V_k.

The algorithm of Theorem 3 is not applicable here because we do not know the graph in advance (it depends on the input), nor do we know the amplitudes in the initial state $|\varsigma\rangle$. However, we know the graph locally, and our ignorance in the amplitudes of $|\varsigma\rangle$ conveniently cancels with our ignorance in the size of G.

Let us briefly describe the implementation of the quantum walk on G following Section 2.3. Let $G = (V \cup Z, E)$ be the graph described above. It is bipartite: the part A contains all V_i and Z_i for i even, and B contains all V_i and Z_i for i odd. The support of $|\varsigma\rangle$ is contained in A. The reflections R_A and R_B are the direct sums of local reflections D_u over all u in A and B, respectively. They are as follows:

– If $u \in V_k$, then D_u is the identity in $\mathcal{H}_u$.

– If $u \in Z_i$, then D_u negates the amplitude of the only edge incident to u.
– If $u \in V_i$ for $i < k$, then D_u is the reflection about the orthogonal complement of $|\psi_u\rangle$ in $\mathcal{H}_u$. If $u \in V_0$, or $u \in V_i$ with $i > 0$, then $|\psi_u\rangle$ is defined as

$$|\psi_u\rangle = \frac{1}{\sqrt{C_1}}|u\rangle + \sum_{uv \in E} |uv\rangle, \qquad \text{or} \qquad |\psi_u\rangle = \sum_{uv \in E} |uv\rangle,$$

respectively. Here, C_1 is a constant.

The space of the algorithm consists of three registers: D, C and Z. The data register D contains the data structure for $S \subseteq [n]$. The coin register C contains an integer in $\{0, 1, \ldots, n\}$, and the qubit Z indicates whether the vertex is an element of Z. A combination $|S\rangle_{\mathsf{D}}|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{Z}}$ with $S \in V_0$ indicates a vertex in V_0 that is used in $|\varsigma\rangle$. A combination $|S\rangle_{\mathsf{D}}|j\rangle_{\mathsf{C}}|0\rangle_{\mathsf{Z}}$ with $j > 0$ indicates the edge between S and $S \triangle \{j\}$ or $(S, j) \in Z$. Finally, a combination $|S\rangle_{\mathsf{D}}|j\rangle_{\mathsf{C}}|1\rangle_{\mathsf{Z}}$ indicates the edge between $(S, j) \in Z$ and $S \in V$.

The reflections R_A and R_B are broken down into the *diffuse* and *update* operations. The diffuse operations perform the local reflections in the list above. For the first one, do nothing conditioned on $|S\rangle_{\mathsf{D}}$ being marked. For the second one, negate the phase conditioned on Z containing 1. The third reflection is the Grover diffusion [21] with one special element if $S \in V_0$.

The representation of the edges is asymmetric. One vertex is contained in the D register, and the other is stored jointly by the D and C registers. The update operation changes the representation of the edge to the opposite one.

The update operation can be performed using the primitives from Section 3. Given $|S\rangle_{\mathsf{D}}|j\rangle_{\mathsf{C}}|b\rangle_{\mathsf{Z}}$, calculate whether $S \triangle \{j\} \in V$ in a fresh qubit Y. Conditioned on Y, query the value of χ_j and perform the update operation for the data structure. Conditioned on Y not being set, flip the value of Z. Finally, uncompute the value in Y. In the last step, we use that $|S\rangle_{\mathsf{D}}|j\rangle_{\mathsf{C}}$ represents an edge between vertices in V if and only if $|S \triangle \{j\}\rangle_{\mathsf{D}}|j\rangle_{\mathsf{C}}$ does the same.

Having shown how to implement the step of the quantum walk efficiently, let us estimate the required number of steps. The argument is very similar to the one in Theorem 3. We start with the positive case. Assume $\{a_1, \ldots, a_k\}$ is the unique k-collision. Let V_0' denote the set of $S \in V_0$ that are disjoint from $\{a_1, \ldots, a_k\}$, and σ' be the uniform probability distribution on V_0'. Define the flow p from σ' to V_k as follows. For each $S \in V_i$ such that $i < k$ and $S \cap M = \{a_1, \ldots, a_i\}$, define flow $p_e = 1/|V_0'|$ on the edge e from S to $S \cup \{a_{i+1}\} \in V_{i+1}$. Define $p_e = 0$ for all other edges e. Let

$$|\phi\rangle = \sqrt{C_1} \sum_{S \in V_0'} \frac{1}{|V_0'|}|S\rangle - \sum_{e \in E} p_e |e\rangle.$$

This vector is orthogonal to all ψ_u, so it is invariant under the action of $R_B R_A$. Also, $\||\phi\rangle\|^2 = (k + C_1)/|V_0'|$, and $\langle\phi|\varsigma\rangle = \sqrt{C_1/|V_0|}$. Hence,

$$\left\langle \frac{\phi}{\|\phi\|}, \varsigma \right\rangle = \sqrt{\frac{C_1|V_0'|}{(k + C_1)|V_0|}} \sim \sqrt{\frac{C_1}{k + C_1}}$$

where $\sim$ denotes asymptotic equivalence as $n \to \infty$.

In the negative case, define

$$|w\rangle = \sqrt{\frac{C_1}{|V_0|}}\left(\sum_{S\in V_0}\frac{1}{\sqrt{C_1}}|S\rangle + \sum_{e\in E}|e\rangle\right).$$

Similarly to the proof of Theorem 3, we have $\Pi_A|w\rangle = 0$ and $\Pi_B|w\rangle = |\varsigma\rangle$.

Let us estimate $\||w\rangle\|$. The number of edges in E is at most n times the number of vertices in $V_0 \cup \cdots \cup V_{k-1}$. Thus, we have to estimate $|V_i|$ for $i \in [k-1]$. Consider the relation between V_0 and V_i where $S \in V_0$ and $S' \in V_i$ are in the relation iff $S' \setminus S$ consists of i equal elements. Each element of V_0 has at most $n\binom{k-1}{i} = O(n)$ images in V_i, where the constant behind the big-O depends exponentially on k. This is because there are at most n maximal collisions in the input, and for each of them, there are at most $\binom{k-1}{i}$ variants to extend S with. On the other hand, each element in V_i has exactly $r_i + 1$ preimages in V_0. Thus, $|V_i| = O(n|V_0|/r_i)$, so

$$\||w\rangle\| = O\left(\sqrt{1 + n/r_1 + n/r_2 + \cdots + n/r_{k-1}}\right) = O\left(n/\sqrt{\min\{r_1, \ldots, r_{k-1}\}}\right).$$

By Lemma 1, we have that if $\Theta = \Omega(1/\||w\rangle\|)$, then the overlap of $|\varsigma\rangle$ with the eigenvectors of $R_B R_A$ with phase less than Θ can be made at most $1/C_2$ for any constant $C_2 > 0$. Thus, it suffices to use phase estimation with precision Θ if C_1 and C_2 are large enough. By Theorem 2, this requires $O(n/\sqrt{\min\{r_1, \ldots, r_{k-1}\}})$ iterations of the quantum walk. □

4 Quantum Walks with Nested Updates

4.1 Introduction

Given a Markov chain P with spectral gap δ and success probability ε in its stationary distribution, one can construct a quantum search algorithm with cost $\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})$ [12], where S, U, and C are respectively the setup, update, and checking costs of the quantum analog of P. Using a quantum walk algorithm with costs $\mathsf{S}', \mathsf{U}', \mathsf{C}', \varepsilon', \delta'$ (as in [12]) as a checking subroutine straightforwardly gives complexity $\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{S}' + \frac{1}{\sqrt{\varepsilon'}}(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'))$. Using nested checking [11], the cost can be reduced to $\mathsf{S} + \mathsf{S}' + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \frac{1}{\sqrt{\varepsilon'}}(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'))$.

It is natural to ask if a quantum walk subroutine can be used for the update step in a similar manner to obtain cost $\mathsf{S} + \mathsf{S}' + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\frac{1}{\sqrt{\varepsilon'}}(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}') + \mathsf{C})$. In most applications, the underlying walk is independent of the input, so the update operation is simple, but for some applications a more complex update may be useful (as in [22], where Grover search is used for the update). In Section 4.3, we describe an example showing that it is not even clear how to use a nested quantum walk for the update with the seemingly trivial cost $\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}(\mathsf{S}' + \frac{1}{\sqrt{\varepsilon'}}(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}')) + \mathsf{C})$. Nevertheless, despite the difficulties that arise in implementing nested updates, we show in Section 4.4 how to achieve the more desirable cost expression in certain cases, and a similar one in general.

To accomplish this, we extend the quantum walk search framework by introducing the concept of *coin-dependent data*. This allows us to implement nested updates, with a quantum walk subroutine carrying out the update procedure. Superficially, our modification appears small. Indeed, the proof of the complexity of our framework is nearly the same as that of [12]. However, there are some subtle differences in the implementation of the walk.

As in [11], this concept is simple yet powerful. We demonstrate this by constructing a quantum walk version of the learning graph for 3-Distinctness with matching query complexity (up to poly-logarithmic factors). Because quantum walks are easy to analyze, the time complexity, which matches the query complexity, follows easily.

4.2 Quantum Walk Search

In the rest of the paper, let P be a reversible, ergodic Markov chain on a connected, undirected graph $G = (X, E)$ with stationary distribution π and spectral gap $\delta > 0$. Let $M \subseteq X$ be a set of marked vertices. The Markov chain can be used to detect wether $M = \emptyset$ or $\Pr_{x\sim\pi}(x \in M) \geq \varepsilon$, for some given $\varepsilon > 0$.

Quantizing this algorithm leads to efficient quantum algorithms [12]. The quantization considers P as a walk on directed edges of G. We write $(x, y) \in \vec{E}$ when we consider an edge $\{x, y\} \in E$ with orientation. The notation (x, y) means that the current vertex of the walk is x and the *coin*, indicating the next move, is y. Swapping x and y changes the current vertex to y and the coin to x.

The quantum algorithm may carry some data structure while walking on G; we formalize this as follows. Let $0 \notin X$. Define $D : X \cup \{0\} \to \mathcal{D}$ for some Hilbert space $\mathcal{D}$, with $|D(0)\rangle = |0\rangle$. We associate a cost with each part of the algorithm. The cost can be any measure of complexity such as queries or time.

***Setup cost*:** Let S be the cost of constructing

$$|\pi\rangle = \sum_{x\in X} \sqrt{\pi(x)}\,|x\rangle\,|D(x)\rangle \sum_{y\in X} \sqrt{P(x,y)}\,|y\rangle\,|D(y)\rangle\,.$$

***Update cost*:** Let U be the cost of the Local Diffusion operation, which is controlled on the first two registers and acts as

$$|x\rangle\,|D(x)\rangle\,|0\rangle\,|D(0)\rangle \mapsto |x\rangle\,|D(x)\rangle \sum_{y\in X} \sqrt{P(x,y)}\,|y\rangle\,|D(y)\rangle\,.$$

***Checking cost*:** Let C be the cost of $|x\rangle\,|D(x)\rangle \mapsto \begin{cases} -\,|x\rangle\,|D(x)\rangle & \text{if } x \in M \\ |x\rangle\,|D(x)\rangle & \text{otherwise.} \end{cases}$

Theorem 6 ([12]). *Let P be a reversible, ergodic Markov chain on $G = (X, E)$ with spectral gap $\delta > 0$. Let $M \subseteq X$ be such that $\Pr_{x\sim\pi}(x \in M) \geq \varepsilon$, for some $\varepsilon > 0$, whenever $M \neq \emptyset$. Then there is a quantum algorithm that finds an element of M, if $M \neq \emptyset$, with bounded error and with cost $O(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}))$.*

Furthermore, we can approximately map $|\pi\rangle$ to $|\pi(M)\rangle$, the normalized projection of $|\pi\rangle$ onto $\operatorname{span}\{|x\rangle\,|D(x)\rangle\,|y\rangle\,|D(y)\rangle : x \in M, y \in X\}$, in cost $\frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})$.

4.3 Motivating Example

3-*Distinctness.* Suppose the input is a sequence $\chi = \chi_1, \ldots, \chi_n$ of integers from $[q] := \{1, \ldots, q\}$. We model the input as an oracle whose answer to query $i \in [n]$ is χ_i. As in Section 3, we assume without loss of generality that there is at most one 3-collision and that the number of 2-collisions is in $\Theta(n)$. Note that any two 2-collisions not both part of the 3-collision are disjoint.

Quantum Walk for Element Distinctness. In [2], a quantum walk for solving Element Distinctness was presented. This walk takes place on a Johnson graph, $J(n, r)$, whose vertices are subsets of $[n]$ of size r, denoted $\binom{[n]}{r}$. In $J(n, r)$, two vertices S, S' are adjacent if $|S \cap S'| = r - 1$. The data function is $D(S) = \{(i, \chi_i) : i \in S\}$. The diffusion step of this walk acts as

$$|S\rangle |D(S)\rangle |0\rangle \mapsto |S\rangle |D(S)\rangle \tfrac{1}{\sqrt{r(n-r)}} \textstyle\sum_{i \in S, j \in [n] \setminus S} |(S \setminus i) \cup j\rangle |D((S \setminus i) \cup j)\rangle .$$

We can perform this diffusion in two queries by performing the transformation

$$|S\rangle |D(S)\rangle |0\rangle \mapsto |S\rangle |D(S)\rangle \tfrac{1}{\sqrt{r}} \textstyle\sum_{i \in S} |(i, \chi_i)\rangle \tfrac{1}{\sqrt{n-r}} \sum_{j \in [n] \setminus S} |(j, \chi_j)\rangle .$$

We can reversibly map this to the desired state with no queries, and by using an appropriate encoding of D, we can make this time efficient as well.

To complete the description of this algorithm, we describe the marked set and checking procedure. We deviate slightly from the usual quantum walk algorithm of [2] and instead describe a variation that is analogous to the learning graph for Element Distinctness [6]. We say a vertex S is marked if it contains an index i such that there exists $j \in [n] \setminus \{i\}$ with $\chi_i = \chi_j$ (in [2] both i and j must be in S). To check if S is marked, we search $[n] \setminus S$ for such a j, in cost $O(\sqrt{n})$.

Attempting a Quantum Walk for 3-*Distinctness.* We now attempt to construct an analogous algorithm for 3-Distinctness. Let $\mathcal{P}$ denote the set of collision pairs in the input, and $n_2 := |\mathcal{P}|$. We walk on $J(n_2, s_2)$, with each vertex S_2 corresponding to a set of s_2 collision pairs. The diffusion for this walk is the map $|S_2, D(S_2)\rangle |0\rangle \mapsto |S_2, D(S_2)\rangle \frac{1}{\sqrt{r(n_2 - s_2)}} \sum_{\substack{(i,i') \in S_2 \\ (j,j') \in \mathcal{P} \setminus S_2}} |(S_2 \setminus (i, i')) \cup (j, j')\rangle |D((S_2 \setminus (i, i')) \cup (j, j'))\rangle .$

To accomplish this, we need to generate $\frac{1}{\sqrt{s_2}} \sum_{(i,i') \in S_2} |(i, i', \chi_i)\rangle$ and $\frac{1}{\sqrt{n_2 - s_2}} \sum_{(j,j') \in \mathcal{P} \setminus S_2} |(j, j', \chi_j)\rangle$. The first superposition is easy to generate since we have S_2; the second is more difficult since we must find new collisions.

The obvious approach is to use the quantum walk algorithm for Element Distinctness as a subroutine. However, this algorithm does not return the desired superposition over collisions; rather, it returns a superposition over sets that contain a collision. That is, we have the state $\frac{1}{\sqrt{n_2}} \sum_{(i,i') \in \mathcal{P}} |(i, i', \chi_i)\rangle |\psi(i, i')\rangle$ for some garbage $|\psi(i, i')\rangle$. The garbage may be only slightly entangled with (i, i'), but even this small amount of error in the state is prohibitive. Since we must call the update subroutine many times, we need the error to be very small.

Unlike for nested checking, where bounded-error subroutines are sufficient, we cannot amplify the success probability of an update operator. We cannot directly use the state returned by the Element Distinctness algorithm for several reasons. First, we cannot append garbage each time we update, as this would prevent proper interference in the walk. Second, when we use a nested walk for the update step, we would like to use the same trick as in nested checking: putting a copy of the starting state for the nested walk in the data structure so that we only need to perform the inner setup once. To do this here we would need to preserve the inner walk starting state; in other words, the update would need to output some state close to $\binom{n}{s_1}^{-1/2} \sum_{S_1 \in \binom{[n]}{s_1}} |S_1\rangle$. While we might try to recycle the garbage to produce this state, it is unclear how to extract the part we need for the update coherently, let alone without damaging the rest of the state.

This appears to be a problem for any approach that directly uses a quantum walk for the update, since all known quantum walks use some variant of a Johnson graph. Our modified framework circumvents this issue by allowing us to do the update with some garbage, which we then uncompute. This lets us use a quantum walk subroutine, with setup performed only at the beginning of the algorithm, to accomplish the update step. More generally, using our modified framework, we can tolerate updates that have garbage for any reason, whether the garbage is the result of the update being implemented by a quantum walk, or by some other quantum subroutine.

4.4 Quantum Walks with Nested Updates

Coin-Dependent Data. A quantum analog of a discrete-time random walk on a graph can be constructed as a unitary process on the directed edges. For an edge $\{x, y\}$, we may have a state $|x\rangle |y\rangle$, where $|x\rangle$ represents the current vertex and $|y\rangle$ represents the *coin* or next vertex. In the framework of [12], some data function on the vertices is employed to help implement the search algorithm. We modify the quantum walk framework to allow this data to depend on both the current vertex and the coin, so that it is a function of the directed edges, which seems natural in hindsight. We show that this point of view has algorithmic applications. In particular, this modification enables efficient nested updates.

Let $0 \notin X$. Let $D : (X \times \{0\}) \cup \vec{E} \to \mathcal{D}$ for some Hilbert space $\mathcal{D}$. A quantum analog of P with coin-dependent data structures can be implemented using three operations, as in [12], but the update now has three parts. The first corresponds to Local Diffusion from the framework of [12], as described in Section 4.2. The others are needed because of the new coin-dependent data.

Update cost: Let U be the cost of implementing

- Local Diffusion: $|x, 0\rangle |D(x, 0)\rangle \mapsto \sum_{y \in X} \sqrt{P(x, y)} |x, y\rangle |D(x, y)\rangle \ \forall x \in X$;
- The $(X, 0)$-Phase Flip: $|x, 0\rangle |D(x, 0)\rangle \mapsto -|x, 0\rangle |D(x, 0)\rangle \ \forall x \in X$, and the identity on the orthogonal subspace; and
- The Database Swap: $|x, y\rangle |D(x, y)\rangle \mapsto |y, x\rangle |D(y, x)\rangle \ \forall (x, y) \in \vec{E}$.

By cost, we mean any desired measure of complexity such as queries, time, or space. We also naturally extend the setup and checking costs as follows, where $M \subseteq X$ is a set of marked vertices.

Setup cost: Let S be the cost of constructing

$$|\pi\rangle := \sum_{x \in X} \sqrt{\pi(x)} \sum_{y \in X} \sqrt{P(x,y)}\, |x,y\rangle\, |D(x,y)\rangle\,.$$

Checking cost: Let C be the cost of the reflection

$$|x,y\rangle\, |D(x,y)\rangle \mapsto \begin{cases} -\,|x,y\rangle\, |D(x,y)\rangle & \text{if } x \in M, \\ |x,y\rangle\, |D(x,y)\rangle & \text{otherwise,} \end{cases} \quad \forall\, (x,y) \in \vec{E}.$$

Observe that $|\pi\rangle^0 := \sum_{x \in X} \sqrt{\pi(x)}\, |x,0\rangle\, |D(x,0)\rangle$ can be mapped to $|\pi\rangle$ by the Local Diffusion, which has cost $\mathsf{U} < \mathsf{S}$, so we can also consider S to be the cost of constructing $|\pi\rangle^0$.

Theorem 7. *Let P be a Markov chain on $G = (X, E)$ with spectral gap $\delta > 0$, and let D be a* coin-dependent *data structure for P. Let $M \subseteq X$ satisfy $\Pr_{x \sim \pi}(x \in M) \geq \varepsilon > 0$ whenever $M \neq \emptyset$. Then there is a quantum algorithm that finds an element of M, if $M \neq \emptyset$, with bounded error and with cost $O(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}))$.*

Proof. Our quantum walk algorithm is nearly identical to that of [12], so the proof of this theorem is also very similar. Just as in [12], we define a walk operator, $W(P)$, and analyze its spectral properties. Let $\mathcal{A} := \mathrm{span}\{\sum_{y \in X} \sqrt{P(x,y)}\, |x,y\rangle\, |D(x,y)\rangle : x \in X\}$ and define $W(P) := ((\textsc{Database Swap}) \cdot \mathrm{ref}_{\mathcal{A}})^2$, where $\mathrm{ref}_{\mathcal{A}}$ denotes the reflection about $\mathcal{A}$.

As in [12], we can define $\mathcal{H} := \mathrm{span}\{|x,y\rangle : (x,y) \in (X \times \{0\}) \cup \vec{E}\}$ and $\mathcal{H}_D := \mathrm{span}\{|x,y,D(x,y)\rangle : (x,y) \in (X \times \{0\}) \cup \vec{E}\}$. Also as in [12], there is a natural isomorphism $|x,y\rangle \mapsto |x,y\rangle_D = |x,y,D(x,y)\rangle$, and $\mathcal{H}_D$ is invariant under both $W(P)$ and the checking operation. Thus, the spectral analysis may be done in $\mathcal{H}$, on states without data, *exactly* as in [12]. However, there are some slight differences in how we implement $W(P)$, which we now discuss.

The first difference is easy to see: in [12], the Database Swap can be accomplished trivially by a SWAP operation, mapping $|x\rangle\, |y\rangle\, |D(x)\rangle\, |D(y)\rangle$ to $|y\rangle\, |x\rangle\, |D(y)\rangle\, |D(x)\rangle$, whereas in our case, there may be a nontrivial cost associated with the mapping $|D(x,y)\rangle \mapsto |D(y,x)\rangle$, which we must include in the calculation of the update cost.

The second difference is more subtle. In [12], $\mathrm{ref}_{\mathcal{A}}$ is implemented by applying $(\textsc{Local Diffusion})^\dagger$, reflecting about $|0, D(0)\rangle$ (since the data only refers to a vertex) in the coin register, and then applying (Local Diffusion). It is simple to reflect about $|0, D(0)\rangle$, since $|D(0)\rangle = |0\rangle$ in the formalism of [12]. In [12], this reflection is sufficient, because the operation $(\textsc{Local Diffusion})^\dagger$ fixes the vertex and its data, $|x\rangle\, |D(x)\rangle$, so in particular, it is still in the space $\mathrm{span}\{|x\rangle\, |D(x)\rangle : x \in X\}$. The register containing the coin and its data,

$|y\rangle\,|D(y)\rangle$, may be moved out of this space by (Local Diffusion)†, so we must reflect about $|0\rangle\,|D(0)\rangle$, but this is straightforward.

With coin-dependent data, a single register $|D(x,0)\rangle$ holds the data for both the vertex and its coin, and the operation (Local Diffusion)† may take the coin as well as the entire data register out of the space $\mathcal{H}_D$, so we need to reflect about $|0\rangle\,|D(x,0)\rangle$, which is not necessarily defined to be $|0\rangle\,|0\rangle$. This explains why the cost of $(X,0)$-Phase Flip is also part of the update cost. In summary, we implement $W(P)$ by $((\text{Database Swap})\cdot(\text{Local Diffusion})\cdot((X,0)\text{-Phase Flip})\cdot(\text{Local Diffusion})^\dagger)^2$. □

Nested Updates. We show how to implement efficient nested updates using the coin-dependent data framework. Let $C : X\cup\{0\}\to\mathcal{C}$ be some coin-independent data structure (that will be a part of the final data structure) with $|C(0)\rangle = |0\rangle$, where we can reflect about $\mathrm{span}\{|x\rangle\,|C(x)\rangle : x\in M\}$ in cost C_C.

Fix $x\in X$. Let P^x be a walk on a graph $G^x=(V^x,E^x)$ with stationary distribution π^x and marked set $M^x\subset V^x$. We use this walk to perform Local Diffusion over $|x\rangle$. Let d^x be the data for this walk.

When there is ambiguity, we specify the data structure with a subscript. For instance, $|\pi\rangle_D = \sum_{x,y\in X}\sqrt{\pi(x)P(x,y)}\,|x,y\rangle\,|D(x,y)\rangle$ and $|\pi\rangle_C^0 = \sum_{x\in X}\sqrt{\pi(x)}\,|x,0\rangle\,|C(x),0\rangle$. Similarly, S_C is the cost to construct the state $|\pi\rangle_C$.

Definition 1. *The family* $(P^x,M^x,d^x)_{x\in X}$ implements the Local Diffusion and Database Swap of (P,C) with cost T *if the following two maps can be implemented with cost* T*:*

Local Diffusion with Garbage*: For some garbage states* $(|\psi(x,y)\rangle)_{(x,y)\in\vec{E}}$*, an operation controlled on the vertex* x *and* $C(x)$*, acting as*

$$|x,0\rangle\,|C(x),0\rangle\,|\pi^x(M^x)\rangle_{d^x}\mapsto\sum_{y\in X}\sqrt{P(x,y)}\,|x,y\rangle\,|C(x),C(y)\rangle\,|\psi(x,y)\rangle\,;$$

Garbage Swap*: For any edge* $(x,y)\in\vec{E}$*,*

$$|x,y\rangle\,|C(x),C(y)\rangle\,|\psi(x,y)\rangle\mapsto|y,x\rangle\,|C(y),C(x)\rangle\,|\psi(y,x)\rangle\,.$$

The data structure of the implementation *is* $|D(x,0)\rangle = |C(x),0\rangle\,|\pi^x(M^x)\rangle_{d^x}$ *for all* $x\in X$ *and* $|D(x,y)\rangle = |C(x),C(y)\rangle\,|\psi(x,y)\rangle$ *for any edge* $(x,y)\in\vec{E}$*.*

Theorem 8. *Let* P *be a reversible, ergodic Markov chain on* $G=(X,E)$ *with spectral gap* $\delta>0$*, and let* C *be a data structure for* P*. Let* $M\subseteq X$ *be such that* $\Pr_{x\sim\pi}(x\in M)\geq\varepsilon$ *for some* $\varepsilon>0$ *whenever* $M\neq\emptyset$*. Let* $(P^x,M^x,d^x)_{x\in X}$ *be a family implementing the* Local Diffusion *and* Database Swap *of* (P,C) *with cost* T*, and let* $\mathsf{S}',\mathsf{U}',\mathsf{C}',1/\varepsilon',1/\delta'$ *be upper bounds on the costs and parameters associated with each of the* (P^x,M^x,d^x)*. Then there is a quantum algorithm that finds an element of* M*, if* $M\neq\emptyset$*, with bounded error and with cost*

$$\tilde{O}\left(\mathsf{S}_C+\mathsf{S}'+\tfrac{1}{\sqrt{\varepsilon}}\left(\tfrac{1}{\sqrt{\delta}}\left(\tfrac{1}{\sqrt{\varepsilon'}}\left(\tfrac{1}{\sqrt{\delta'}}\mathsf{U}'+\mathsf{C}'\right)+\mathsf{T}\right)+\mathsf{C}_C\right)\right).$$

Proof. We achieve this upper bound using the quantization of P with the data structure of the implementation, D. We must compute the cost of the setup, update, and checking operations associated with this walk.

Checking: The checking cost $\mathsf{C} = \mathsf{C}_D$ is the cost to reflect about $\text{span}\{|x\rangle|y\rangle|D(x,y)\rangle : x \in M\} = \text{span}\{|x\rangle|y\rangle|C(x),C(y)\rangle|\psi(x,y)\rangle : x \in M\}$. We can implement this in $\mathcal{H}_D$ by reflecting about $\text{span}\{|x\rangle|C(x)\rangle : x \in M\}$, which costs C_C.

Setup: Recall that $|C(0)\rangle = |0\rangle$. The setup cost $\mathsf{S} = \mathsf{S}_D$ is the cost of constructing the state $\sum_{x\in X}\sqrt{\pi(x)}|x\rangle|0\rangle|D(x,0)\rangle = \sum_{x\in X}\sqrt{\pi(x)}|x\rangle|0\rangle|C(x),0\rangle|\pi^x(M^x)\rangle$. We do this as follows. We first construct $\sum_{x\in X}\sqrt{\pi(x)}|x,0\rangle|C(x),0\rangle$ in cost S_C. Next, we apply the mapping $|x\rangle \mapsto |x\rangle|\pi^x\rangle$ in cost S'. Finally, we use the quantization of P^x to perform the mapping $|x\rangle|\pi^x\rangle \mapsto |x\rangle|\pi^x(M^x)\rangle$ in cost $\frac{1}{\sqrt{\varepsilon'}}(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}')$. The full setup cost is then $\mathsf{S} = \mathsf{S}_C + \mathsf{S}' + \frac{1}{\sqrt{\varepsilon'}}(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}')$.

Update: The update cost has three contributions. The first is the Local Diffusion operation, which, by the definition of D, is exactly the Local Diffusion with Garbage operation. Similarly, the Database Swap is exactly the Garbage Swap, so these two operations have total cost T. The $(X,0)$-Phase Flip is simply a reflection about states of the form $|x\rangle|D(x,0)\rangle = |x\rangle|C(x)\rangle|\pi^x(M^x)\rangle$. Given any $x \in X$, we can reflect about $|\pi^x(M^x)\rangle$ using the quantization of P^x in cost $\frac{1}{\sqrt{\varepsilon'}}(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}')$ by running the algorithm of Theorem 7. In particular, we can run the walk backward to prepare the state $|\pi^x\rangle$, perform phase estimation on the walk operator to implement the reflection about this state, and then run the walk forward to recover $|\pi^x(M^x)\rangle$. However, this transformation is implemented approximately. To keep the overall error small, we need an accuracy of $O(1/\sqrt{\varepsilon\delta\varepsilon'\delta'})$, which leads to an overhead logarithmic in the required accuracy. The reflection about $|\pi^x(M^x)\rangle$, controlled on $|x\rangle$, is sufficient because Local Diffusion with Garbage is controlled on $|x\rangle|C(x)\rangle$, and so it leaves these registers unchanged. Since we apply the $(X,0)$-Phase Flip just after applying $(\text{Local Diffusion})^\dagger$ (see proof of Theorem 7) to a state in $\mathcal{H}_D$, we can guarantee that these registers contain $|x\rangle|C(x)\rangle$ for some $x \in X$. The total update cost (up to log factors) is $\mathsf{U} = \mathsf{T} + \frac{1}{\sqrt{\varepsilon'}}(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}')$.

Finally, the full cost of the quantization of P (up to log factors) is

$$\begin{aligned}&\mathsf{S}_C + \mathsf{S}' + \tfrac{1}{\sqrt{\varepsilon'}}\left(\tfrac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right) + \tfrac{1}{\sqrt{\varepsilon}}\left(\tfrac{1}{\sqrt{\delta}}\left(\tfrac{1}{\sqrt{\varepsilon'}}\left(\tfrac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right) + \mathsf{T}\right) + \mathsf{C}_C\right)\\ &= \tilde{O}\left(\mathsf{S}_C + \mathsf{S}' + \tfrac{1}{\sqrt{\varepsilon}}\left(\tfrac{1}{\sqrt{\delta}}\left(\tfrac{1}{\sqrt{\varepsilon'}}\left(\tfrac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right) + \mathsf{T}\right) + \mathsf{C}_C\right)\right).\end{aligned}$$ □

If $\mathsf{T} = 0$ (as when, e.g., the notion of cost is query complexity), then the expression is exactly what we would have liked for nested updates.

5 Application to 3-Distinctness

In this section we sketch an alternate proof of Theorem 4, giving a high-level description of the quantum walk algorithm, before summarizing the cost of each required procedure. First we define some notation.

We partition the input space into three disjoint sets A_1, A_2, A_3 of equal size, and assume that if there is a 3-collision $\{i, j, k\}$, then we have $i \in A_1$, $j \in A_2$ and $k \in A_3$. This assumption holds with constant probability, so we need only repeat the algorithm $O(1)$ times with independent choices of the tripartition to find any 3-collision with high probability. Thus, we assume we have such a partition.

For any set $S_1 \subseteq A_1 \cup A_2$, let $\mathcal{P}(S_1) := \{(i,j) \in A_1 \times A_2 : i, j \in S_1, i \neq j, \chi_i = \chi_j\}$ be the set of 2-collisions in S_1 and for any set $S_2 \subset A_1 \times A_2$, let $\mathcal{I}(S_2) := \bigcup_{(i,j)\in S_2}\{i,j\}$ be the set of indices that are part of pairs in S_2. In general, we only consider 2-collisions in $A_1 \times A_2$; other 2-collisions in χ are ignored. For any pair of sets A, B, let $\mathcal{P}(A,B) := \{(i,j) \in A \times B : i \neq j, \chi_i = \chi_j\}$ be the set of 2-collisions between A and B. For convenience, we define $\mathcal{P} := \mathcal{P}(A_1, A_2)$. Let $n_2 := |\mathcal{P}|$. For any set $S_2 \subseteq \mathcal{P}$, we denote the set of queried values by $Q(S_2) := \{(i, j, \chi_i) : (i,j) \in S_2\}$. Similarly, for any set $S_1 \subset [n]$, we denote the set of queried values by $Q(S_1) := \{(i, \chi_i) : i \in S_1\}$.

The Walk. Our overall strategy is to find a 2-collision $(i,j) \in A_1 \times A_2$ such that $\exists k \in A_3$ with $\{i,j,k\}$ a 3-collision. Let $s_1, s_2 < n$ be parameters to be optimized. We walk on the vertices $X = \binom{\mathcal{P}}{s_2}$, with each vertex corresponding to a set of s_2 2-collisions from $A_1 \times A_2$. A vertex is considered marked if it contains (i,j) such that $\exists k \in A_3$ with $\{i,j,k\}$ a 3-collision. Thus, if $M \neq \emptyset$, the proportion of marked vertices is $\varepsilon = \Omega(\frac{s_2}{n_2})$.

To perform an update, we use an Element Distinctness subroutine that walks on s_1-sized subsets of $A_1 \cup A_2$. However, since n_2 is large by assumption, the expected number of collisions in a set of size s_1 is large if $s_1 \gg \sqrt{n}$, which we suppose holds. It would be a waste to take only one and leave the rest, so we replace multiple elements of S_2 in each step. This motivates using a generalized Johnson graph $J(n_2, s_2, m)$ for the main walk, where we set $m := \frac{s_1^2 n_2}{n^2} = O(\frac{s_1^2}{n})$, the expected number of 2-collisions in a set of size s_1. In $J(n_2, s_2, m)$, two vertices S_2 and S_2' are adjacent if $|S_2 \cap S_2'| = s_2 - m$, so we can move from S_2 to S_2' by replacing m elements of S_2 by m distinct elements. Let $\Gamma(S_2)$ denote the set of vertices adjacent to S_2. The spectral gap of $J(n_2, s_2, m)$ is $\delta = \Omega(\frac{m}{s_2})$.

The Update. To perform an update step on the vertex S_2, we use the Element Distinctness algorithm of [2] as a subroutine, with some difference in how we define the marked set. Specifically, we use the subroutine to look for m 2-collisions, with $m \gg 1$. Furthermore, we only want to find 2-collisions that are not already in S_2, so P^{S_2} is a walk on $J(2n/3 - 2s_2, s_1)$, with vertices corresponding to sets of s_1 indices from $(A_1 \cup A_2) \setminus \mathcal{I}(S_2)$, and we consider a vertex marked if it contains at least m pairs of indices that are 2-collisions (i.e., $M^{S_2} = \{S_1 \in \binom{(A_1 \cup A_2)\setminus \mathcal{I}(S_2)}{s_1} : |\mathcal{P}(S_1)| \geq m\}$).

The Data. We store the value χ_i with each $(i,j) \in S_2$ and $i \in S_1$, i.e., $|C(S_2)\rangle = |Q(S_2)\rangle$ and $\left|d^{S_2}(S_1, S_1')\right\rangle = |Q(S_1), Q(S_1')\rangle$. As in Section 3, we use the data structure of [2]. Although technically this is part of the data, it is classical and coin-independent, so it is straightforward. Furthermore, since S_1 is encoded in $Q(S_1)$ and S_2 in $Q(S_2)$, we simply write $|Q(S_1)\rangle$ instead of $|S_1, Q(S_1)\rangle$ and $|Q(S_2)\rangle$ instead of $|S_2, Q(S_2)\rangle$.

The rest of the data is what is actually interesting. We use the state $\left|\pi^{S_2}(M^{S_2})\right\rangle^0_{d^{S_2}}$ in the following instead of $\left|\pi^{S_2}(M^{S_2})\right\rangle_{d^{S_2}}$ since it is easy to map between these two states. For every $S_2 \in X$, let

$$|D(S_2,0)\rangle := |Q(S_2),0\rangle \left|\pi^{S_2}(M^{S_2})\right\rangle^0_{d^{S_2}} = |Q(S_2)\rangle \frac{1}{\sqrt{|M^{S_2}|}} \sum_{S_1 \in M^{S_2}} |Q(S_1)\rangle\,,$$

and for every edge (S_2, S_2'), let $|D(S_2,S_2')\rangle := |Q(S_2), Q(S_2')\rangle\,|\psi(S_2,S_2')\rangle$ where

$$|\psi(S_2,S_2')\rangle := \sum_{\tilde{S}_1 \in \binom{(A_1 \cup A_2)\setminus \mathcal{I}(S_2 \cup S_2')}{s_1 - 2m}} \sqrt{\frac{\binom{n_2 - s_2}{m}}{\binom{|\mathcal{P}(\tilde{S}_1)|+m}{m}\,|M^{S_2}|}} \left|Q(\tilde{S}_1)\right\rangle. \tag{6}$$

We define $|\psi\rangle$ in this way precisely because it is what naturally occurs when we attempt to perform the diffusion.

Summary of Costs. Our setup is similar to that of Section 3. We create a uniform superposition of sets of s_1 queried indices from A_1 in cost $\tilde{O}(s_1)$, search for s_2 elements of A_2 that collide with the queried indices in cost $\tilde{O}\left(s_2\sqrt{n/s_1}\right)$, and measure those queried indices for which we did not find a collision. We add the measured indices to A_3. This leaves a uniform superposition of sets of s_2 2-collisions in $A_1 \times A_2$. We create a uniform superposition of sets of s_1 queried indices from A_1 in cost $\tilde{O}(s_1)$, for a total setup cost of $\mathsf{S}_C + \mathsf{S}' = \tilde{O}\left(s_1 + s_2\sqrt{n/s_1}\right)$.

The update walk costs follow from the above discussion, with $\delta' = \Omega(\frac{m}{s_2})$ (the spectral gap of $J(n_2, s_2, m)$); $\varepsilon' = \Omega(1)$ (the proportion of sets of size s_1 containing $\geq m$ 2-collisions); $\mathsf{U}' = \tilde{O}(1)$; and $\mathsf{C}' = O(1)$, achievable by keeping a count of the number of 2-collisions in the set.

It's not difficult to see that our garbage is symmetric, so our garbage swap is quite straightforward and requires simply moving $O(m)$ already queried elements between data structures. Similarly, the local diffusion with garbage is accomplished by moving $O(m)$ already queried 2-collisions between data structures, thus we have $\mathsf{T} = \tilde{O}(m)$. The checking is accomplished by searching A_3 for an element in collision with one of the stored 2-collisions, giving $\mathsf{C} = \tilde{O}(\sqrt{n})$.

Plugging these into the formula of Theorem 8 gives an upper bound of $\tilde{O}(n^{5/7})$ time complexity, using the optimal values of $s_1 = n^{5/7}$ and $s_2 = n^{4/7}$.

References

1. Buhrman, H., Dürr, C., Heiligman, M., Høyer, P., Santha, M., Magniez, F., de Wolf, R.: Quantum algorithms for Element Distinctness. SIAM Journal on Computing 34, 1324–1330 (2005)
2. Ambainis, A.: Quantum walk algorithm for element distinctness. In: 45th IEEE FOCS, pp. 22–31 (2004)
3. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and element distinctness problems. Journal of the ACM 51, 595–605 (2004)
4. Reichardt, B.: Reflections for quantum query algorithms. In: 22nd ACM-SIAM SODA, pp. 560–569 (2011)
5. Lee, T., Mittal, R., Reichardt, B., Spalek, R., Szegedy, M.: Quantum query complexity of state conversion. In: 52nd IEEE FOCS, pp. 344–353 (2011)
6. Belovs, A.: Span programs for functions with constant-sized 1-certificates. In: 44th ACM STOC, pp. 77–84 (2012)
7. Lee, T., Magniez, F., Santha, M.: A learning graph based quantum query algorithm for finding constant-size subgraphs. Chicago Journal of Theoretical Computer Science (2012)
8. Zhu, Y.: Quantum query of subgraph containment with constant-sized certificates. International Journal of Quantum Information 10, 1250019 (2012)
9. Lee, T., Magniez, F., Santha, M.: Improved quantum query algorithms for triangle finding and associativity testing. In: ACM-SIAM SODA, pp. 1486–1502 (2013)
10. Belovs, A.: Learning-graph-based quantum algorithm for k-distinctness. In: 53rd IEEE FOCS, pp. 207–216 (2012)
11. Jeffery, S., Kothari, R., Magniez, F.: Nested quantum walks with quantum data structures. In: 24th ACM-SIAM SODA, pp. 1474–1485 (2013)
12. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. SIAM Journal on Computing 40, 142–164 (2011)
13. Bollobás, B.: Modern graph theory. Graduate Texts in Mathematics, vol. 184. Springer (1998)
14. Chandra, A.K., Raghavan, P., Ruzzo, W.L., Smolensky, R., Tiwari, P.: The electrical resistance of a graph captures its commute and cover times. Computational Complexity 6, 312–340 (1996)
15. Szegedy, M.: Quantum speed-up of Markov chain based algorithms. In: Proc. of 45th IEEE FOCS, pp. 32–41 (2004)
16. Kitaev, A.: Quantum measurements and the abelian stabilizer problem (1995)
17. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum algorithms revisited. Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 454, 339–354 (1998)
18. Ambainis, A., Childs, A.M., Reichardt, B.W., Špalek, R., Zhang, S.: Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. SIAM Journal on Computing 39, 2513–2530 (2010)
19. Belovs, A., Lee, T.: Quantum algorithm for k-distinctness with prior knowledge on the input. Technical Report arXiv:1108.3022, arXiv (2011)
20. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem (2011)
21. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proc. of 28th ACM STOC, pp. 212–219 (1996)
22. Childs, A.M., Kothari, R.: Quantum query complexity of minor-closed graph properties. In: 28th STACS, pp. 661–672 (2011)

An Algebraic Characterization of Testable Boolean CSPs

Arnab Bhattacharyya[1] and Yuichi Yoshida[2,⋆]

[1] DIMACS & Rutgers University
arnabb@dimacs.rutgers.edu
[2] National Institute of Informatics and Preferred Infrastructure, Inc.
yyoshida@nii.ac.jp

Abstract. Given an instance $\mathcal{I}$ of a CSP, a tester for $\mathcal{I}$ distinguishes assignments satisfying $\mathcal{I}$ from those which are far from any assignment satisfying $\mathcal{I}$. The efficiency of a tester is measured by its query complexity, the number of variable assignments queried by the algorithm. In this paper, we characterize the hardness of testing Boolean CSPs in terms of the algebra generated by the relations used to form constraints. In terms of computational complexity, we show that if a non-trivial Boolean CSP is sublinear-query testable (resp., not sublinear-query testable), then the CSP is in NL (resp., P-complete, $\oplus$L-complete or NP-complete) and that if a sublinear-query testable Boolean CSP is constant-query testable (resp., not constant-query testable), then counting the number of solutions of the CSP is in P (resp., #P-complete).

Also, we conjecture that a CSP instance is testable in sublinear time if its Gaifman graph has bounded treewidth. We confirm the conjecture when a near-unanimity operation is a polymorphism of the CSP.

1 Introduction

In *property testing*, we want to decide whether an instance satisfies some particular property or is far from the property. More specifically, an algorithm is called an ε-*tester* for a property if, given an instance, it accepts with probability at least $2/3$ if the instance satisfies the property, and it rejects with probability at least $2/3$ if the instance is ε-far from the property. Here, an instance is called ε-*far* from a property if we must modify an ε-fraction of the instance to make it satisfy the property. The concept of property testing was introduced in [1] and extended to a combinatorial setting in [2]. Since then, many problems have been revealed to be testable in constant time, that is, independent of input size.

In *constraint satisfaction problems* (*CSP*s), we are given a set of variables and a set of constraints imposed on variables. The objective is to find an assignment that satisfies all the constraints. Depending on the relations used to make constraints, CSPs coincide with many fundamental problems such as SAT, graph

⋆ Supported by JSPS Grant-in-Aid for Research Activity Start-up (24800082), MEXT Grant-in-Aid for Scientific Research on Innovative Areas (24106001), and JST, ERATO, Kawarabayashi Large Graph Project.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 123–134, 2013.

coloring and linear equation systems. For example, a graph G is 2-colorable exactly when the CSP instance with variables $\{x_u \mid u \in V(G)\}$ and constraints $(x_u \neq x_v)$ for every $(u, v) \in E(G)$ has a satisfying assignment over $\{0, 1\}$.

In this paper, we are concerned with testing whether a given assignment satisfies a particular CSP instance. That is, for a known instance $\mathcal{I}$ of a CSP, we want to distinguish assignments which satisfy $\mathcal{I}$ from those which are ε-far from satisfying $\mathcal{I}$. In this context, an assignment f on n variables is said to be *ε-far* from satisfying $\mathcal{I}$ if f differs on at least εn variables from any assignment that satisfies $\mathcal{I}$.

The efficiency of a tester for CSP assignments is measured by its *query complexity*, that is, the number of variable assignments queried by the testing algorithm. We investigate the following question:

> Is the query complexity of testing assignments characterized by the types of constraints used to form the CSP instance?

In what follows, instead of saying testing assignments for a CSP instance $\mathcal{I}$, we usually simply say testing $\mathcal{I}$. By query complexity of a class $\mathcal{C}$ of CSPs, we mean the worst case query complexity, over all instances $\mathcal{I}$ in $\mathcal{C}$, of testing $\mathcal{I}$.

The problem of testing Boolean CSPs has been well-studied. In [3], Ben-Sasson, Harsha and Raskhodnikova showed that testing 3-LIN and 3-SAT require $\Omega(n)$ queries, where n is the number of variables. (We use standard nomenclature for CSP classes. For precise definitions, refer to Section 2.) Note that although 3-LIN is in P while 3-SAT is a classic NP-complete problem, they behave similarly in terms of query complexity. In [4], Fischer et al. showed that 2-SAT instances are testable with $O(\sqrt{n})$ queries. Yoshida in [5] initiated a unified theory for testing CSPs by using techniques from universal algebra. Universal algebra is by now a basic tool [6,7] to investigate the computational complexity of CSPs, and Yoshida showed that many of the same ideas used for studying tractability could also be used for testability. Although his main focus was the list homomorphism problem for undirected graphs, one important consequence of [5] is that testing any NP-complete Boolean[1] CSP requires $\Omega(n)$ queries (if each variable is allowed to contribute with a different weight to the distance measure[2]).

Due to the seminal work [8] of Schaefer, it is known that if a Boolean CSP is not NP-complete, then, assuming P $\neq$ NP, it must be polynomial-time solvable and, moreover, must belong to one of the following classes: 0-valid, 1-valid, 2-SAT, Horn SAT, Dual Horn SAT, and system of linear equations. Therefore, given the results of [3,4,5], characterization of sublinear-query testable CSPs requires determining the query complexity of 0-valid, 1-valid, Horn SAT and Dual Horn SAT CSPs. We complete this characterization here.

Among sublinear-query testable properties, there are properties that can be tested with a constant number of queries, independent of the number of variables. Such *strongly testable* properties are of special interest in the area of

[1] In fact, [5] also implies the same result for non-Boolean CSPs if the Dichotomy Conjecture is true.

[2] In this work, we can remove this technical condition and obtain the same result for the standard Hamming distance.

property testing. For example, the characterization of constant-query testable graph [9,10,11] and algebraic [12,13] properties have been major research projects. Here, we give an algebraic characterization of constant-query testable CSPs.

Finally, we ask if instead of restricting the relations that can be used to form constraints of the instance, one can restrict the *structure* of the variable-clause incidence relationship so as to get low query complexity. The analogy is with Grohe's result [14] that asserts that a CSP instance is solvable in polynomial time if and only if it has bounded treewidth structure (under standard assumptions). We conjecture that bounded treewidth structure also implies sublinear-query testability and confirm this conjecture for a large class of CSPs.

1.1 Our Results

To describe our algebraic characterization of sublinear-query and constant-query testable properties, we introduce some notions from universal algebra. A *constraint language* is a finite set of relations on a finite set, in this paper always $\{0,1\}$. Given a constraint language Γ, let $\mathrm{CSP}(\Gamma)$ be the class of CSP instances that can use relations in Γ to make constraints. For example, 2-colorability is equivalent to $\mathrm{CSP}(\{\neq\})$. An n-ary operation $f : \{0,1\}^n \to \{0,1\}$ is said to *preserve* Γ if for every relation R in Γ and for every set of n strings $x_1, \ldots, x_n \in R$, it is the case that f applied component-wise to $x_1, \ldots, x_n$ also produces a string in R. For example, one can check that the 3-ary majority operation preserves $\{\neq\}$. Let the *polymorphisms* of Γ, denoted $\mathrm{Pol}(\Gamma)$, be the set of all operations that preserve Γ. It turns out that if $\mathrm{Pol}(\Gamma_1) = \mathrm{Pol}(\Gamma_2)$, then the query complexity of testing $\mathrm{CSP}(\Gamma_1)$ and of testing $\mathrm{CSP}(\Gamma_2)$ are roughly equal (precise statement in Section 4). Hence, in order to classify $\mathrm{CSP}(\Gamma)$ into whether it is constant-query or sublinear-query testable, it is enough to study $\mathrm{Pol}(\Gamma)$.

For every possible set of operations $\mathrm{Pol}(\Gamma)$, our main result specifies whether it is constant-query testable, sublinear-query testable or whether it requires linear number of queries. The classification is shown in Figure 1. The figure shows the lattice of Boolean clones (which contain all sets generated as $\mathrm{Pol}(\Gamma)$ for some constraint language Γ) ordered according to containment, known as Post's Lattice [15]. The labels used for the clones are standard in the literature; see [16] for definitions. As would be expected, Figure 1 shows that as $\mathrm{Pol}(\Gamma)$ contains less and less polymorphisms, and so as Γ gets more unrestricted, the problem $\mathrm{CSP}(\Gamma)$ gets harder and the query complexity increases. Here are a few sample observations immediate from the figure and the definitions of the clones:

- Testing NAE-SAT, corresponding to the clone N_2, requires $\Omega(n)$ queries.
- Testing Horn SAT, corresponding to the clone E_2, requires $\Omega(n)$ queries.
- Testing monotonicity, corresponding to the clone M_2, is sublinear-query testable.
- Testing if a 2-coloring is proper, corresponding to the clone D_2, is constant-query testable.

We can summarize our classification by its connection to computational complexity, using results from [16,17]. It seems that coincidentally, we can match our classification with some of the complexity-theoretic CSP characterizations.

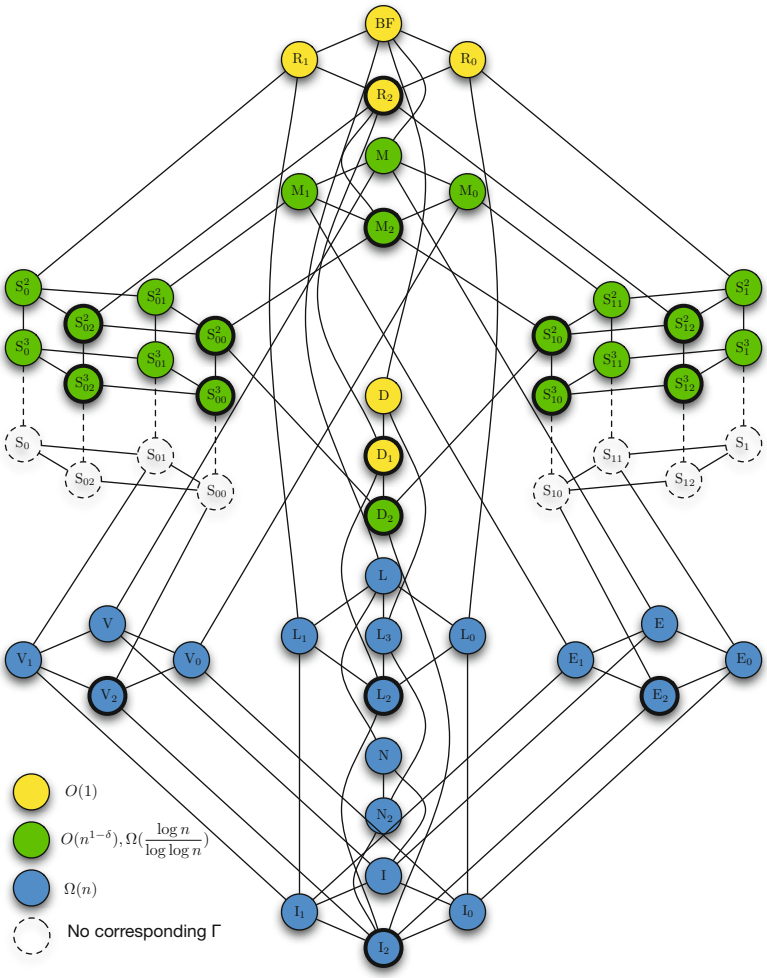


Fig. 1. Post's Lattice and our result. Here, δ is a constant determined by each clone (see Section 4 for the definition). Clones consisting of idempotent operations are circled in bold. Dashed clones cannot be generated as $\mathrm{Pol}(\Gamma)$ for any constraint language Γ.

Theorem 1. *Let* $\mathrm{CSP}(\Gamma)$ *be a non-trivial Boolean CSP. If* $\mathrm{CSP}(\Gamma)$ *is sublinear-query testable (resp., not sublinear-query testable), then* $\mathrm{CSP}(\Gamma)$ *is in* NL *(resp.,* P*-complete,* $\oplus$L*-complete or* NP*-complete). If* $\mathrm{CSP}(\Gamma)$ *is constant-query testable (resp., not constant-query testable but sublinear-query testable), then counting the number of solutions of* $\mathrm{CSP}(\Gamma)$ *is in* P *(resp.,* #P*-complete).*

A *trivial* Boolean CSP is satisfied by the all-zero or all-one assignments.

The classification in Figure 1 is established through a web of reductions between different $\mathrm{CSP}(\Gamma)$ problems. For the characterization of sublinear-query properties, as we mentioned earlier, a central role is played by the query

complexity of Horn k-SAT and Dual Horn k-SAT (corresponding to clones E_2 and V_2), where $k \geqslant 3$ is the arity of each constraint. We show that there exist instances of Horn k-SAT and Dual Horn k-SAT requiring $\Omega(n)$ queries to test. The proof is via an interesting reduction from testing 3-LIN, which uses the fact that the hard instances of 3-LIN in [3] have expanders as their underlying graph.

Another contribution to Figure 1 comes from settling the query complexity of trivial CSPs. We note that they can be hard in terms of testing. We show that the query complexity of any such $\mathrm{CSP}(\Gamma)$ approximately equals that of a related non-trivial CSP obtained by adding constant relations to Γ. Combining this result with bounds obtained in this and previous works allows us to complete Figure 1.

One can think of restrictions on the constraint language Γ as a parameter that controls the query complexity. We next ask if there are structural parameters of CSP instances that also control query complexity. For example, it is easy to see that if all constraints are of bounded arity and on disjoint sets of variables, then one can test with a constant number of queries, no matter what the constraint language is. We conjecture that the same is true if the constraints are structured in a "tree-like" fashion. More precisely, given a CSP instance $\mathcal{I}$, define the *Gaifman graph* $G_{\mathcal{I}}$ to be the graph whose vertex set is the set of variables and whose edge set consists of pairs (u, v) such that u and v appear in the same constraint of $\mathcal{I}$. Then, we conjecture:

Conjecture 1. Given CSP instance $\mathcal{I}$ whose Gaifman graph has treewidth $\leqslant w$, there exists a tester for $\mathcal{I}$ with query complexity $O(f(w)n^c)$ for some function f and $c < 1$.

Indeed, it even seems possible that bounded treewidth instances always have polylogarithmic query complexity. We show logarithmic query complexity for a class of CSPs that contain 2-SAT.

Theorem 2. *Let Γ be a constraint language such that* $\mathrm{Pol}(\Gamma)$ *contains a k-ary near-unanimity operation, and let $\mathcal{I} = (V, \mathcal{C})$ be an instance of* $\mathrm{CSP}(\Gamma)$*. If the treewidth of $G_{\mathcal{I}}$ is w, then we can test $\mathcal{I}$ with query complexity $O(\frac{kw \log n}{\varepsilon})$.*

1.2 Previous Work

The work most relevant to this paper is [5], which studied the query complexity to test List H-homomorphism. For two graphs G and H, a function $f : V(G) \to V(H)$ is called a *homomorphism* from G to H if $(f(u), f(v)) \in E(H)$ whenever $(u, v) \in E(G)$. Testing List H-homomorphism is a problem, in which given a graph G, a list constraint $L : V(G) \to 2^{V(H)}$, and a function f, we want to test whether f is a homomorphism from G to H and $f(v) \in L(v)$ for each $v \in V(G)$. Testing List H-homomorphism is a special case of testing CSPs. Similarly to our classification, [5] showed the following. List H-homomorphism is sublinear-query testable (resp., not testable) if it is in NL (resp., P-complete, $\oplus$L-complete, or NP-complete). If sublinear-query testable List H-homomorphism is constant-query testable (resp., not testable), then counting the number of solutions is in P (resp., #P-complete).

We note that the problem of testing CSP assignments belongs to the massively parametrized model (see [18,19]), where the tester is given free access to part of the input and oracle access to the rest. Here, a CSP instance $\mathcal{I}$ corresponds to the former one and an assignment f corresponds to the latter one.

Organization. In Section 2, we introduce definitions used throughout the paper. Section 3 is devoted to the linear lower bound for testing Horn k-SAT where $k \geqslant 3$. In Section 4, we classify Boolean constraint language Γ with respect to the query complexity to test CSP(Γ). Omitted proofs are given in the full version of the paper. For constraint languages whose clones contain near-unanimity operations, we prove Theorem 2 also in the full version of the paper.

2 Preliminaries

For an integer $k \geqslant 1$, a *k-ary relation* on a domain A is a subset of A^k. A *constraint language* on a domain A is a finite set of relations on A. A *(finite) relational structure*, or simply a *structure* $\mathcal{A} = \langle A; \Gamma \rangle$ consists of a non-empty set A, called the *domain*, and a constraint language Γ on A. For a structure $\mathcal{A} = \langle A; \Gamma \rangle$, we define the problem CSP($\mathcal{A}$) as follows. An instance $\mathcal{I} = (V, \mathcal{C})$ consists of a set of variables V and a set of constraints $\mathcal{C}$. Here, each constraint $C \in \mathcal{C}$ is of the form $(v_1, \ldots, v_k; R)$, where $v_1, \ldots, v_k \in V$ are variables, R is a relation in $\mathcal{A}$ and k is the arity of R. Then, the objective is to find an assignment $f : V \to A$ that satisfies all the constraints, that is $(f(v_1), \ldots, f(v_k)) \in R$ for every constraint $C = (v_1, \ldots, v_k; R) \in \mathcal{C}$. Throughout this paper, we study Boolean CSPs, and so, A is fixed to be $\{0, 1\}$. So, we often write CSP(Γ) instead of CSP($\mathcal{A}$).

Let us formally define the CSPs that most concern us here.

- k-LIN corresponds to CSP(Γ_{LIN}) where $\Gamma_{\mathrm{LIN}} = \{R_0, R_1\}, R_0 = \{(x_1, \ldots, x_k) \mid \sum_{i=1}^{k} x_i = 0 \pmod 2\}$ and $R_1 = \{(x_1, \ldots, x_k) \mid \sum_{i=1}^{k} x_i = 1 \pmod 2\}$.
- k-SAT corresponds to CSP(Γ_{SAT}) where $\Gamma_{\mathrm{SAT}} = \{R_\phi \mid \phi \in \{0,1\}^k\}, R_\phi = \{0,1\}^k \setminus \{\phi\}$.
- Horn k-SAT corresponds to CSP(Γ_{Horn}) where $\Gamma_{\mathrm{Horn}} = \{U, R_{1^k}, R_{1^{k-1}0}\}, U = \{1\}$ and R_{1^k}, $R_{1^{k-1}0}$ as above. Dual Horn k-SAT corresponds to CSP($\Gamma_{\mathrm{DualHorn}}$) where $\Gamma_{\mathrm{DualHorn}} = \{Z, R_{0^k}, R_{0^{k-1}1}\}$, $Z = \{0\}$, and $R_{0^k}, R_{0^{k-1}1}$ as above.
- A CSP is said to be 0-*valid* if for every instance $\mathcal{I}$ of the CSP, the all-zero assignment satisfies $\mathcal{I}$. Similarly, a CSP is said to be 1-*valid* if the all-ones assignment satisfies every instance.

It is known that there is an explicitly known collection $\mathcal{P}$ of CSPs, such that any Boolean CSP(Γ) is log-space reducible to a CSP in $\mathcal{P}$ [16].

Let $\mathcal{I} = (V, \mathcal{C})$ be a CSP instance. For two assignments $f, f' : V \to \{0, 1\}$, we define $\mathrm{dist}(f, f') = \mathbf{Pr}_v[f(v) \neq f'(v)]$, where v is chosen according to the uniform distribution. We define $\mathrm{dist}_{\mathcal{I}}(f)$ as the distance of f from satisfying assignments, that is $\mathrm{dist}_{\mathcal{I}}(f) = \min_{f'} \mathrm{dist}(f, f')$, where f' is over satisfying assignments of $\mathcal{I}$. We say that f is ε-far from satisfying assignments if $\mathrm{dist}_{\mathcal{I}}(f) \geqslant \varepsilon$.

An algorithm is called an $(\varepsilon, \eta_+, \eta_-)$-*tester* for a property P if it accepts an input with probability at least $1-\eta_+$ when it satisfies P, and it rejects an input with probability at least $1-\eta_-$ when it is ε-far from P. An $(\varepsilon, 1/3, 1/3)$-tester is simply referred to as an ε-*tester*. (As long as $\eta_+, \eta_- < \frac{1}{2}$, an $(\varepsilon, \eta_+, \eta_-)$-tester can be converted into an ε-tester by repeating the original tester a constant number of times and taking the majority decision.)

We always use the symbol n (resp., m) to denote the number of variables (resp., constraints) in the instance we are concerned with. For a CSP instance $\mathcal{I}$, an algorithm is called an ε-*tester for* $\mathcal{I}$ if, given an assignment f for $\mathcal{I}$, it ε-tests whether f is a satisfying assignment of $\mathcal{I}$, where farness is measured using the distance function $\mathrm{dist}_{\mathcal{I}}(\cdot)$. Given a structure $\mathcal{A}$, we say that $\mathrm{CSP}(\mathcal{A})$ is *testable with query complexity* $q(n, m, \varepsilon)$ if for every instance $\mathcal{I}$ in $\mathrm{CSP}(\mathcal{A})$, there is an ε-tester for $\mathcal{I}$ making $q(n, m, \varepsilon)$ queries.

3 Lower Bound for Horn k-SAT and Dual Horn k-SAT

In this section, we prove the following theorem.

Theorem 3. *There exist constants $\varepsilon, \delta, \eta \in (0, 1)$ such that for all large enough n, there is a Horn 3-SAT formula $\mathcal{I}_{\mathrm{Horn}}$ on n variables and $O(n)$ constraints such that any adaptive $(\varepsilon, \eta_+, \eta_-)$-test for $\mathcal{I}_{\mathrm{Horn}}$ makes at least δn queries if $\eta_+ + \eta_- < \eta$.*

Note that Theorem 3 also implies a linear lower bound for ε-testing Dual Horn 3-SAT. We can simply negate each literal in the hard Horn 3-SAT formula to obtain the hard Dual Horn 3-SAT formula.

The proof of Theorem 3 is by a reduction from 3-LIN which is known to require $\Omega(n)$ queries. We first revisit the construction of the hard 3-LIN instance $\mathcal{I}_{\mathrm{LIN}}$. Then, we show how to reduce to Horn 3-SAT using the structure of $\mathcal{I}_{\mathrm{LIN}}$.

3.1 Construction of Hard 3-LIN Instance

In [3], Ben-Sasson, Harsha and Raskhodnikova constructed a 3-LIN instance $\mathcal{I}_{\mathrm{LIN}}$ such that any two-sided adaptive tester for $\mathcal{I}_{\mathrm{LIN}}$ requires $\Omega(n)$ queries. The construction proceeded in two steps.

The first step shows the existence of hard k-LIN formulae for sufficiently large k. Call a bipartite multigraph $G = (L, R, E)$ (c, k)-*regular* if the degree of every left vertex $u \in L$ is c and the degree of every right vertex $v \in R$ is k. Every (c, k)-regular graph G describes a k-LIN formula $\psi(G)$: for every right vertex $v \in R$, $\psi(G)$ contains a constraint $\sum_{u \in N(v)} x_u = 0 \pmod 2$ where $N(v)$ is the set of neighbors of v. A *random* (c, k)-*regular LDPC code of length* n is obtained by taking $\psi(G)$ for a random (c, k)-regular graph G with n left vertices. The following was shown in [3]:

Theorem 4 (Theorem 3.7 of [3]). *For any odd integer $c \geqslant 7$ and for $\mu, \varepsilon, \delta, k > 0$ satisfying $\mu \leqslant \frac{1}{100c^2}, \delta < \mu^c, k > \frac{2\mu c^2}{(\mu^c - \delta)^2}, \varepsilon \leqslant \frac{1}{100k^2}$, and for sufficiently large n, it is the case that with high probability for a random (c, k)-regular LDPC code $\psi(G)$ of length n, every adaptive $(\varepsilon, \eta_+, \eta_-)$-test for $\psi(G)$ makes at least δn queries, if $\eta_+ + \eta_- \leqslant 1 - 2\mu$.*

An important fact about random (c,k)-regular graphs that was used in the proof of Theorem 4 and will be useful to us is:

Lemma 1 (Lemma 6.3 of [3]). *For all integers $c \geqslant 7$, $k \geqslant 2$, and sufficiently large n, a random (c,k)-regular graph $G = (L,R,E)$ with n left vertices has the following property with high probability: for every nonempty subset $S \subseteq L$ of left vertices such that $|S| \leqslant \frac{n}{100k^2}$, there exists a right vertex $v \in R$ such that v has exactly one neighbor in S.*

In the second step, testing satisfiability of k-LIN instances is reduced to testing satisfiability of 3-LIN instances. This is done by repeating $\lceil \log(k-2) \rceil$ times a reduction $\mathcal{R}$ from k-LIN instances ψ to $(\lceil k/2 \rceil + 1)$-LIN instances $\mathcal{R}(\psi)$. If ψ is a k-LIN instance with n variables and m linear constraints $A_1, \ldots, A_m$, then $\mathcal{R}(\psi)$ is a $(\lceil k/2 \rceil + 1)$-LIN instance with $n+m$ variables and $2m$ linear constraints $A'_1, A''_1, \ldots, A'_m, A''_m$, where if the constraint A_i is $x_1 + x_2 + \cdots + x_k = 0 \pmod 2$, then the constraints A'_i and A''_i are, respectively:

$$x_1 + \cdots + x_{\lceil k/2 \rceil} + z_i = 0 \pmod 2 \quad \text{and} \quad x_{\lceil k/2 \rceil + 1} + \cdots + x_k + z_i = 0 \pmod 2$$

with z_i being a new variable.

The desired 3-LIN instance $\mathcal{I}_{\mathrm{LIN}}$ is constructed by applying the reduction $\mathcal{R}$ $\lceil \log(k-2) \rceil$ many times on a random (c,k)-regular LDPC code $\psi(G)$ with $c = 7$ and $k = 16c^2(100c^2)^{2c-1}$. If G has n_0 left vertices and m_0 right vertices, then $\mathcal{I}_{\mathrm{LIN}}$ has $m \leqslant 2km_0$ constraints and $n \leqslant n_0 + 2km_0 = (2c+1)n_0$ variables. The instance $\mathcal{I}_{\mathrm{LIN}}$ also has the following property, which is implicit in the proof of Lemma 3.8 in [3] but which will be convenient for us to make explicit.

Lemma 2 (Unique neighbor property). *Suppose $\mathcal{I}_{\mathrm{LIN}}$, an instance of 3-LIN with n variables, is constructed as described above. Then, with high probability, for every nonempty subset S of variables such that $|S| \leqslant \frac{n}{300ck^2}$, there exists a constraint in $\mathcal{I}_{\mathrm{LIN}}$ which involves exactly one variable of S.*

Proof. Suppose ψ is a linear formula with n' variables $x_1, \ldots, x_{n'}$ and m' linear constraints $A_1, \ldots, A_{m'}$ such that for every subset S of variables such that $|S| \leqslant \varepsilon n'$, there exists a constraint in ψ involving exactly one variable of S. Then, we claim that also for $\mathcal{R}(\psi)$, a linear formula on $n'+m'$ variables $x_1, \ldots, x_{n'}, z_1, \ldots, z_{m'}$, it holds that for every nonempty subset T of variables such that $|T| \leqslant \varepsilon n'$, there exists a constraint in $\mathcal{R}(\psi)$ involving exactly one variable of T.

This claim is enough to prove the lemma, because $\mathcal{I}_{\mathrm{LIN}}$ is formed by composing $\mathcal{R}$ several times on a random (c,k)-regular LDPC code $\psi(G)$. If $\mathcal{I}_{\mathrm{LIN}}$ is on n variables, then $\psi(G)$ is on at least $\frac{n}{2c+1} \geqslant \frac{n}{3c}$ variables. For $\psi(G)$, Lemma 1 shows that with high probability, if S is a subset of variables of size at most $\frac{n/3c}{100k^2} = \frac{n}{300ck^2}$, then there is a constraint in $\psi(G)$ which involves exactly one variable of S. The lemma immediately follows from the claim.

It remains to prove the claim. Let T be a subset of the $n' + m'$ variables of $\mathcal{R}(\psi)$ such that $|T| \leqslant \varepsilon n'$. Let $T_0 = T \cap \{x_1, \ldots, x_{n'}\}$, the subset of T which corresponds to variables in the original formula ψ. Of course, $|T_0| \leqslant \varepsilon n'$, and so, there must exist a constraint A_i in ψ containing exactly one variable from

T_0. In $\mathcal{R}(\psi)$, corresponding to A_i, there are two constraints A'_i and A''_i. Call A'_i the constraint which contains exactly one variable from T_0. Then A''_i does not involve any variables from T_0. Now, there are two cases. If T does not contain z_i, then A'_i contains exactly one variable from T. Else, if T does contain z_i, then A''_i contains exactly one variable from T, z_i itself. Therefore, in either case, our claim is proved.

[3] further showed that the reduction $\mathcal{R}$ preserves the query complexity of the instances. Without elaborating on their proof, we state their main result.

Theorem 5 (Theorem 3.1 of [3]). *Suppose $\mathcal{I}_{\text{LIN}}$, an instance of 3-LIN with n variables and $\Theta(n)$ constraints, is constructed as described above. Then, there exist $\varepsilon, \delta, \eta \in (0,1)$ such that with high probability over the construction of $\mathcal{I}_{\text{LIN}}$, any adaptive $(\varepsilon, \eta_+, \eta_-)$-test for $\mathcal{I}_{\text{LIN}}$ makes at least δn queries, if $\eta_+ + \eta_- \leqslant \eta$. In particular, there exists a 3-LIN formula on n variables which requires $\Omega(n)$ queries for testing satisfiability.*

3.2 Reduction to Horn 3-SAT

Let $\mathcal{I}_{\text{LIN}}$ on n variables be defined as above. We now construct an instance of Horn 3-SAT, $\mathcal{I}_{\text{Horn}}$, in the following way. For each variable x_i in $\mathcal{I}_{\text{LIN}}$, we have two variables v_i and v'_i in $\mathcal{I}_{\text{LIN}}$. For each linear constraint $x_i + x_j + x_k = 0 \pmod 2$ in $\mathcal{I}_{\text{LIN}}$, we have 12 Horn constraints in $\mathcal{I}_{\text{Horn}}$:

$$\begin{array}{llll} v_i \wedge v_j \to v'_k & v_i \wedge v_k \to v'_j & v_j \wedge v_k \to v'_i & v'_i \wedge v_j \to v_k \\ v'_i \wedge v_k \to v_j & v'_j \wedge v_k \to v_i & v_i \wedge v'_j \to v_k & v_i \wedge v'_k \to v_j \\ v_j \wedge v'_k \to v_i & v'_i \wedge v'_j \to v'_k & v'_i \wedge v'_k \to v'_j & v'_j \wedge v'_k \to v'_i \end{array}$$

Given an assignment f_{LIN} for $\mathcal{I}_{\text{LIN}}$, let the assignment f_{Horn} for $\mathcal{I}_{\text{Horn}}$ be defined as: $\forall i \in [n], f_{\text{Horn}}(v_i) = f_{\text{LIN}}(x_i), f_{\text{Horn}}(v'_i) = \overline{f_{\text{LIN}}}(x_i)$

Lemma 3. *For every small enough $\varepsilon > 0$ and large enough n:*

(a) If f_{LIN} satisfies $\mathcal{I}_{\text{LIN}}$, then f_{Horn} also satisfies $\mathcal{I}_{\text{Horn}}$.
(b) If f_{LIN} is ε-far from satisfying $\mathcal{I}_{\text{LIN}}$, then f_{Horn} is also ε-far from satisfying $\mathcal{I}_{\text{Horn}}$.

Proof. The first part is immediate. It is easy to check that if f_{LIN} satisfies a constraint in $\mathcal{I}_{\text{LIN}}$, then f_{Horn} also satisfies the corresponding constraints in $\mathcal{I}_{\text{Horn}}$.

To see part (b), assume it is false so that f_{LIN} is ε-far from satisfying $\mathcal{I}_{\text{LIN}}$ but f_{Horn} is ε-close to a satisfying assignment g_{Horn} for $\mathcal{I}_{\text{Horn}}$. Let $S = \{x_i \mid i \in [n] \text{ such that } g_{\text{Horn}}(v_i) = g_{\text{Horn}}(v'_i)\}$. Clearly, $|S| \leqslant 2\varepsilon n$, since $f_{\text{Horn}}(v_i) \neq f_{\text{Horn}}(v'_i)$ for every i.

Also, we can prove $S \neq \emptyset$. Suppose otherwise. Then, define an assignment g_{LIN} for $\mathcal{I}_{\text{LIN}}$ as: $g_{\text{LIN}}(x_i) = g_{\text{Horn}}(v_i)$ for every $i \in [n]$. Note that g_{LIN} is ε-close to f_{LIN}. We now show that g_{LIN} satisfies $\mathcal{I}_{\text{LIN}}$, and so f_{LIN} is ε-close to

$\mathcal{I}_{\mathrm{LIN}}$, a contradiction. Consider a constraint $x_i + x_j + x_k = 0 \pmod 2$ in $\mathcal{I}_{\mathrm{LIN}}$. We know that $g_{\mathrm{Horn}}(v_i) = g_{\mathrm{LIN}}(x_i)$ and, since $|S| = 0$, $g_{\mathrm{Horn}}(v'_i) = \overline{g_{\mathrm{LIN}}}(x_i)$. Since g_{Horn} satisfies $\mathcal{I}_{\mathrm{Horn}}$, the following constraints must be true: $g_{\mathrm{LIN}}(x_i) \wedge g_{\mathrm{LIN}}(x_j) \to \overline{g_{\mathrm{LIN}}}(x_k), \overline{g_{\mathrm{LIN}}}(x_i) \wedge g_{\mathrm{LIN}}(x_j) \to g_{\mathrm{LIN}}(x_k), g_{\mathrm{LIN}}(x_i) \wedge \overline{g_{\mathrm{LIN}}}(x_j) \to g_{\mathrm{LIN}}(x_k), \overline{g_{\mathrm{LIN}}}(x_i) \wedge \overline{g_{\mathrm{LIN}}}(x_j) \to \overline{g_{\mathrm{LIN}}}(x_k)$. It is now easy to check that these constraints hold iff $g_{\mathrm{LIN}}(x_i) + g_{\mathrm{LIN}}(x_j) + g_{\mathrm{LIN}}(x_k) = 0 \pmod 2$.

Therefore, $0 < |S| \leqslant 2\varepsilon n$. If ε is sufficiently small, Lemma 2 shows that there must exist a constraint in $\mathcal{I}_{\mathrm{LIN}}$ that contains exactly one variable of S. On the other hand, we now show that any constraint in $\mathcal{I}_{\mathrm{LIN}}$ containing one variable in S must contain at least one other variable in S, thus causing a contradiction and finishing the proof. Consider a constraint $x_i + x_j + x_k = 0 \pmod 2$ in $\mathcal{I}_{\mathrm{LIN}}$, and suppose $x_i \in S$. There are two cases.

- Suppose $g_{\mathrm{Horn}}(v_i) = g_{\mathrm{Horn}}(v'_i) = 1$. Since g_{Horn} is a satisfying assignment, it must be: $g_{\mathrm{Horn}}(v_j) \to g_{\mathrm{Horn}}(v'_k)$, $g_{\mathrm{Horn}}(v_k) \to g_{\mathrm{Horn}}(v'_j)$, $g_{\mathrm{Horn}}(v_j) \to g_{\mathrm{Horn}}(v_k)$, $g_{\mathrm{Horn}}(v_k) \to g_{\mathrm{Horn}}(v_j)$, $g_{\mathrm{Horn}}(v'_j) \to g_{\mathrm{Horn}}(v_k)$, $g_{\mathrm{Horn}}(v'_k) \to g_{\mathrm{Horn}}(v_j)$, $g_{\mathrm{Horn}}(v'_j) \to g_{\mathrm{Horn}}(v'_k)$, $g_{\mathrm{Horn}}(v'_k) \to g_{\mathrm{Horn}}(v'_j)$. So, $g_{\mathrm{Horn}}(v_j) = g_{\mathrm{Horn}}(v'_j) = g_{\mathrm{Horn}}(v_k) = g_{\mathrm{Horn}}(v'_k)$, and therefore, $x_j, x_k \in S$.
- Suppose $g_{\mathrm{Horn}}(v_i) = g_{\mathrm{Horn}}(v'_i) = 0$. Then, the eight Horn constraints corresponding to $x_i + x_j + x_k = 0$ (that have either v_i or v'_i on the LHS) are vacuously satisfied. The remaining four are satisfied exactly when $g_{\mathrm{Horn}}(v_j) \wedge g_{\mathrm{Horn}}(v_k), g_{\mathrm{Horn}}(v'_j) \wedge g_{\mathrm{Horn}}(v_k), g_{\mathrm{Horn}}(v_j) \wedge g_{\mathrm{Horn}}(v'_k), g_{\mathrm{Horn}}(v'_j) \wedge g_{\mathrm{Horn}}(v'_k)$ are all false. This can only hold when $g_{\mathrm{Horn}}(v_j) = g_{\mathrm{Horn}}(v'_j) = 0$ or $g_{\mathrm{Horn}}(v_k) = g_{\mathrm{Horn}}(v'_k) = 0$. Thus, either x_j or x_k is in S.

4 Classification of Testable Constraint Languages

In this section, we classify (finite) Boolean structures $\mathcal{A} = \langle \{0,1\}; \Gamma \rangle$ into three categories with respect to the query complexity for testing $\mathrm{CSP}(\Gamma)$. Namely, we give necessary and sufficient conditions for each of the following three cases: (i) $\mathrm{CSP}(\Gamma)$ is constant-query testable, (ii) $\mathrm{CSP}(\Gamma)$ is sublinear-query testable but not constant-query testable, and (iii) $\mathrm{CSP}(\Gamma)$ is not sublinear-query testable.

4.1 Universal Algebra Preliminaries

An *n-ary operation* on a set A is a map from A^n to A. An n-ary operation f on A *preserves* the k-ary relation R on A (equivalently, we say that R is *invariant* under f) if the following holds: given any matrix M of size $k \times n$ whose columns are in R, applying f to the rows of M will produce a k-tuple in R. Given a constraint language Γ, let $\mathrm{Pol}(\Gamma)$ denote the set of all operations that preserve all relations in Γ. Now for any Γ, $\mathrm{Pol}(\Gamma)$ forms a *clone*, i.e., a set of operations closed under compositions and containing all the projections (operations of the form $f(x_1, \ldots, x_k) = x_i$). We note that we may need a infinite set Γ of relations to realize a clone as $\mathrm{Pol}(\Gamma)$.

Remarkably, it turns out that there is an explicit description [15] of the Boolean clones. When ordered by inclusion, they form a countable lattice known as *Post's lattice*, shown in Figure 1. In the rest of this section, we will settle the query complexity for the CSPs corresponding to each clone in Post's lattice.

4.2 Reductions between Constraint Languages

Definition 1 ([5]). *Given constraint languages Γ, Γ', a* gap-preserving local reduction from $\mathrm{CSP}(\Gamma')$ to $\mathrm{CSP}(\Gamma)$ *exists if there are functions $t_1(n,m), t_2(n,m)$ and constants c_1, c_2 satisfying the following: given an instance $\mathcal{I}' = (V', \mathcal{C}')$ of $\mathrm{CSP}(\Gamma')$ and an assignment f' for $\mathcal{I}'$, there exist an instance $\mathcal{I} = (V, \mathcal{C})$ of $\mathrm{CSP}(\Gamma)$ and an assignment f for $\mathcal{I}$ such that: (i) $|V| \leqslant t_1(|V'|, |\mathcal{C}'|)$, (ii) $|\mathcal{C}| \leqslant t_2(|V'|, |\mathcal{C}'|)$, (iii) if f' satisfies $\mathcal{I}'$, then f also satisfies $\mathcal{I}$, (iv) if $\mathrm{dist}_{\mathcal{I}'}(f') \geqslant \varepsilon$, then $\mathrm{dist}_{\mathcal{I}}(f) \geqslant c_1\varepsilon$, and (v) we can compute $f(v)$ for any $v \in V$ by querying f' at most c_2 times.*

It is known [7,20] in the context of computational complexity, that if $\mathrm{Pol}(\Gamma')$ contains $\mathrm{Pol}(\Gamma)$, then there is a log-space reduction from deciding $\mathrm{CSP}(\Gamma')$ to deciding $\mathrm{CSP}(\Gamma)$. The same is qualitatively true for query complexity also, where the reduction is gap-preserving. We say that a k-ary relation R has a *redundancy* if there exist $i, j \in [k]$ such that for any $x_1, \ldots, x_k \in R$, $x_i = x_j$. Then we have:

Lemma 4. *Given constraint language Γ, suppose $\mathrm{CSP}(\Gamma)$ is testable with $q(n, m, \varepsilon)$ queries. If Γ' is a constraint language such that no relation in Γ' contains redundancies and $\mathrm{Pol}(\Gamma') \supseteq \mathrm{Pol}(\Gamma)$, then $\mathrm{CSP}(\Gamma')$ is testable with $q(O(n+m), O(m), O(\varepsilon))$ queries.*

Thus, if Γ' is without redundancies, Lemma 4 shows that a lower bound for $\mathrm{CSP}(\Gamma')$ implies a lower bound for $\mathrm{CSP}(\Gamma)$. Next, we show that Γ' can be assumed to have all polymorphisms *idempotent* without much loss of generality.

Given a constraint language Γ, define the *singleton-expansion of* Γ to be $\Gamma' = \Gamma \cup \{\{0\}, \{1\}\}$. $\mathrm{Pol}(\Gamma')$ consists of exactly the idempotent polymorphisms of Γ, meaning polymorphisms f satisfying $f(x, \ldots, x) = x$ for $x \in \{0, 1\}$, since any polymorphism that preserves the relations $\{0\}$ and $\{1\}$ must be idempotent. We show there is a gap-preserving local reduction from $\mathrm{CSP}(\Gamma')$ to $\mathrm{CSP}(\Gamma)$.

Lemma 5. *Given a constraint language Γ, let Γ' be the singleton-expansion of Γ. Assume that $\varepsilon \ll 1/2$. If $\mathrm{CSP}(\Gamma)$ is testable with $q(n, m, \varepsilon)$ queries, then $\mathrm{CSP}(\Gamma')$ is testable with $q(O(n), O(mn), O(\varepsilon))$ queries.*

4.3 Classification of Structures

The following Theorem classifies the clones circled in bold in Figure 1. The rest are handled by applying Lemma 5.

Theorem 6. *Let Γ be a constraint language with only idempotent polymorphisms.*

- *If $\mathrm{Pol}(\Gamma) \in \{D_1, R_2\}$, then $\mathrm{CSP}(\Gamma)$ is testable with $O(1)$ queries.*
- *If $S_{00} \subseteq \mathrm{Pol}(\Gamma) \subseteq S^2_{02}, S_{10} \subseteq \mathrm{Pol}(\Gamma) \subseteq S^2_{12}$ or $\mathrm{Pol}(\Gamma) \in \{D_2, M_2\}$, then testing $\mathrm{CSP}(\Gamma)$ requires $\Omega(\log n / \log\log n)$ queries and is testable with $o(n)$ queries. The lower bound holds even when $m = n^{1+O(1/\log\log n)}$.*
- *If $\mathrm{Pol}(\Gamma) \in \{I_2, E_2, V_2, L_2\}$, then testing $\mathrm{CSP}(\Gamma)$ requires $\Omega(n)$ queries. The lower bound holds even when $m = O(n)$.*

References

1. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM J. on Comput. 25(2), 252–271 (1996)
2. Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connection to learning and approximation. J. ACM 45(4), 653–750 (1998)
3. Ben-Sasson, E., Harsha, P., Raskhodnikova, S.: Some 3CNF properties are hard to test. SIAM J. on Comput. 35(1), 1–21 (2006)
4. Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., Samorodnitsky, A.: Monotonicity testing over general poset domains. In: Proc. 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 474–483 (2002)
5. Yoshida, Y.: Testing list H-homomorphisms. In: Proc. 27th Annual IEEE Conference on Computational Complexity, pp. 85–95 (2012)
6. Bulatov, A.A., Krokhin, A.A., Jeavons, P.G.: Constraint satisfaction problems and finite algebras. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, p. 272. Springer, Heidelberg (2000)
7. Jeavons, P.: On the algebraic structure of combinatorial problems. Theoretical Computer Science 200(1-2), 185–204 (1998)
8. Schaefer, T.: The complexity of satisfiability problems. In: Proc. 10th Annual ACM Symposium on the Theory of Computing, pp. 216–226 (1978)
9. Alon, N., Shapira, A.: A characterization of the (natural) graph properties testable with one-sided error. SIAM J. on Comput. 37(6), 1703–1727 (2008)
10. Alon, N., Fischer, E., Newman, I., Shapira, A.: A combinatorial characterization of the testable graph properties: it's all about regularity. In: Proc. 38th Annual ACM Symposium on Theory of Computing, pp. 251–260 (2006)
11. Borgs, C., Chayes, J.T., Lovász, L., Sós, V.T., Szegedy, B., Vesztergombi, K.: Graph limits and parameter testing. In: Proc. 36th Annual ACM Symposium on the Theory of Computing, pp. 261–270 (2006)
12. Kaufman, T., Sudan, M.: Algebraic property testing: the role of invariance. In: Proc. 40th Annual ACM Symposium on the Theory of Computing, pp. 403–412 (2008)
13. Bhattacharyya, A., Grigorescu, E., Shapira, A.: A unified framework for testing linear-invariant properties. In: Proc. 51st Annual IEEE Symposium on Foundations of Computer Science, pp. 478–487 (2010)
14. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. J. ACM 54(1), 1–24 (2007)
15. Post, E.: The two-valued iterative systems of mathematical logic. Annals of Mathematics Studies, vol. 5. Princeton Univ. Pr. (1941)
16. Allender, E., Bauland, M., Immerman, N., Schnoor, H., Vollmer, H.: The complexity of satisfiability problems: Refining Schaefer's theorem. J. Comp. Sys. Sci. 75(4), 245–254 (2009)
17. Creignou, N., Hermann, M.: Complexity of generalized satisfiability counting problems. Inform. and Comput. 125(1), 1–12 (1996)
18. Chakraborty, S., Fischer, E., Lachish, O., Matsliah, A., Newman, I.: Testing st-connectivity. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 380–394. Springer, Heidelberg (2007)
19. Newman, I.: Property testing of massively parametrized problems - A survey. In: Goldreich, O. (ed.) Property Testing. LNCS, vol. 6390, pp. 142–157. Springer, Heidelberg (2010)
20. Reingold, O.: Undirected connectivity in log-space. J. ACM 55(4) (2008)

Approximation Algorithms for the Joint Replenishment Problem with Deadlines*

Marcin Bienkowski[1], Jaroslaw Byrka[1], Marek Chrobak[2], Neil Dobbs[3], Tomasz Nowicki[3], Maxim Sviridenko[4], Grzegorz Świrszcz[4], and Neal E. Young[2]

[1] Institute of Computer Science, University of Wrocław, Poland
[2] Department of Computer Science, University of California at Riverside, USA
[3] IBM T.J. Watson Research Center, Yorktown Heights, USA
[4] Department of Computer Science, University of Warwick, UK

Abstract. The Joint Replenishment Problem (JRP) is a fundamental optimization problem in supply-chain management, concerned with optimizing the flow of goods over time from a supplier to retailers. Over time, in response to demands at the retailers, the supplier sends shipments, via a warehouse, to the retailers. The objective is to schedule shipments to minimize the sum of shipping costs and retailers' waiting costs.

We study the approximability of JRP with deadlines, where instead of waiting costs the retailers impose strict deadlines. We study the integrality gap of the standard linear-program (LP) relaxation, giving a lower bound of 1.207, and an upper bound and approximation ratio of 1.574. The best previous upper bound and approximation ratio was 1.667; no lower bound was previously published. For the special case when all demand periods are of equal length we give an upper bound of 1.5, a lower bound of 1.2, and show APX-hardness.

Keywords: Joint replenishment problem with deadlines, inventory theory, linear programming, integrality gap, randomized rounding, approximation algorithm.

1 Introduction

The Joint Replenishment Problem with Deadlines (JRP-D) is an optimization problem in inventory theory concerned with optimizing a schedule of shipments of a commodity from a supplier, via a warehouse, to satisfy demands at m retailers (cf. Figure 1). An instance is specified by a tuple $(C, c, \mathcal{D})$ where $C \in \mathbb{Q}$ is the *warehouse ordering cost*, each retailer $\rho \in \{1, 2, \ldots, m\}$ has *retailer ordering cost* $c_\rho \in \mathbb{Q}$, and $\mathcal{D}$ is a set of n *demands*, where each demand is a triple (ρ, r, d), where ρ is a retailer, $r \in \mathbb{N}$ is the demand's release time and $d \in \mathbb{N}$ is its deadline. The interval $[r, d]$ is the demand *period*. Without loss of generality, we assume $r, d \in [2n]$, where $[i]$ denotes $\{1, 2, \ldots, i\}$.

* Full version available at [4]. Research supported by MNiSW grant number N N206 368839 (2010-2013); NSF grants CCF-1217314, CCF-1117954, OISE-1157129; EPSRC grants EP/J021814/1 and EP/D063191/1; FP7 Marie Curie Career Integration Grant; and Royal Society Wolfson Research Merit Award.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 135–147, 2013.

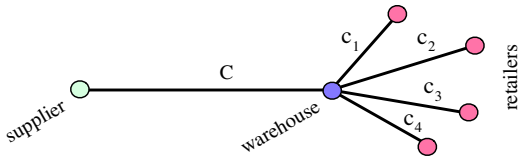


Fig. 1. An instance with four retailers, with ordering costs as distances. The cost of an order is the weight of the subtree connecting the supplier and the involved retailers.

A solution (called a *schedule*) is a set of *orders*, each specified by a pair (t, R), where t is the time of the order and R is a subset of the retailers. An order (t, R) *satisfies* those demands (ρ, r, d) whose retailer is in R and whose demand period contains t (that is, $\rho \in R$ and $t \in [r, d]$). A schedule is *feasible* if all demands are satisfied by some order in the schedule.

The *cost* of order (t, R) is the ordering cost of the warehouse plus the ordering costs of respective retailers, i.e., $C + \sum_{\rho \in R} c_\rho$. It is convenient to think of this order as consisting of a warehouse order of cost C, which is then joined by each retailer $\rho \in R$ at cost c_ρ. The cost of the schedule is the sum of the costs of its orders. The objective is to find a feasible schedule of minimum cost.

Previous Results. The decision variant of JRP-D was shown to be strongly $\mathbb{NP}$-complete by Becchetti et al. [3]. (They considered an equivalent problem of packet aggregation with deadlines on two-level trees.) Nonner and Souza [12] then showed that JRP-D is $\mathbb{APX}$-hard, even if each retailer issues only three demands. Using the primal-dual method, Levi, Roundy and Shmoys [9] gave a 2-approximation algorithm. Using randomized rounding, Levi et al. [10,11] (building on [8]) improved the approximation ratio to 1.8; Nonner and Souza [12] reduced it further to 5/3. These results use a natural linear-program (LP) relaxation, which we use too.

The randomized-rounding approach from [12] uses a natural rounding scheme whose analysis can be reduced to a probabilistic game. For any probability distribution p on $[0, 1]$, the integrality gap of the LP relaxation is at most $1/\mathcal{Z}(p)$, where $\mathcal{Z}(p)$ is a particular statistic of p (see Lemma 1). The challenge in this approach is to find a distribution where $1/\mathcal{Z}(p)$ is small. Nonner and Souza show that there is a distribution p with $1/\mathcal{Z}(p) \le 5/3 \approx 1.67$. As long as the distribution can be sampled from efficiently, the approach yields a polynomial-time $(1/\mathcal{Z}(p))$-approximation algorithm.

Our Contributions. We show that there is a distribution p with $1/\mathcal{Z}(p) \le 1.574$. We present this result in two steps: we show the bound $e/(e-1) \approx 1.58$ with a simple and elegant analysis, then improve it to 1.574 by refining the underlying distribution. We conjecture that this distribution minimizes $1/\mathcal{Z}(p)$. This shows that the integrality gap is at most 1.574 and gives a 1.574-approximation algorithm. We also prove that the LP integrality gap is *at least* 1.207 and we provide a computer-assisted proof that this gap is at least 1.245. (As far as we know, no explicit lower bounds have been previously published.)

For the special case when all demand periods have the same length (as occurs in applications where time-to-delivery is globally standardized) we give an upper bound of 1.5, a lower bound of 1.2, and show $\mathbb{APX}$-hardness.

Related Work. JRP-D is a special case of the Joint Replenishment Problem (JRP). In JRP, instead of having a deadline, each demand is associated with a delay-cost function that specifies the cost for the delay between the times the demand was released and satisfied by an order. JRP is $\mathbb{NP}$-complete, even if the delay cost is linear [2,12]. JRP is in turn a special case of the One-Warehouse Multi-Retailer (OWMR) problem, where the commodities may be stored at the warehouse for a given cost per time unit. The 1.8-approximation by Levi et al. [11] holds also for OWMR. JRP was also studied in the online scenario: a 3-competitive algorithm was given by Buchbinder et al. [6] (see also [5]).

Another generalization of JRP involves a tree-like structure with the supplier in the root and retailers at the leaves, modeling packet aggregation in convergecasting trees. A 2-approximation is known for the variant with deadlines [3]; the case of linear delay costs has also been studied [7].

The LP Relaxation. Here is the standard LP relaxation of the problem. Let $U = \max\{d \mid (\rho, r, d) \in \mathcal{D}\}$ be the maximum deadline, and assume that each release time and deadline is in universe $\mathcal{U} = [U]$.

$$\begin{aligned} \text{minimize} \quad & cost(\boldsymbol{x}) = \textstyle\sum_{t=1}^{U} (C\, x_t + \sum_{\rho=1}^{m} c_\rho\, x_t^\rho) \\ \text{subject to} \quad & x_t \geq x_t^\rho \quad \text{for all } t \in \mathcal{U}, \rho \in \{1, \ldots, m\} \qquad (1) \\ & \textstyle\sum_{t=r}^{d} x_t^\rho \geq 1 \quad \text{for all } (\rho, r, d) \in \mathcal{D} \qquad (2) \\ & x_t, x_t^\rho \geq 0 \quad \text{for all } t \in \mathcal{U}, \rho \in \{1, \ldots, m\}. \end{aligned}$$

We use $\boldsymbol{x}$ to denote an optimal fractional solution to this LP relaxation.

2 Upper Bound of 1.574

The statistic $\mathcal{Z}(p)$. The approximation ratio of algorithm Round_p (defined below) and the integrality gap of the LP are at most $1/\mathcal{Z}(p)$, where $\mathcal{Z}(p)$ is defined by the following so-called *tally game* (following [12]). To begin the game, fix any *threshold* $z \geq 0$, then draw a sequence of independent samples $s_1, s_2, \ldots, s_h$ from p, stopping when their sum exceeds z. Call $z - (s_1 + s_2 + \ldots + s_{h-1})$ the *waste*. Note that, since the waste is less than s_h, it is in $[0, 1)$. Let $\mathcal{W}(p, z)$ denote the expectation of the waste. Abusing notation, let $\mathbf{E}[p]$ denote the expected value of a single sample drawn from p. Then $\mathcal{Z}(p)$ is defined to be the minimum of $\mathbf{E}[p]$ and $1 - \sup_{z \geq 0} \mathcal{W}(p, z)$.

Note that the distribution p that chooses $1/2$ with probability 1 has $\mathcal{Z}(p) = 1/2$. The challenge is to increase $\mathbf{E}[p]$ and reduce the maximum expected waste.

A Generic Randomized-Rounding Algorithm. Next we define the randomized-rounding algorithm Round_p. The algorithm is parameterized by any probability distribution p on $[0, 1]$.

For the rest of this section, fix any instance $(C, c, \mathcal{D})$ and fractional solution $\boldsymbol{x}$ of the LP relaxation. Define $\hat{U} = \sum_{t=1}^{U} x_t$. For each retailer ρ, let ω_ρ denote the piecewise-linear continuous bijection from $[0, \hat{U}]$ to $[0, \sum_{t=1}^{U} x_t^\rho]$ such that $\omega_\rho(0) = 0$, and, for each $T \in [U]$,

$$\omega_\rho(\textstyle\sum_{t=1}^{T} x_t) \;=\; \sum_{t=1}^{T} x_t^\rho.$$

The intuition is that $\omega_\rho(z)$ is the cumulative fractional number of orders joined by retailer ρ by the time the fractional number of warehouse orders reaches z. The equations above determine ω_ρ at its breakpoints; since ω_ρ is piecewise linear and continuous, this determines the entire function. The LP inequalities (1) imply that $0 \le \omega_\rho(z') - \omega_\rho(z) \le z' - z$ when $z' \ge z$. That is, ω_ρ has derivative in $[0, 1]$. Here is the rounding algorithm. Recall that $\hat{U}$ denotes $\sum_{t=1}^{U} x_t$.

Algorithm $\mathsf{Round}_p(C, c_\rho, \mathcal{D}, \boldsymbol{x})$

1: Draw independent random samples $s_1, s_2, \ldots$ from p. Let $g_i = \sum_{h \le i} s_h$.
Set *global cutoff sequence* $\boldsymbol{g} = (g_1, g_2, \ldots, g_I)$, where $I = \min\{i \mid g_i \ge \hat{U} - 1\}$.

2: For each retailer ρ independently, choose ρ's *local cutoff sequence* $\boldsymbol{\ell}^\rho \subseteq \boldsymbol{g}$ greedily to touch all intervals $[a, b]$ with $\omega_\rho(b) - \omega_\rho(a) \ge 1$.
That is, $\boldsymbol{\ell}^\rho = (\ell_1^\rho, \ell_2^\rho, \ldots, \ell_{J^\rho}^\rho)$ where ℓ_j^ρ is $\max\{g \in \boldsymbol{g} \mid \omega_\rho(g) - \omega_\rho(\ell_{j-1}^\rho) \le 1\}$ (interpret ℓ_0^ρ as 0), and J^ρ is $\min\{j \mid \omega_\rho(\hat{U}) - \omega_\rho(\ell_j^\rho) \le 1\}$.

3: For each $g_i \in \boldsymbol{g}$, define time $t_i \in [U]$ to be minimum such that $\sum_{t=1}^{t_i} x_t \ge g_i$.
Return the schedule $\big\{\big(t_i,\, \{\rho \mid g_i \in \boldsymbol{\ell}^\rho\}\big) \mid g_i \in \boldsymbol{g}\big\}$.

The idea of algorithm Round_p and its analysis are from [12]. The presentation below highlights some important technical subtleties in the proof.

Lemma 1. *For any distribution p and fractional solution $\boldsymbol{x}$, the above algorithm, $\mathsf{Round}_p(C, c, \mathcal{D}, \boldsymbol{x})$, returns a schedule of expected cost at most $cost(\boldsymbol{x})/\mathcal{Z}(p)$.*

Proof. Feasibility. Suppose for contradiction that the schedule does not satisfy some demand (ρ, r, d) in $\mathcal{D}$. Then (ignoring boundary cases) there are consecutive local cutoffs ℓ_j^ρ and ℓ_{j+1}^ρ equal to global cutoffs g_i and $g_{i'}$ whose times t_i and $t_{i'}$ (per Step 3) satisfy $t_i < r \le d < t_{i'}$, and, hence, $t_i + 1 \le r \le d \le t_{i'} - 1$. But, then, by Step 2 of the algorithm,

$$1 \;\ge\; \omega_\rho(\ell_{j+1}^\rho) - \omega_\rho(\ell_j^\rho) \;=\; \omega_\rho(g_{i'}) - \omega_\rho(g_i) \;>\; \sum_{t=t_i+1}^{t_{i'}-1} x_t^\rho \;\ge\; \sum_{t=r}^{d} x_t^\rho \;\ge\; 1,$$

where the last step follows from LP constraint (2), and the proper inequality in the second-to-last step follows from the minimality of $t_{i'}$ in Step 3 of the algorithm. This gives $1 > 1$, a contradiction. (The boundary cases, and the proof that Step 2 is well-defined, are similar.)

To finish the proof, for each term in the cost $C|\boldsymbol{g}| + \sum_\rho c_\rho |\boldsymbol{\ell}^\rho|$, we bound the term's expectation by $1/\mathcal{Z}(p)$ times its corresponding part in $cost(\boldsymbol{x})$.

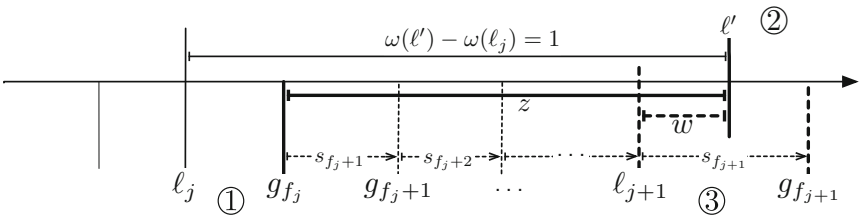


Fig. 2. The proof of $\mathbf{E}\big[\omega(\ell_{j+1}) - \omega(\ell_j) \mid S_j\big] \geq \mathcal{Z}(p)$. Dashed lines are for quantities that are independent of the current state S_j, but determined by the next state S_{j+1}.

The global order cost $C|\boldsymbol{g}|$. The expectation of each global cutoff g_{i+1}, given g_i, is $g_i + \mathbf{E}[p]$, which (by definition of $\mathcal{Z}(p)$) is at least $g_i + \mathcal{Z}(p)$. The final index I is the first such that $g_I \geq \hat{U} - 1$, so $g_I < \hat{U}$. By Wald's equation (Lemma 5), since I is a stopping time, the expected length of $\boldsymbol{g}$ is at most $\hat{U}/\mathcal{Z}(p)$. So, $\mathbf{E}[C|\boldsymbol{g}|]$ is at most $C\hat{U}/\mathcal{Z}(p)$. In comparison the global order cost in $cost(\boldsymbol{x})$ is $C\sum_{t=1}^{U} x_t = C\,\hat{U}$.

The retailer cost $c_\rho|\boldsymbol{\ell}^\rho|$ *for* ρ. Fix a retailer ρ. Since ρ is fixed for the rest of the proof, we may omit it as a subscript or superscript. Let $\boldsymbol{\ell}$ be ρ's local cutoff sequence $(\ell_1, \ell_2, \ldots, \ell_J)$. For each $j = 1, 2, \ldots, J$, define the *state* S_j *after step* j to be the first f_j random samples, where f_j is the number of random samples needed to determine ℓ_j. Generally, a given global cutoff g_i will be chosen as the jth local cutoff ℓ_j iff $\omega(g_i) - \omega(\ell_{j-1}) \leq 1 < \omega(g_{i+1}) - \omega(\ell_{j-1})$. So, f_j equals $i + 1$, where i is the index such that $g_i = \ell_j$. That is, g_{f_j} *follows* ℓ_j in the global sequence. (The only exception is the *last* local cutoff ℓ_J, which can be the maximum global cutoff g_I, in which case it is not followed by any cutoff and $f_J = I$.)

For the analysis, define $S_0 = (s_1)$ and $\ell_0 = 0$ so $\omega(\ell_0) = 0$.

We claim that, *with each step* $j = 0, \ldots, J-1$, *given the state* S_j *after step* j, *the expected increase in* $\omega(\cdot)$ *during step* $j+1$, *namely* $\mathbf{E}[\omega(\ell_{j+1}) - \omega(\ell_j) \mid S_j]$, *is at least* $\mathcal{Z}(p)$. Before we prove the claim, note that this implies the desired bound: by the stopping condition for $\boldsymbol{\ell}$, the total increase in ω is at most $\omega(\hat{U})$, so by Wald's equation (using that the last index J is a stopping time), the expectation of J is at most $\omega(\hat{U})/\mathcal{Z}(p)$. So, $\mathbf{E}[\,c_\rho|\boldsymbol{\ell}|\,]$, the expected cost for retailer ρ, is at most $c_\rho\,\omega(\hat{U})/\mathcal{Z}(p)$. In comparison, the cost for retailer ρ in $cost(\boldsymbol{x})$ is $c_\rho \sum_{t=1}^{U} x_t^\rho = c_\rho\,\omega(\hat{U})$.

To prove the claim, we describe how moving from state S_j to state S_{j+1} is a play of the tally game in the definition of $\mathcal{Z}(p)$. Fix any j and condition on the state S_j. Fig. 2 shows the steps:

① The current state S_j determines the j'th local cutoff ℓ_j and the following global cutoff g_{f_j}.

② Given ℓ_j and g_{f_j}, the next local cutoff for retailer ρ will be the maximum global cutoff in the interval $[g_{f_j}, \ell']$, where ℓ' is chosen so that $\omega(\ell') - \omega(\ell_j)$ equals 1. (Note that $\ell' < \hat{U}$ because, since we haven't stopped yet, $\omega(\hat{U}) - \omega(\ell_j) > 1$.)

③ The algorithm reaches the next state S_{j+1} by drawing some more random samples $s_{f_j+1}, s_{f_j+2}, \ldots, s_i$ from p, stopping with the first index i such that the corresponding global cutoff g_i exceeds ℓ'. (That is, such that $g_i = g_{f_j} + s_{f_j+1} + \cdots + s_i > \ell'$.) The next local cutoff ℓ_{j+1} is g_{i-1} (the global cutoff just before g_i, so that $g_{i-1} \le \ell' < g_i$) and this index i is f_{j+1}; that is, the next state S_{j+1} is $(s_1, s_2, \ldots, s_i)$.

Step ③ is a play of the "tally game" in the definition of $\mathcal{Z}(p)$, with threshold $z = \ell' - g_{f_j}$. The waste w equals the gap $\ell' - \ell_{j+1}$. By the definition of $\mathcal{Z}(p)$, the expectation of w is $\mathcal{W}(p, z) \le 1 - \mathcal{Z}(p)$. Finally,

$$\omega(\ell_{j+1}) - \omega(\ell_j) \;=\; 1 - (\omega(\ell') - \omega(\ell_{j+1})) \;\ge\; 1 - (\ell' - \ell_{j+1}) \;=\; 1 - w.$$

The expectation of $1 - w$ is at least $\mathcal{Z}(p)$, proving the claim.

The careful reader may notice that the above analysis is incorrect for the last step J, because it may happen that there is no global cutoff after ℓ'. (Then $\ell_J = g_I = \max_i g_i$.) To analyze this case, imagine modifying the algorithm so that, in choosing $\boldsymbol{g} = (g_1, g_2, \ldots, g_I)$, instead of stopping with $I = i$ such that $g_i \ge \hat{U} - 1$, it stops with $I = i$ such that $g_i \ge \hat{U}$. Because the last global cutoff is now at least $\hat{U}$, and $\ell' < \hat{U}$, there is always a global cutoff after ℓ'. So the previous analysis is correct for the modified algorithm, and its expected local order cost is bounded as claimed. To finish, observe that, since this modification only *extends* $\boldsymbol{g}$, it cannot *decrease* the number of local cutoffs selected from $\boldsymbol{g}$, so the modification does not decrease the local order cost. □

2.1 Upper Bound of $e/(e-1) \approx 1.582$

The next utility lemma says that, in analyzing the expected waste in the tally game, it is enough to consider thresholds z in $[0, 1]$.

Lemma 2. *For any distribution p on $[0, 1]$, $\sup_{z \ge 0} \mathcal{W}(p, z) \;=\; \sup_{z \in [0,1)} \mathcal{W}(p, z)$.*

Proof. Play the tally game with any threshold $z > 1$. Consider the first prefix sum $s_1 + s_2 + \cdots + s_h$ of the samples such that the "slack" $z - (s_1 + s_2 + \cdots + s_h)$ is at most 1. Let random variable z' be this slack. Note $z' \in [0, 1)$. Then, conditioned on $z' = y$, the expected waste is $\mathcal{W}(p, y)$, which is at most $\sup_{Y \in [0,1]} \mathcal{W}(p, Y)$. Thus, for *any* threshold $z \ge 1$, $\mathcal{W}(p, z)$ is at most $\sup_{Y \in [0,1)} \mathcal{W}(p, Y)$. □

Now consider the specific probability distribution p on $[0, 1]$ with probability density function $p(y) = 1/y$ for $y \in [1/e, 1]$ and $p(y) = 0$ elsewhere.

Lemma 3. *For this distribution p, $\mathcal{Z}(p) \;\ge\; (e-1)/e \;=\; 1 - 1/e$.*

Proof. By Lemma 2, $\mathcal{Z}(p)$ is the minimum of $\mathbf{E}[p]$ and $1 - \max_{z \in [0,1]} \mathcal{W}(p, z)$.

By direct calculation, $\mathbf{E}[p] = \int_{1/e}^{1} y\, p(y)\, dy \;=\; \int_{1/e}^{1} 1\, dy = 1 - 1/e$.

Now consider playing the tally game with threshold z. If $z \in [0, 1/e]$, then (since the waste is at most z) trivially $\mathcal{W}(p, z) \le z \le 1/e$.

So consider any $z \in [1/e, 1]$. Let s_1 be just the first sample.

The waste is z if $s_1 > z$ and otherwise is at most $z - s_1$.
So, the expected waste is at most $\Pr[s_1 > z]z + \Pr[s_1 \le z]\,\mathbf{E}[z - s_1 \mid s_1 \le z]$.
This simplifies to $z - \Pr[s_1 \le z]\,\mathbf{E}[s_1 \mid s_1 \le z]$, which by calculation is

$$z - \int_{1/e}^{z} y\,p(y)\,dy \;=\; z - \int_{1/e}^{z} dy \;=\; z - (z - 1/e) \;=\; 1/e. \qquad \square$$

2.2 Upper Bound of 1.574

Next we define a probability distribution on $[0, 1]$ that has a point mass at 1. Fix $\theta = 0.36455$ (slightly less than $1/e$). Over the half-open interval $[0, 1)$, the probability density function p is $p(y) = \begin{cases} 0 & \text{for } y \in [0, \theta) \\ 1/y & \text{for } y \in [\theta, 2\theta) \\ \frac{1-\ln((y-\theta)/\theta)}{y} & \text{for } y \in [2\theta, 1). \end{cases}$
The probability of choosing 1 is $1 - \int_0^1 p(y)\,dy \approx 0.0821824$.

Note that $p(y) \ge 0$ for $y \in [2\theta, 1)$ since $\ln((1-\theta)/\theta) \approx 0.55567$.

Lemma 4. *The statistic $\mathcal{Z}(p)$ for this p is at least* $0.63533 > 1/1.574$.

The proof is in the full paper [4]. Here is a sketch.

Proof (sketch). By Lemma 2, $\mathcal{Z}(p)$ is the minimum of $\mathbf{E}[p]$ and $1 - \sup_{z\in[0,1]} \mathcal{W}(p, z)$.

That $\mathbf{E}[p] \ge 0.63533$ follows from a direct calculation (about five lines; details in the full proof).

It remains to show $1 - \sup_{z\in[0,1)} \mathcal{W}(p, z) \ge 0.63533$. To do this, we show $\sup_{z\in[0,1)} \mathcal{W}(p, z) = \theta$ $(\le 1 - 0.63533)$.

In the tally game defining $\mathcal{W}(p, z)$, let s_1 be the first random sample drawn from p. If $s_1 > z$, then the waste equals z. Otherwise, the process continues recursively with z replaced by $z' = z - s_1$. This gives the recurrence

$$\mathcal{W}(p, z) \;=\; z\,\Pr[s_1 > z] + \int_{\theta}^{z} \mathcal{W}(p, z - y)\,p(y)\,dy.$$

We analyze the right-hand side of the recurrence in three cases.

Case (i) $z \in [0, \theta)$. The recurrence gives $\mathcal{W}(p, z) = z$ because $\Pr[s_1 > z] = 1$.

Case (ii) $z \in [\theta, 2\theta)$. For $y \in [\theta, z]$, we have $z - y < \theta$, so by Case (i) $\mathcal{W}(p, z-y) = z - y$. Substituting and calculating (about two lines) gives $\mathcal{W}(p, z) = \theta$.

Case (iii) $z \in [2\theta, 1)$. For $y \in [\theta, z]$, we have $z - y < 2\theta$, so Case (i) or (ii) applies to simplify $\mathcal{W}(p, z - y)$ (to $z - y$ if $z - y < \theta$ or θ otherwise). The calculation (about seven lines) gives $\mathcal{W}(p, z) = \theta$. $\square$

Theorem 1. *JRP-D has a randomized polynomial-time* 1.574*-approximation algorithm, and the integrality gap of the LP relaxation is at most* 1.574.

Proof. By Lemma 4, for any fractional solution $\boldsymbol{x}$, the algorithm Round_p (using the probability distribution p from that lemma) returns a feasible schedule of expected cost at most 1.574 times $\mathit{cost}(\boldsymbol{x})$.

To see that the schedule can be computed in polynomial time, note first that the (discrete-time) LP relaxation can be solved in polynomial time. The optimal

solution $\boldsymbol{x}$ is minimal, so each x_t is at most 1, so $\hat{U} = \sum_t x_t$ is at most the number of demands, n. In the algorithm Round_p, each sample from the distribution p from Lemma 4 can be drawn in polynomial time. Each sample is $\Omega(1)$, and the sum of the samples is at most $\hat{U} \le n$, so the number of samples is $O(n)$. Then, for each retailer ρ, each integral $\mu(\ell^{\rho}_{j-1}, g)$ in step 3 can be evaluated in constant amortized time per evaluation, so the time per retailer is $O(n)$. □

For the record, here is the variant of Wald's equation (also known as Wald's identity or Wald's lemma, and a consequence of standard "optional stopping" theorems) that we use above. Consider a random experiment that, starting from a fixed start state S_0, produces a random sequence of states $S_1, S_2, S_3, \ldots$ Let random index $T \in \{0, 1, 2, \ldots\}$ be a stopping time for the sequence, that is, for any positive integer t, the event "$T < t$" is determined by state S_t. Let function $\phi : \{S_t\} \to \mathbb{R}$ map the states to $\mathbb{R}$.

Lemma 5 (Wald's equation). *Suppose that*
(i) $(\forall t < T)\ \mathbf{E}[\phi(S_{t+1}) \mid S_t] \ge \phi(S_t) + \xi$ *for fixed* ξ*, and*
(ii) either $(\forall t < T)\ \phi(S_{t+1}) - \phi(S_t) \ge F$ *or* $(\forall t < T)\ \phi(S_{t+1}) - \phi(S_t) \le F$*, for some fixed finite* F*, and* T *has finite expectation.*
Then $\xi\, \mathbf{E}[T] \le \mathbf{E}[\phi(S_T) - \phi(S_0)]$.

The proof is standard; it is in the full paper [4].

In the applications here, we always have $\xi = \mathcal{Z}(p) > 0$ and $\phi(S_T) - \phi(S_0) \le U$ for some fixed U. In this case Wald's equation implies $\mathbf{E}[T] \le U/\mathcal{Z}(p)$.

3 Upper Bound of 1.5 for Equal-Length Periods

In this section, we present a 1.5-approximation algorithm for the case where all the demand periods are of equal length. For convenience, we allow here release times and deadlines to be rational numbers and we assume that all demand periods have length 1.

We denote the input instance by $\mathcal{I}$. Let the *width* of an instance be the difference between the deadline of the last demand and the release time of the first one. The building block of our approach is an algorithm that creates an optimal solution to an instance of width at most 3. Later, we divide $\mathcal{I}$ into overlapping sub-instances of width 3, solve each of them optimally, and finally show that by aggregating their solutions we obtain a 1.5-approximation for $\mathcal{I}$.

Lemma 6. *A solution to any instance* $\mathcal{J}$ *of width at most* 3 *consisting of unit-length demand periods can be computed in polynomial time.*

Proof. We shift all demands in time, so that $\mathcal{J}$ is entirely contained in interval $[0, 3]$. Recall that C is the warehouse ordering cost and c_ρ is the ordering cost of retailer $\rho \in \{1, 2, ..., m\}$. Without loss of generality, we can assume that all retailers $1, .., m$ have some demands.

Let $d_{\min}$ be the first deadline of a demand from $\mathcal{J}$ and $r_{\max}$ the last release time. If $r_{\max} \le d_{\min}$, then placing one order at any time from $[r_{\max}, d_{\min}]$ is sufficient (and necessary). Its cost is then equal to $C + \sum_\rho c_\rho$.

Thus, in the following we focus on the case $d_{\min} < r_{\max}$. Any feasible solution has to place an order at or before $d_{\min}$ and at or after $r_{\max}$. Furthermore, by shifting these orders we may assume that the first and last orders occur exactly at times $d_{\min}$ and $r_{\max}$, respectively.

The problem is thus to choose a set T of warehouse ordering times that contains $d_{\min}$, $r_{\max}$, and possibly other times from the interval $(d_{\min}, r_{\max})$, and then to decide, for each retailer ρ, which warehouse orders it joins. Note that $r_{\max} - d_{\min} \leq 1$, and therefore each demand period contains $d_{\min}$, $r_{\max}$, or both. Hence, all demands of a retailer ρ can be satisfied by joining the warehouse orders at times $d_{\min}$ and $r_{\max}$ at additional cost of $2b_\rho$. It is possible to reduce the retailer ordering cost to c_ρ if (and only if) there is a warehouse order that occurs within $\mathcal{D}_\rho$, where $\mathcal{D}_\rho$ is the intersection of all demand periods of retailer ρ. (To this end, $\mathcal{D}_\rho$ has to be non-empty.)

Hence, the optimal cost for $\mathcal{J}$ can be expressed as the sum of four parts:

(i) the unavoidable ordering cost c_ρ for each retailer ρ,
(ii) the additional ordering cost c_ρ for each retailer ρ with empty $\mathcal{D}_\rho$,
(iii) the total warehouse ordering cost $C \cdot |T|$, and
(iv) the additional ordering cost c_ρ for each retailer ρ whose $\mathcal{D}_\rho$ is non-empty and does not contain any ordering time from T.

As the first two parts of the cost are independent of T, we focus on minimizing the sum of parts (iii) and (iv), which we call the *adjusted cost*. Let $AC(t)$ be the minimum possible adjusted cost associated with the interval $[d_{\min}, t]$ under the assumption that there is an order at time t. Formally, $AC(t)$ is the minimum, over all choices of sets $T \subseteq [d_{\min}, t]$ that contain $d_{\min}$ and t, of $C \cdot |T| + \sum_{\rho \in Q(T)} c_\rho$, where $Q(T)$ is the set of retailers ρ for which $\mathcal{D}_\rho \neq \emptyset$ and $\mathcal{D}_\rho \subseteq [0, t] - T$. (Note that the second term consists of expenditures that actually occur outside the interval $[d_{\min}, t]$.)

As there are no $\mathcal{D}_\rho$'s strictly to the left of $d_{\min}$, $AC(d_{\min}) = C$. Furthermore, to compute $AC(t)$ for any $t \in (d_{\min}, r_{\max}]$, we can express it recursively using the value of $AC(u)$ for $u \in [d_{\min}, t)$ being the warehouse order time immediately preceding t in the set T that realizes $AC(t)$. This gives us the formula

$$AC(t) = C + \min_{u \in [d_{\min}, t)} \left(AC(u) + \sum_{\rho : \emptyset \neq \mathcal{D}_\rho \subset (u,t)} c_\rho \right) .$$

In the minimum above, we may restrict computation of $AC(t)$ to t's and u's that are ends of demand periods. Hence, the actual values of function $AC(\cdot)$ can be computed by dynamic programming in polynomial time. Finally, the total adjusted cost is equal to $AC(r_{\max})$. Once we computed the minimum adjusted cost, recovering the actual orders can be performed by a straightforward extension of the dynamic programming presented above. □

Now, we construct an approximate solution for the original instance $\mathcal{I}$ consisting of unit-length demand periods. For $i \in \mathbb{N}$, let $\mathcal{I}_i$ be the sub-instance containing all demands entirely contained in $[i, i+3)$. By Lemma 6, an optimal solution for $\mathcal{I}_i$, denoted $A(\mathcal{I}_i)$, can be computed in polynomial time. Let S_0 be the

solution created by aggregating $A(\mathcal{I}_0), A(\mathcal{I}_2), A(\mathcal{I}_4), \ldots$ and S_1 by aggregating $A(\mathcal{I}_1), A(\mathcal{I}_3), A(\mathcal{I}_5), \ldots$. Among solutions S_0 and S_1, we output the one with the smaller cost.

Theorem 2. *The above algorithm produces a feasible solution of cost at most* 1.5 *times the optimum cost.*

Proof. Each unit-length demand of instance $\mathcal{I}$ is entirely contained in some $\mathcal{I}_{2k}$ for some $k \in \mathbb{N}$. Hence, it is satisfied in $A(\mathcal{I}_{2k})$, and thus also in S_0, which yields the feasibility of S_0. An analogous argument shows the feasibility of S_1.

To estimate the approximation ratio, we fix an optimal solution Opt for instance $\mathcal{I}$ and let opt_i be the cost of Opt's orders in the interval $[i, i+1)$. Note that Opt's orders in $[i, i+3)$ satisfy all demands contained entirely in $[i, i+3)$. Since $A(\mathcal{I}_i)$ is an optimal solution for these demands, we have $cost(A(\mathcal{I}_i)) \leq opt_i + opt_{i+1} + opt_{i+2}$ and, by taking the sum, we obtain $cost(S_0) + cost(S_1) \leq \sum_i cost(A(\mathcal{I}_i)) \leq 3 \cdot cost(\text{Opt})$. Therefore, either of the two solutions (S_0 or S_1) has cost at most $1.5 \cdot cost(\text{Opt})$. □

4 Two Lower Bounds

In this section we present two lower bounds on the integrality gap of the LP relaxation from Section 1:

Theorem 3. (i) *The integrality gap of the LP relaxation is at least* $\frac{1}{2}(1+\sqrt{2})$*, which is at least* 1.207*.* (ii) *The integrality gap is at least* 1.2 *for instances with equal-length demand periods.*

In the full paper [4], we sketch how the lower bound in Part (i) can be increased to 1.245 via a computer-based proof; we also give the complete proof of Thm. 3 (about six pages altogether). Here is a sketch of that proof.

Proof (sketch). It is convenient to work with a *continuous-time* variant of the LP, in which the universe $\mathcal{U}$ of allowable release times, deadlines and order times is the entire interval $[0, U]$, where U is the maximum deadline. Time t now is a real number ranging over interval $[0, U]$. A fractional solution is now represented by *functions*, $x : [0, U] \to \mathbb{R}_{\geq 0}$ and $x^\rho : [0, U] \to \mathbb{R}_{\geq 0}$, for each retailer ρ. To retain consistency, we will write x_t and x_t^ρ for the values of these functions. For any fractional solution $\boldsymbol{x}$, then, in the LP formulation each sum over t is replaced by the appropriate integral. For example, the objective function will now take form $\int_{t=0}^{U} (C\, x_t + \sum_{\rho=1}^{m} c_\rho\, x_t^\rho)$. By a straightforward limit argument (to be provided in the final version of the paper), the continuous-time LP has the same integrality gap as the discrete-time LP.

(i) The instance used to prove Part (i) has $C = 1$ and two retailers numbered (for convenience) 0 and 1, one with $c_0 = 0$ and the other with $c_1 = \sqrt{2}$. We use infinitely many demand periods: for any t, the first retailer has demand periods $[t, t+1]$ and the second retailer has demand periods $[t, t+\sqrt{2}]$. A fractional solution where retailer 0 orders at rate 1 and retailer 1 orders at rate $1/\sqrt{2}$

is feasible and its cost is 2 per time step. Now consider some integral solution. Without loss of generality, retailer 0 orders any time a warehouse order is issued. Retailer 0 must make at least one order per time unit, so his cost (counting the warehouse order cost as his) is 1 per time unit. Retailer 1 must make at least one order in any time interval of length $\sqrt{2}$, so the cost of his orders, not including the warehouse cost, is at least 1 per time unit as well. This already gives us cost 2 per time unit, the same as the optimal fractional cost. But in order to synchronize the orders of retailer 1 with the warehouse orders, the schedule needs to increase either the number of retailer 1's orders or the number of warehouse orders by a constant fraction, thus creating a gap.

(ii) The argument for Part (ii) is more involved. We only outline the general idea. Take $C = 1$ and three retailers numbered 0, 1 and 2, each with order cost $c_\rho = \frac{1}{3}$. The idea is to create an analogue of a 3-cycle, which has a fractional vertex cover with all vertices assigned value $\frac{1}{2}$ and total cost $\frac{5}{6}$, while any integral cover requires two vertices. We implement this idea by starting with the following fractional solution x: if $t \bmod 3 = 0$, then $x_t = x_t^0 = x_t^1 = \frac{1}{2}$ and $x_t^2 = 0$; if $t \bmod 3 = 1$, then $x_t = x_t^1 = x_t^2 = \frac{1}{2}$ and $x_t^0 = 0$; if $t \bmod 3 = 2$, then $x_t = x_t^0 = x_t^2 = \frac{1}{2}$ and $x_t^1 = 0$. The cost is $\frac{5}{6}$ per time unit. Then we choose demand periods that x satisfies, but such that, in any (integral) schedule, each retailer must have at least one order in every three time units $\{t, t+1, t+3\}$, *and* there has to be a warehouse order in every two time units $\{t, t+1\}$. These costs independently add up to $\frac{5}{6}$ per time unit, even ignoring the requirement that retailers have orders only when the warehouse does. To synchronize the orders to meet this additional requirement, any schedule must further increase the order rate by a constant fraction, thus creating a gap. □

5 APX Hardness for Unit Demand Periods

Theorem 4. *JRP-D is $\mathbb{APX}$-hard even if restricted to instances with unit demand periods and with at most four demands per retailer.*

The proof (about four pages) is in the full paper [4]. Here is the idea.

Proof (idea). We use the result by Alimonti and Kann [1] that Vertex Cover is $\mathbb{APX}$-hard even for cubic graphs. For any given cubic graph $G = (V, E)$ with n vertices (where n is even) and $m = 1.5n$ edges, in polynomial time we construct an instance $\mathcal{J}_G$ of JRP-D, such that the existence of a vertex cover for G of size at most K is equivalent to the existence of an order schedule for $\mathcal{J}_G$ of cost at most $10.5n + K + 6$. In $\mathcal{J}_G$ all demand periods have the same length and each retailer has at most four demands. The construction consists of gadgets that represent G's vertices and edges. The main challenge, related to the equal-length restriction, is in "transmitting information" along the time axis about the vertices chosen for a vertex cover. We resolve it by having each vertex represented twice and assuring consistency via an appropriate sub-gadget. □

6 Final Comments

The integrality gap for standard JRP-D LP relaxation is between 1.245 and 1.574. We conjecture that neither bound is tight, but that the refined distribution for the tally game given here is essentially optimal, so that improving the upper bound will require a different approach.

There is a simple algorithm for JRP-D that provides a $(1, 2)$-approximation, in the following sense: its warehouse order cost is not larger than that in the optimum, while its retailer order cost is at most twice that in the optimum [12]. One can then try to balance the two approaches by choosing each algorithm with a certain probability. This simple approach does not improve the ratio. But it may be possible to achieve a better ratio if, instead of using our algorithm as presented, we appropriately adjust the probability distribution.

If we parametrize JRP-D by the maximum number p of demand periods of each retailer, its complexity status is essentially resolved: for $p \geq 3$ the problem is $\mathbb{APX}$-hard [12], while for $p \leq 2$ it can be solved in polynomial time (by a greedy algorithm for $p = 1$ and dynamic programming for $p = 2$). In case of equal-length demand periods, we showed that the problem is $\mathbb{APX}$-hard for $p \geq 4$, but the case $p = 3$ remains open, and it would be nice to settle this case as well. We conjecture that in this case the problem is $\mathbb{NP}$-complete.

Acknowledgements. We would like to thank Łukasz Jeż, Jiří Sgall, and Grzegorz Stachowiak for stimulating discussions and useful comments.

References

1. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. Theoretical Computer Science 237(1-2), 123–134 (2000)
2. Arkin, E., Joneja, D., Roundy, R.: Computational complexity of uncapacitated multi-echelon production planning problems. Operations Research Letters 8(2), 61–66 (1989)
3. Becchetti, L., Marchetti-Spaccamela, A., Vitaletti, A., Korteweg, P., Skutella, M., Stougie, L.: Latency-constrained aggregation in sensor networks. ACM Transactions on Algorithms 6(1), 13:1–13:20 (2009)
4. Bienkowski, M., Byrka, J., Chrobak, M., Dobbs, N., Nowicki, T., Sviridenko, M., Świrszcz, G., Young, N.E.: Approximation algorithms for the joint replenishment problem with deadlines. CoRR abs/1212.3233v2 (2013), `http://arxiv.org/abs/1212.3233v2`
5. Brito, C., Koutsoupias, E., Vaya, S.: Competitive analysis of organization networks or multicast acknowledgement: How much to wait? In: Proc. of the 15th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 627–635 (2004)
6. Buchbinder, N., Kimbrel, T., Levi, R., Makarychev, K., Sviridenko, M.: Online make-to-order joint replenishment model: Primal dual competitive algorithms. In: Proc. of the 19th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 952–961 (2008)
7. Khanna, S., Naor, J.(S.), Raz, D.: Control message aggregation in group communication protocols. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 135–146. Springer, Heidelberg (2002)

8. Levi, R., Roundy, R., Shmoys, D.B.: A constant approximation algorithm for the one-warehouse multi-retailer problem. In: Proc. of the 16th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 365–374 (2005)
9. Levi, R., Roundy, R., Shmoys, D.B.: Primal-dual algorithms for deterministic inventory problems. Mathematics of Operations Research 31(2), 267–284 (2006)
10. Levi, R., Roundy, R., Shmoys, D.B., Sviridenko, M.: A constant approximation algorithm for the one-warehouse multiretailer problem. Management Science 54(4), 763–776 (2008)
11. Levi, R., Sviridenko, M.: Improved approximation algorithm for the one-warehouse multi-retailer problem. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 188–199. Springer, Heidelberg (2006)
12. Nonner, T., Souza, A.: Approximating the joint replenishment problem with deadlines. Discrete Mathematics, Algorithms and Applications 1(2), 153–174 (2009)

Sparse Suffix Tree Construction in Small Space

Philip Bille[1], Johannes Fischer[2], Inge Li Gørtz[1], Tsvi Kopelowitz[3], Benjamin Sach[4], and Hjalte Wedel Vildhøj[1]

[1] Technical University of Denmark
{phbi,inge}@dtu.dk, hwv@hwv.dk
[2] KIT, Institute of Theoretical Informatics
johannes.fischer@kit.edu
[3] Weizmann Institute of Science
kopelot@gmail.com
[4] University of Warwick
sach@dcs.warwick.ac.uk

Abstract. We consider the problem of constructing a sparse suffix tree (or suffix array) for b suffixes of a given text T of length n, using only $O(b)$ words of space during construction. Attempts at breaking the naive bound of $\Omega(nb)$ time for this problem can be traced back to the origins of string indexing in 1968. First results were only obtained in 1996, but only for the case where the suffixes were evenly spaced in T. In this paper there is no constraint on the locations of the suffixes.

We show that the sparse suffix tree can be constructed in $O(n \log^2 b)$ time. To achieve this we develop a technique, which may be of independent interest, that allows to efficiently answer b longest common prefix queries on suffixes of T, using only $O(b)$ space. We expect that this technique will prove useful in many other applications in which space usage is a concern. Our first solution is Monte-Carlo and outputs the correct tree with high probability. We then give a Las-Vegas algorithm which also uses $O(b)$ space and runs in the same time bounds with high probability when $b = O(\sqrt{n})$. Furthermore, additional tradeoffs between the space usage and the construction time for the Monte-Carlo algorithm are given.

1 Introduction

In the *sparse text indexing problem* we are given a string $T = t_1 \dots t_n$ of length n, and a list of b interesting positions in T. The goal is to construct an *index* for only those b positions, while using only $O(b)$ words of space during the construction process (in addition to storing the text T). Here, by index we mean a data structure allowing for the quick location of all occurrences of patterns *starting at interesting positions only*. A natural application comes from computational biology, where the string would be a sequence of nucleotides or amino acids, and additional biological knowledge rules out many positions where patterns could potentially start. Another application is indexing far eastern languages, where one might be interested in indexing only those positions where words start, but natural word boundaries do not exist.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 148–159, 2013.

Examples of suitable $O(b)$ space indexes include suffix trees [18] or suffix arrays [14] built on only those suffixes starting at interesting positions. Of course, one can always first compute a full-text suffix tree or array in linear time, and then postprocess it to include the interesting positions only. The problem of this approach is that it needs $O(n)$ words of *intermediate working space*, which may be much more than the $O(b)$ words needed for the final result, and also much more than the space needed for storing T itself. In situations where the RAM is large enough for the string itself, but not for an index on *all* positions, a more space efficient solution is desirable. Another situation is where the text is held in read-only memory and only a small amount of read-write memory is available. Such situations often arise in embedded systems or in networks, where the text may be held remotely.

A "straightforward" space-saving solution would be to sort the interesting suffixes by an *arbitrary* string sorter, for example, by inserting them one after the other into a compacted trie. However, such an approach is doomed to take $\Omega(nb + n \log n)$ time [5], since it takes no advantage of the fact that the strings are suffixes of one large text, so it cannot be faster than a general string sorter.

Breaking these naive bounds has been a problem that can be traced back to—according to Kärkkäinen and Ukkonen [10]—the origins of string indexing in 1968 [15]. First results were only obtained in 1996, where Andersson et al. [2,3] and Kärkkäinen and Ukkonen [10] considered *restricted* variants of the problem: the first [2,3] assumed that the interesting positions coincide with natural *word boundaries* of the text, and the authors achieved *expected* linear running time using $O(b)$ space. The expectancy was later removed [9,7], and the result was recently generalized to variable length codes such as Huffman code [17]. The second restricted case [10] assumed that the text of interesting positions is *evenly spaced*; i.e., every k^{th} position in the text. They achieved linear running time and optimal $O(b)$ space. It should be mentioned that the data structure by Kärkkäinen and Ukkonen [10] was not necessarily meant for finding only pattern occurrences starting at the evenly spaced indexed positions, as a large portion of the paper is devoted to recovering *all* occurrences from the indexed ones. Their technique has recently been refined by Kolpakov et al. [13]. Another restricted case admitting an $O(b)$ space solution is if the interesting positions have the same period ρ (i.e., if position i is interesting then so is position $i + \rho$). In this case the sparse suffix array can be constructed in $O(b\rho + b \log b)$ time. This was shown by Burkhardt and Kärkkäinen [6], who used it to sort *difference cover samples* leading to a clever technique for constructing the full suffix array in sublinear space. Interestingly, their technique also implies a time-space tradeoff for sorting b *arbitrary* suffixes in $O(v + n/\sqrt{v})$ space and $O(\sqrt{v}n + (n/\sqrt{v}) \log(n/\sqrt{v}) + vb + b \log b)$ time for any $v \in [2, n]$.

1.1 Our Results

We are the first to break the naive $O(nb)$ time algorithm for general sparse suffix trees, by showing how to construct a sparse suffix tree in $O(n \log^2 b)$ time, using only $O(b)$ words of space. To achieve this, we develop a novel technique for

performing efficient batched longest common prefix (LCP) queries, using little space. In particular, in Section 3, we show that a batch of b LCP queries can be answered using only $O(b)$ words of space, in $O(n \log b)$ time. This technique may be of independent interest, and we expect it to be helpful in other applications in which space usage is a factor. Both algorithms are Monte-Carlo and output correct answers with high probability, i.e., at least $1 - 1/n^c$ for any constant c.

In Section 5 we give a Las-Vegas version of our sparse suffix tree algorithm. This is achieved by developing a deterministic verifier for the answers to a batch of b longest common prefix queries. We show that this verifier can be used to obtain the sparse suffix tree with certainty in $O(n \log^2 b + b^2 \log b)$ time with high probability using only $O(b)$ space. For example for $b = O(\sqrt{n})$ we can construct the sparse suffix tree correctly in $O(n \log^2 b)$ time with high probability using $O(b)$ space in the worst case. This follows because, for verification, a single batch of b LCP queries suffices to check the sparse suffix tree. The verifier we develop encodes the relevant structure of the text in a graph with $O(b)$ edges. We then exploit novel properties of this graph to verify the answers to the LCP queries efficiently.

Finally, in Section 6, we show some tradeoffs of construction time and space usage of our Monte-Carlo algorithm, which are based on time-space tradeoffs of the batched LCP queries. In particular we show that using $O(b\alpha)$ space the construction time is reduced to $O\left(n\frac{\log^2 b}{\log \alpha} + \frac{\alpha b \log^2 b}{\log \alpha}\right)$. So, for example, the cost for constructing the sparse suffix tree can be reduced to $O(n \log b)$ time, using $O(b^{1+\varepsilon})$ words of space where $\varepsilon > 0$ is any constant.

2 Preliminaries

For a string $T = t_1 \cdots t_n$ of length n, denote by $T_i = t_i \cdots t_n$ the i^{th} suffix of T. The LCP of two suffixes T_i and T_j is denoted by $LCP(T_i, T_j)$, but we will slightly abuse notation and write $LCP(i, j) = LCP(T_i, T_j)$. We denote by $T_{i,j}$ the substring $t_i \cdots t_j$. We say that $T_{i,j}$ has period $\rho > 0$ iff $T_{i+\rho,j} = T_{i,j-\rho}$. Note that ρ is *a period* of $T_{i,j}$ and not necessarily the unique minimal period of $T_{i,j}$, commonly referred to as *the period*.

We assume the reader is familiar with both the suffix tree data structure [18] as well as suffix and LCP arrays [14].

Fingerprinting. We make use of the fingerprinting techniques of Karp and Rabin [11]. Our algorithms are in the word-RAM model with word size $\Theta(\log n)$ and we assume that each character in T fits in a constant number of words. Hence each character can be interpreted as a positive integer, no larger than $n^{O(1)}$. Let p be a prime between n^c and $2n^c$ (where $c > 0$ is a constant picked below) and choose $r \in \mathbb{Z}_p$ uniformly at random. A fingerprint for a substring $T_{i,j}$, denoted by $FP[i, j]$, is the number $\sum_{k=i}^{j} r^{j-k} \cdot t_k \mod p$. Two equal substrings will always have the same fingerprint, however the converse is not true. Fortunately, as each character fits in $O(1)$ words, the probability of any two different substrings having the same fingerprint is at most by $n^{-\Omega(1)}$ [11]. By making a suitable choice of c and applying the union bound we can ensure that

with probability at least $1-n^{-\Omega(1)}$, all fingerprints of substring of T are collision free. I.e. for every pair of substrings T_{i_1,j_1} and T_{i_2,j_2} we have that $T_{i_1,j_1} = T_{i_2,j_2}$ iff $FP[i_1,j_1] = FP[i_2,j_2]$. The exponent in the probability can be amplified by increasing the value of c. As c is a constant, any fingerprint fits into a constant number of words.

We utilize two important properties of fingerprints. The first is that $FP[i,j+1]$ can be computed from $FP[i,j]$ in constant time. This is done by the formula $FP[i,j+1] = FP[i,j] \cdot r + t_{j+1} \mod p$. The second is that the fingerprint of $T_{k,j}$ can be computed in $O(1)$ time from the fingerprint of $T_{i,j}$ and $T_{i,k}$, for $i \leq k \leq j$. This is done by the formula $FP[k,j] = FP[i,j] - FP[i,k] \cdot r^{j-k} \mod p$. Notice however that in order to perform this computation, we must have stored $r^{j-k} \mod p$ as computing it on the fly may be costly.

3 Batched LCP Queries

3.1 The Algorithm

Given a string T of length n and a list of q pairs of indices P, we wish to compute $LCP(i,j)$ for all $(i,j) \in P$. To do this we perform $\log q$ rounds of computation, where at the k^{th} round the input is a set of q pairs denoted by P_k, where we are guaranteed that for any $(i,j) \in P_k, LCP(i,j) \leq 2^{\log n-(k-1)}$. The goal of the k^{th} iteration is to decide for any $(i,j) \in P_k$ if $LCP(i,j) \leq 2^{\log n-k}$ or not. In addition, the k^{th} round will prepare P_{k+1}, which is the input for the $(k+1)^{\text{th}}$ round. To begin the execution of the procedure we set $P_0 = P$, as we are always guaranteed that for any $(i,j) \in P$, $LCP(i,j) \leq n = 2^{\log n}$. We will first provide a description of what happens during each of the $\log q$ rounds, and after we will explain how the algorithm uses $P_{\log q}$ to derive $LCP(i,j)$ for all $(i,j) \in P$.

A Single Round. The k^{th} round, for $1 \leq k \leq \log q$, is executed as follows. We begin by constructing the set $L = \bigcup_{(i,j)\in P_k} \{i-1, j-1, i+2^{\log n-k}, j+2^{\log n-k}\}$ of size $4q$, and construct a perfect hash table for the values in L, using a 2-wise independent hash function into a world of size q^c for some constant c (which with high probability guarantees that there are no collisions). Notice if two elements in L have the same value, then we store them in a list at their hashed value. In addition, for every value in L we store which index created it, so for example, for $i-1$ and $i+2^{\log n-k}$ we remember that they were created from i.

Next, we scan T from t_1 till t_n. When we reach t_ℓ we compute $FP[1,\ell]$ in constant time from $FP[1,\ell-1]$. In addition, if $\ell \in L$ then we store $FP[1,\ell]$ together with ℓ in the hash table. Once the scan of T is completed, for every $(i,j) \in P_k$ we compute $FP[i,i+2^{\log n-k}]$ in constant time, as we stored $FP[1,i-1]$ and $FP[1,i+2^{\log n-k}]$. Similarly we compute $FP[j,j+2^{\log n-k}]$. Notice that to do this we need to compute $r^{2^{\log n-k}} \mod p = r^{\frac{n}{2^k}}$ in $O(\log n-k)$ time, which can be easily afforded within our bounds, as one computation suffices for all pairs.

If $FP[i, i + 2^{\log n-k}] \neq FP[j, j + 2^{\log n-k}]$ then $LCP(i, j) < 2^{\log n-k}$, and so we add (i, j) to P_{k+1}. Otherwise, with high probability $LCP(i, j) \geq 2^{\log n-k}$ and so we add $(i + 2^{\log n+k}, j + 2^{\log n+k})$ to P_{k+1}. Notice there is a natural bijection between pairs in P_{k-1} and pairs in P following from the method of constructing the pairs for the next round. For each pair in P_{k+1} we will remember which pair in P originated it, which can be easily transferred when P_{k+1} is constructed from P_k.

LCP on Small Strings. After the $\log q$ rounds have taken place, we know that for every $(i, j) \in P_{\log q}$, $LCP(i, j) \leq 2^{\log n-\log q} = \frac{n}{q}$. For each such pair, we spend $O(\frac{n}{q})$ time in order to exactly compute $LCP(i, j)$. Notice that this is performed for q pairs, so the total cost is $O(n)$ for this last phase. We then construct $P_{final} = \{(i + LCP(i, j), j + LCP(i, j)) : (i, j) \in P_{\log q}\}$. For each $(i, j) \in P_{final}$ denote by $(i_0, j_0) \in P$ the pair which originated (i, j). We claim that for any $(i, j) \in P_{final}$, $LCP(i_0, j_0) = i - i_0$.

3.2 Runtime and Correctness

Each round takes $O(n+q)$ time, and the number of rounds is $O(\log q)$ for a total of $O((n + q) \log q)$ time for all rounds. The work executed for computing P_{final} is an additional $O(n)$.

The following lemma on LCPs, which follows directly from the definition, will be helpful in proving the correctness of the batched LCP query.

Lemma 1. *For any $1 \leq i, j \leq n$, for any $0 \leq m \leq LCP(i, j)$, it holds that $LCP(i + m, j + m) + m = LCP(i, j)$.*

We now proceed on to prove that for any $(i, j) \in P_{final}$, $LCP(i_0, j_0) = i - i_0$. Lemma 2 shows that the algorithm behaves as expected during the $\log q$ rounds, and Lemma 3 proves that the work done in the final round suffices for computing the LCPs.

Lemma 2. *At round k, for any $(i_k, j_k) \in P_k$, $i_k - i_0 \leq LCP(i_0, j_0) \leq i_k - i_0 + 2^{\log n-k}$, assuming the fingerprints do not give a false positive.*

Proof. The proof is by induction on k. For the base, $k = 0$ and so $P_0 = P$ meaning that $i_k = i_0$. Therefore, $i_k - i_0 = 0 \leq LCP(i_0, j_0) \leq 2^{\log n} = n$, which is always true. For the inductive step, we assume correctness for $k-1$ and we prove for k as follows. By the induction hypothesis, for any $(i_{k-1}, j_{k-1}) \in P_{k-1}$, $i - i_0 \leq LCP(i_0, j_0) \leq i - i_0 + 2^{\log n-k+1}$. Let (i_k, j_k) be the pair in P_k corresponding to (i_{k-1}, j_{k-1}) in P_{k-1}. If $i_k = i_{k-1}$ then $LCP(i_{k-1}, j_{k-1}) < 2^{\log n-k}$. Therefore,

$$\begin{aligned} i_k - i_0 = i_{k-1} - i_0 &\leq LCP(i_0, j_0) \\ &\leq i_{k-1} - i_0 + LCP(i_{k-1}, j_{k-1}) \leq i_k - i_0 + 2^{\log n-k}. \end{aligned}$$

If $i_k = i_{k-1} + 2^{\log n - k}$ then $FP[i, i + 2^{\log n-k}] = FP[j, j + 2^{\log n-k}]$, and because we assume that the fingerprints do not produce false positives, $LCP(i_{k-1}, j_{k-1}) \geq 2^{\log n-k}$. Therefore,

$$\begin{aligned} i_k - i_0 &= i_{k-1} + 2^{\log n-k} - i_0 \leq i_{k-1} - i_0 + LCP(i_{k-1}, j_{k-1}) \\ &\leq LCP(i_0, j_0) \leq i_{k-1} - i_0 + 2^{\log n-k+1} \\ &\leq i_k - i_0 + 2^{\log n-k}, \end{aligned}$$

where the third inequality holds from Lemma 1, and the fourth inequality holds as $LCP(i_0, j_0) = i_{k-1} - i_0 + LCP(i_{k-1}, j_{k-1})$ (which is the third inequality), and $LCP(i_{k-1}, j_{k-1}) \leq 2^{\log n-k+1}$ by the induction hypothesis. □

Lemma 3. *For any* $(i, j) \in P_{final}$, $LCP(i_0, j_0) = i - i_0 (= j - j_0)$.

Proof. Using Lemma 2 with $k = \log q$ we have that for any $(i_{\log q}, j_{\log q}) \in P_{\log q}$, $i_{\log q} - i_0 \leq LCP(i_0, j_0) \leq i_{\log q} - i_0 + 2^{\log n - \log q} = i_{\log q} - i_0 + \frac{n}{q}$. Because $LCP(i_{\log q}, j_{\log q}) \leq 2^{\log n - \log q}$ it must be that $LCP(i_0, j_0) = i_{\log q} - i_0 + LCP(i_{\log q}, j_{\log q})$. Notice that $i_{final} = i_{\log q} + LCP(i_{\log q}, j_{\log q})$. Therefore, $LCP(i_0, j_0) = i_{final} - i_0$ as required. □

Notice that the space used in each round is the set of pairs and the hash table for L, both of which require only $O(q)$ words of space. Thus, we have obtained the following. We discuss several other time/space tradeoffs in Section 6.

Theorem 1. *There exists a randomized Monte-Carlo algorithm that with high probability correctly answers a batch of q LCP queries on suffixes from a string of length n. The algorithm uses $O((n+q)\log q)$ time and $O(q)$ space in the worst case.*

4 Constructing the Sparse Suffix Tree

We now describe a Monte-Carlo algorithm for constructing the sparse suffix tree on any b suffixes of T in $O(n \log^2 b)$ time and $O(b)$ space. The main idea is to use batched LCP queries in order to sort the b suffixes, as once the LCP of two suffixes is known, deciding which is lexicographically smaller than the other takes constant time by examining the first two characters that differ in said suffixes.

To arrive at the claimed complexity bounds, we are interested in grouping the LCP queries into $O(\log b)$ batches each containing $q = O(b)$ queries on pairs of suffixes. One way to do this is to simulate a sorting network on the b suffixes of depth $\log b$ [1]. Unfortunately, such known networks have very large constants hidden in them, and are generally considered impractical [16]. There are some practical networks with depth $\log^2 b$ such as [4], however, we wish to do better.

Consequently, we choose to simulate the quick-sort algorithm by each time picking a random suffix called the pivot, and lexicographically comparing all of the other $b - 1$ suffixes to the pivot. Once a partition is made to the set of

suffixes which are lexicographically smaller than the pivot, and the set of suffixes which are lexicographically larger than the pivot, we recursively sort each set in the partition with the following modification. Each level of the recursion tree is performed concurrently using one single batch of $q = O(b)$ LCP queries for the entire level. Thus, by Theorem 1 a level can be computed in $O(n \log b)$ time and $O(b)$ space. Furthermore, with high probability, the number of levels in the randomized quicksort is $O(\log b)$, so the total amount of time spent is $O(n \log^2 b)$ with high probability. The time bound can immediately be made worst-case by aborting if the number of levels becomes too large, since the algorithm is still guaranteed to return the correct answer with high probability.

Notice that once the suffixes have been sorted, then we have in fact computed the sparse suffix array for the b suffixes. Moreover, the corresponding sparse LCP array can be obtained as a by-product or computed subsequently by a answering a single batch of $q = O(b)$ LCP queries in $O(n \log b)$ time. Hence we have obtained the following.

Theorem 2. *There exists a randomized Monte-Carlo algorithm that with high probability correctly constructs the sparse suffix array and the sparse LCP array for any b suffixes from a string of length n. The algorithm uses $O(n \log^2 b)$ time and $O(b)$ space in the worst case.*

Having obtained the sparse suffix and LCP arrays, the sparse suffix tree can be constructed deterministically in $O(b)$ time and space using well-known techniques, e.g. by simulating a bottom-up traversal of the tree [12].

Corollary 1. *There exists a randomized Monte-Carlo algorithm that with high probability correctly constructs the sparse suffix tree on b suffixes from a string of length n. The algorithm uses $O(n \log^2 b)$ time and $O(b)$ space in the worst case.*

5 Verifying the Sparse Suffix and LCP Arrays

In this section we give a deterministic algorithm which verifies the correctness of the sparse suffix and LCP arrays constructed in Theorem 2. This immediately gives a Las-Vegas algorithm for constructing either the sparse suffix array or sparse suffix tree with certainty. For space reasons some proofs are omitted.

First observe that as lexicographical ordering is transitive it suffices to verify the correct ordering of each pair of indices which are adjacent in the sparse suffix array. The correct ordering of suffixes T_i and T_j can be decided deterministically in constant time by comparing $t_{i+LCP(i,j)}$ to $t_{j+LCP(i,j)}$. Therefore the problem reduces to checking the LCP of each pair of indices which are adjacent in the sparse suffix array. These LCPs are computed as a by-product of our Monte-Carlo algorithm, and there is a small probability that they are incorrect.

Therefore our focus in this section is on giving a deterministic algorithm which verifies the correctness of the answers to a batch of b LCP queries. As before, to do this we perform $O(\log b)$ rounds of computation. The rounds occur in decreasing order. In the k^{th} round the input is a set of (at most) b index pairs

to be verified. Let $\{x, y\}$ be such a pair of indices, corresponding to a pair of substrings $T_{x,x+m_k-1}$ and $T_{y,y+m_k-1}$ where $m_k = 2^k$. We say that $\{x, y\}$ *matches* iff $T_{x,x+m_k-1} = T_{y,y+m_k-1}$. In round k we will replace each pair $\{x, y\}$ with a new pair $\{x', y'\}$ to be inserted into round $(k-1)$ such that $T_{x,x+m_k-1} = T_{y,y+m_k-1}$ iff $T_{x',x'+m_{k-1}-1} = T_{y',y'+m_{k-1}-1}$. Each new pair will in fact always correspond to substrings of the old pair. In some cases we may choose to directly verify some $\{x, y\}$, in which case no new pair is inserted into the next round. The initial, largest value of k is the largest integer such that $m_k < n$. We perform $O(\log b)$ rounds, halting when $n/b < m_k < 2n/b$ after which point we can verify all pairs by scanning T in $O(m_k \cdot b) = O(n)$ time.

Of course an original query pair $\{x, y\}$ may not have $LCP(T_x, T_y) = m_k$ for any k. This is resolved by inserting two overlapping pairs into round k where $m_{k-1} < LCP(T_x, T_y) < m_k$. If the verifier succeeds, for each original pair we have that $T_{x,x+LCP(T_x,T_y)-1}$ equals $T_{y,y+LCP(T_x,T_y)-1}$. We also need to check that $t_{x+LCP(T_x,T_y)}$ does not equal $t_{x+LCP(T_x,T_y)}$ - otherwise the true LCP value is larger than was claimed. Where it is clear from context, for simplicity, we abuse notation by letting $m = m_k$. We now focus on an arbitrary round k.

The Suffix Implication Graph. We now build a graph (V, E) which will encode the structure in the text. We build the vertex set V greedily. Consider each text index $1 \leq x \leq n$ in ascending order. We include index x as a vertex in V iff it occurs in some pair $\{x, y\}$ (or $\{y, x\}$) and the last index included in V was at least $m/(9 \cdot \log b)$ characters ago. Observe that $|V| \leq 9 \cdot (n/m) \log b$ and also varies between $9 \cdot \log b \leq |V| \leq b$ as it contains at most one vertex per index pair.

Each pair of indices $\{x, y\}$ corresponds to an edge between vertices $v(x)$ and $v(y)$. Here $v(x)$ is the unique vertex such that $v(x) \leq x < v(x)+m/(9 \cdot \log b)$. The vertex $v(y)$ is defined analogously. This may create multiple edges between two vertices $v(x)$ and $v(y)$. Any multi-edges imply a two-cycle and can be handled first using a simplification of the main algorithm without increasing the overall time or space complexity. For brevity we omit this case and continue under the assumption that there are no multi-edges. It is simple to build the graph in $O(b \log b)$ time by traversing the pairs. As $|E| \leq b$ we can store the graph in $O(b)$ space.

We now discuss the structure of the graph constructed and show how it can be exploited to efficiently solve the problem. The following simple lemma will be essential to our algorithm and underpins the main arguments below.

Lemma 4. *Let (V', E') be a connected component of an undirected graph in which every vertex has degree at least three. There is a (simple) cycle in (V', E') of length at most $2 \log |V'| + 1$.*

The graph we have constructed may have vertices with degree less than three, preventing us from applying Lemma 4. For each vertex $v(x)$ with degree less than three, we verify every index pair $\{x, y\}$ (which corresponds to an edge $(v(x), v(y))$). By directly scanning the corresponding text portions this takes $O(|V|m)$ time. We can then safely remove all such vertices and the corresponding edges. This may introduce new low degree vertices which are then verified

iteratively in the same manner. As $|V| \leq 9 \cdot (n/m) \log b$, this takes a total of $O(n \log b)$ time. In the remainder we continue under the assumption that every vertex has degree at least three.

Algorithm Summary. The algorithm for round k processes each connected component separately. However the time complexity arguments will be amortized over all components. Consider a connected component (V', E'). As every vertex has degree at least three, any component has a short cycle of length at most $2 \log |V'| + 1 \leq 2 \log b + 1$ by Lemma 4. We begin by finding such a cycle in $O(b)$ time by performing a BFS of (V', E') starting at any vertex (this follows immediately from the proof of Lemma 4). Having located such a cycle, we will distinguish two cases. The first case is when the cycle is lock-stepped (defined below) and the other when it is unlocked. In both cases we will show below that we can exploit the structure of the text to safely delete an edge from the cycle, breaking the cycle. The index pair corresponding to the deleted edge will be replaced by a new index pair to be inserted into the next round where $m \leftarrow m_{k-1} = m_k/2$. Observe that both cases reduce the number of edges in the graph by one. Whenever we delete an edge we may reduce the degree of some vertex to below three. In this case we immediately directly process this vertex in $O(m)$ time as discussed above (iterating if necessary). As we do this at most once per vertex (and $O(|V|m) = O(n \log b)$), this does not increase the overall complexity. We then continue by finding and processing the next short cycle. The algorithm therefore searches for a cycle at most $|E| \leq b$ times over all components, contributing an $O(b^2)$ time additive term. In the remainder we will explain the two cycle cases in more detail and prove that summed over all components, the time complexity for round k is upper bounded by $O(n \log b)$ (excluding finding the cycles). As there are $O(\log b)$ rounds the final time complexity is $O(n \log^2 b + b^2 \log b)$ and the space is $O(b)$.

Cycles. We now define a lock-stepped cycle. Let $(v(x_i), v(y_i))$ for $i = 1 \ldots \ell$ be a cycle of length at most $2 \log b + 1$, i.e. $v(y_i) = v(x_{i+1})$ for $1 \leq i < \ell$ and $v(y_\ell) = v(x_1)$. Here $\{x_i, y_i\}$ for all i are the underlying text index pairs. Let $d_i = x_{i+1} - y_i$ for $1 \leq i < \ell$, $d_\ell = x_1 - y_\ell$ and let $\rho = \sum_{i=1}^{\ell} d_i$. We say that the cycle is *lock-stepped* iff $\rho = 0$ (and *unlocked* otherwise). Intuitively, lock-stepped cycles are ones where all the underlying pairs are in sync. Lemma 5 gives the key property of lock-stepped cycles which we will use.

Lemma 5. *Let $(v(x_i), v(y_i))$ for $i = 1 \ldots \ell$ be the edges in a lock-stepped cycle. Further let $j = \arg\max \sum_{i=1}^{j} d_j$. If $\{x_i, y_i\}$ match for all $i \neq j$ then $T_{x_j, x_j+m/2-1} = T_{y_j, y_j+m/2-1}$.*

Case 1: Lock-stepped Cycles. The first, simpler case is when we find a lock-stepped cycle in the connected component (V', E'). By Lemma 5, once we have found a lock-stepped cycle we can safely remove some single edge, $(v(x_j), v(y_j))$ from the cycle. When we remove a single edge, we still need to verify the right half of the removed pair, $\{x_j, y_j\}$. This is achieved by inserting a new pair, $\{x_j + m/2, y_j +$

$m/2\}$ into the next round where $m \leftarrow m_{k-1} = m_k/2$. We can determine which edge can be deleted by traversing the cycle in $O(\log b)$ time. Processing all lock-stepped cycles (over all components) takes $O(b \log b)$ time in total.

Case 2: Unlocked Cycles. The remaining case is when we find an unlocked cycle in the connected component (V', E'). Lemma 6 tells us that in this case (if all pairs match) then one of the pairs, $\{x_j, y_j\}$ corresponding to an edge, $(v(x_j), v(y_j))$ in the cycle must have a long, periodic prefix. We can again determine the suitable pair $\{x_j, y_j\}$ as well as ρ in $O(\log b)$ time by inspecting the cycle. This follows immediately from the statement of the lemma and the definition of ρ.

Lemma 6. *Assume that the connected component, (V', E') contains an unlocked cycle denoted, $(v(x_i), v(y_i))$ for $i = 1 \ldots \ell$. Further let $j = \arg\max \sum_{i=1}^{j} d_j$. If $\{x_i, y_i\}$ match for all $i = 1 \ldots \ell$ then $T_{x_j, x_j+3m/4-1}$ has a period $|\rho| \leq m/4$.*

Consider some pair $\{x, y\}$ such that both $T_{x,x+3m/4-1}$ and $T_{y,y+3m/4-1}$ are periodic with period at most $m/4$. We have that $T_{x,x+m-1} = T_{y,y+m-1}$ iff $T_{x+m/2,x+m-1} = T_{y+m/2,y+m-1}$. This is because $T_{x+m/2,x+m-1}$ contains at least a full period of characters from $T_{x,x+3m/4-1}$, and similarly with $T_{y+m/2,y+m-1}$ and $T_{y,y+3m/4-1}$ analogously. So we have that if $\{x_i, y_i\}$ match for all $i \neq j$ then the chosen pair $\{x_j, y_j\}$ matches iff both $T_{x_j,x_j+3m/4-1}$ and $T_{y_j,y_j+3m/4-1}$ have period $|\rho| \leq m/4$ and $T_{x_j+m/2,x_j+m-1} = T_{y_j+m/2,y_j+m-1}$. We can therefore delete the pair $\{x_j, y_j\}$ (and the corresponding edge, $(v(x_j), v(y_j))$) and insert a new pair, $\{x_j + m/2, y_j + m/2\}$ into the next round where $m \leftarrow m_{k-1} = m/2$.

However, for this approach to work we still need to verify that both strings $T_{x_j,x_j+3m/4-1}$ and $T_{y_j,y_j+3m/4-1}$ have $|\rho|$ as period. We do not immediately check the periodicity, we instead delay computation until the end of round k, after all cycles have been processed. At the current point in the algorithm, we simply add the tuple $(\{x, y\}, \rho)$ to a list, Π of text substrings to be checked later for periodicity. This takes $O(b)$ space over all components. Excluding checking for periodicity, processing all unlocked cycles takes $O(b \log b)$ time in total.

Checking for Substring Periodicity. The final task in round k is to scan the text and check that for each $(\{x, y\}, \rho) \in \Pi$, $|\rho|$ is a period of both $T_{x,x+3m/4-1}$ and $T_{y,y+3m/4-1}$. We process the tuples in left to right order. On the first pass we consider $T_{x,x+3m/4-1}$ for each $(\{x, y\}, \rho) \in \Pi$. In the second pass we consider y. The two passes are identical and we focus on the first.

We begin by splitting the tuples greedily into groups in left to right order. A tuple $(\{x, y\}, \rho)$ is in the same group as the previous tuple iff the previous tuple $(\{x', y'\}, \rho')$ has $x - x' \leq m/4$. Let $T_{z,z+m'-1}$ be the substring of T which spans every substring, $T_{x,x+3m/4-1}$ which appears in some $(\{x, y\}, \rho)$ in a single group of tuples. We now apply the classic periodicity lemma stated below.

Lemma 7 (see e.g. [8]). *Let S be a string with periods ρ_1 and ρ_2 and with $|S| > \rho_1 + \rho_2$. S has period $\gcd(\rho_1, \rho_2)$, the greatest common divisor of ρ_1 and ρ_2. Also, if S has period ρ_3 then S has period $\alpha \cdot \rho_3 \leq |S|$ for any integer $\alpha > 0$.*

First observe that consecutive tuples $(\{x, y\}, \rho)$ and $(\{x', y'\}, \rho')$ in the same group have overlap least $m/2 \geq |\rho| + |\rho'|$. Therefore by Lemma 7, if $T_{x,x+3m/4-1}$ has period $|\rho|$ and $T_{x',x'+3m/4-1}$ has period $|\rho'|$ then their overlap also has gcd $(|\rho|, |\rho'|)$ as a period. However as their overlap is longer than a full period in each string, both $T_{x,x+3m/4-1}$ and $T_{x',x'+3m/4-1}$ also have period $\gcd(|\rho|, |\rho'|)$. By repeat application of this argument we have that if for every tuple $(\{x, y\}, \rho)$, the substring $T_{x,x+3m/4-1}$ has period $|\rho|$ then $T_{z,z+m'-1}$ has a period equal to the greatest common divisor of the periods of all tuples in the group, denoted g. To process the entire group we can simply check whether $T_{z,z+m'-1}$ has period g in $O(m')$ time. If $T_{z,z+m'-1}$ does not have period g, we can safely abort the verifier. If $T_{z,z+m'-1}$ has period g then by Lemma 7, for each $(\{x, y\}, \rho)$ in the group, $T_{x,x+3m/4-1}$ has period $|\rho|$ as g divides $|\rho|$. As every $m' \geq 3m/4$ and the groups overlap by less than $m/2$ characters, this process takes $O(n)$ total time.

Theorem 3. *There exists a randomized Las-Vegas algorithm that correctly constructs the sparse suffix array and the sparse LCP array for any b suffixes from a string of length n. The algorithm uses $O(n \log^2 b + b^2 \log b)$ time with high probability and $O(b)$ space in the worst case.*

6 Time-Space Tradeoffs for Batched LCP Queries

We provide an overview of the techniques used to obtain the time-space tradeoff for the batched LCP process, as it closely follows those of Section 3. In Section 3 the algorithm simulates concurrent binary searches in order to determine the LCP of each input pair (with some extra work at the end). The idea for obtaining the tradeoff is to generalize the binary search to an α-ary search. So in the k^{th} round the input is a set of q pairs denoted by P_k, where we are guaranteed that for any $(i, j) \in P_k, LCP(i, j) \leq 2^{\log n-(k-1)\log \alpha}$, and the goal of the k^{th} iteration is to decide for any $(i, j) \in P_k$ if $LCP(i, j) \leq 2^{\log n-k \log \alpha}$ or not. From a space perspective, this means we need $O(\alpha q)$ space in order to compute α fingerprints for each index in any $(i, j) \in P_k$. From a time perspective, we only need to perform $O(\log_\alpha q)$ rounds before we may begin the final round. However, each round now costs $O(n + \alpha q)$, so we have the following trade-off.

Theorem 4. *Let $2 \leq \alpha \leq n$. There exists a randomized Monte-Carlo algorithm that with high probability correctly answers a batch of q LCP queries on suffixes from a string of length n. The algorithm uses $O((n + \alpha q) \log_\alpha q)$ time and $O(\alpha q)$ space in the worst case.*

In particular, for $\alpha = 2$, we obtain Theorem 1 as a corollary. Consequently, the total time cost for constructing the sparse suffix tree in $O(\alpha b)$ space becomes

$$O\left(n\frac{\log^2 b}{\log \alpha} + \frac{\alpha b \log^2 b}{\log \alpha}\right).$$

If, for example, $\alpha = b^\varepsilon$ for a small constant $\varepsilon > 0$, the cost for constructing the sparse suffix tree becomes $O(\frac{1}{\varepsilon}(n \log b + b^{1+\varepsilon} \log b))$, using $O(b^{1+\varepsilon})$ words of space. Finally by minimizing with the standard $O(n)$ time, $O(n)$ space algorithm we achieve the stated result of $O(n \log b)$ time, using $O(b^{1+\varepsilon})$ space.

References

1. Ajtai, M., Komlós, J., Szemerédi, E.: An $O(n \log n)$ Sorting Network. In: Proc. 15th STOC, pp. 1–9 (1983)
2. Andersson, A., Larsson, N.J., Swanson, K.: Suffix Trees on Words. In: Hirschberg, D.S., Meyers, G. (eds.) CPM 1996. LNCS, vol. 1075, pp. 102–115. Springer, Heidelberg (1996)
3. Andersson, A., Larsson, N.J., Swanson, K.: Suffix Trees on Words. Algorithmica 23(3), 246–260 (1999)
4. Batcher, K.E.: Sorting Networks and Their Applications. In: Proc. AFIPS Spring JCC, pp. 307–314 (1968)
5. Bentley, J.L., Sedgewick, R.: Fast algorithms for sorting and searching strings. In: Proc. 8th SODA, pp. 360–369 (1997)
6. Burkhardt, S., Kärkkäinen, J.: Fast Lightweight Suffix Array Construction and Checking. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) CPM 2003. LNCS, vol. 2676, pp. 55–69. Springer, Heidelberg (2003)
7. Ferragina, P., Fischer, J.: Suffix Arrays on Words. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 328–339. Springer, Heidelberg (2007)
8. Fine, N.J., Wilf, H.S.: Uniqueness Theorems for Periodic Functions. Proc. AMS 16(1), 109–114 (1965)
9. Inenaga, S., Takeda, M.: On-line linear-time construction of word suffix trees. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 60–71. Springer, Heidelberg (2006)
10. Kärkkäinen, J., Ukkonen, E.: Sparse Suffix Trees. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, pp. 219–230. Springer, Heidelberg (1996)
11. Karp, R.M., Rabin, M.O.: Efficient Randomized Pattern-Matching Algorithms. IBM J. Res. Dev. 31(2), 249–260 (1987)
12. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)
13. Kolpakov, R., Kucherov, G., Starikovskaya, T.A.: Pattern Matching on Sparse Suffix Trees. In: Proc. 1st CCP, pp. 92–97 (2011)
14. Manber, U., Myers, G.: Suffix Arrays: A New Method for On-Line String Searches. SIAM J. Comput. 22(5), 935–948 (1993)
15. Morrison, D.R.: Patricia-practical algorithm to retrieve information coded in alphanumeric. J. ACM 15(4), 514–534 (1968)
16. Paterson, M.: Improved Sorting Networks with $O(\log N)$ Depth. Algorithmica 5(1), 65–92 (1990)
17. Uemura, T., Arimura, H.: Sparse and truncated suffix trees on variable-length codes. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 246–260. Springer, Heidelberg (2011)
18. Weiner, P.: Linear Pattern Matching Algorithms. In: Proc. 14th FOCS (SWAT), pp. 1–11 (1973)

Tree Compression with Top Trees*

Philip Bille[1,**], Inge Li Gørtz[1,**], Gad M. Landau[2,***], and Oren Weimann[2,**]

[1] Technical University of Denmark, DTU Compute
{phbi,ilg}@dtu.dk
[2] University of Haifa
{oren,landau}@cs.haifa.ac.il

Abstract. We introduce a new compression scheme for labeled trees based on top trees [3]. Our compression scheme is the first to simultaneously take advantage of internal repeats in the tree (as opposed to the classical DAG compression that only exploits rooted subtree repeats) while also supporting fast navigational queries directly on the compressed representation. We show that the new compression scheme achieves close to optimal worst-case compression, can compress exponentially better than DAG compression, is never much worse than DAG compression, and supports navigational queries in logarithmic time.

1 Introduction

A labeled tree T is a rooted ordered tree with n nodes where each node has a label from an alphabet Σ. Many classical applications of trees in computer science (such as tries, dictionaries, parse trees, suffix trees, and XML databases) generate navigational queries on labeled trees (e.g, returning the label of node v, the parent of v, the depth of v, the size of v's subtree, etc.). In this paper we present new and simple compression scheme that support such queries directly on the compressed representation.

While a huge literature exists on string compression, labeled tree compression is much less studied. The simplest way to compress a tree is to serialize it using, say, preorder traversal to get a string of labels to which string compression can be applied. This approach is fast and is used in practice, but it does not support the various navigational queries. Furthermore, it does not capture possible repeats contained in the tree structure.

To get a sublinear space representation for trees with many repeated substructures (such as XML databases), one needs to define "repeated substructures" and devise an algorithm that identifies such repeats and collapses them (like Lempel-Ziv does to strings). There have been two main ways to define repeats: *subtree*

* A draft of the full version of the paper can be found as Arxiv preprint arXiv:1304.5702.

** Partially supported by the Danish Agency for Science, Technology and Innovation.

*** Partially supported by the National Science Foundation Award 0904246, Israel Science Foundation grant 347/09, Yahoo, Grant No. 2008217 from the United States-Israel Binational Science Foundation (BSF) and DFG.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 160–171, 2013.

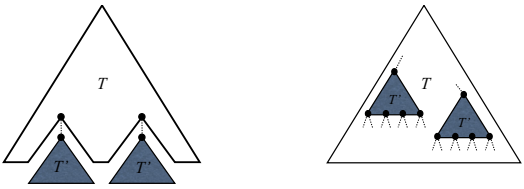


Fig. 1. A tree T with a *subtree* repeat T' (left), and a *tree pattern* repeat T' (right)

repeats and the more general *tree pattern repeats* (see Fig. 1). A *subtree repeat* is an identical (both in structure and in labels) occurrence of a *rooted* subtree in T. A *tree pattern repeat* is an identical (both in structure and in labels) occurrence of any *connected subgraph* of T. Subtree repeats are used in DAG compression [8,14] and tree patterns repeats in tree grammars [9,10,17–19]. In this paper we introduce *top tree compression* based top trees [3] that exploits tree pattern repeats. Compared to the existing techniques our compression scheme has the following advantages: Let T be a tree of size n with nodes labeled from an alphabet of size σ. We support navigational queries in $O(\log n)$ time (a similar result is not known for tree grammars), the compression ratio is in the worst case at least $\log_\sigma^{0.19} n$ (no such result is known for neither DAG compression or tree grammars), our scheme can compress exponentially better than DAG compression, and is never worse than DAG compression by more than a $\log n$ factor.

Previous Work. Using subtree repeats, a node in the tree T that has a child with subtree T' can instead point to any other occurrence of T'. This way, it is possible to represent T as a directed acyclic graph (DAG). Over all possible DAGs that can represent T, the smallest one is unique and can be computed in $O(n)$ time [12]. Its size can be exponentially smaller than n. Using subtree repeats for compression was studied in [8,14], and a Lempel-Ziv analog of subtree repeats was suggested in [1]. It is also possible to support navigational queries [7] and path queries [8] directly on the DAG representation in logarithmic time.

The problem with subtree repeats is that we can miss many internal repeats. Consider for example the case where T is a single path of n nodes with the same label. Even though T is highly compressible (we can represent it by just storing the label and the length of the path) it does not contain a single subtree repeat and its minimal DAG is of size n.

Alternatively, *tree grammars* are capable of exploiting tree pattern repeats. Tree grammars generalize grammars from deriving strings to deriving trees and were studied in [9,10,17–19]. Compared to DAG compression a tree grammar can be exponentially smaller than the minimal DAG [17]. Unfortunately, computing a minimal tree grammar is NP-Hard [11], and all known tree grammar based compression schemes can only support navigational queries in time proportional to the height of the grammar which can be $\Omega(n)$.

Our Results. We propose a new compression scheme for labeled trees, which we call *top tree compression*. To the best of our knowledge, this is the first

compression scheme for trees that (i) takes advantage of tree pattern repeats (like tree grammars) but (ii) simultaneously supports navigational queries on the compressed representation in logarithmic time (like DAG compression). In the worst case, we show that (iii) the compression ratio of top tree compression is always at least $\log_\sigma^{0.19} n$ (compared to the information-theoretic lower bound of $\log_\sigma n$). This is in contrast to both tree grammars and DAG compression that do not have good provable worst-case compression performance. Finally, we compare the performance of top tree compression to DAG compression. We show that top tree compression (iv) can compress exponentially better than DAG compression, and (v) is never much worse than DAG compression.

With these features, top tree compression significantly improves the state-of-the-art for tree compression. Specifically, it is the first scheme to simultaneously achieve (i) and (ii) and the first scheme based on either subtree repeats or tree pattern repeats with provable good compression performance compared to worst-case (iii) or the DAG (iv).

The key idea in top tree compression is to transform the input tree T into another tree $\mathcal{T}$ such that tree pattern repeats in T become subtree repeats in $\mathcal{T}$. The transformation is based on top trees [2–4] – a data structure originally designed for dynamic (uncompressed) trees. After the transformation, we compress the new tree $\mathcal{T}$ using the classical DAG compression resulting in the *top DAG* $\mathcal{TD}$. The top DAG $\mathcal{TD}$ forms the basis for our compression scheme. We obtain our bounds on compression (iii), (iv), and (v) by analyzing the size of $\mathcal{TD}$, and we obtain efficient navigational queries (ii) by augmenting $\mathcal{TD}$ with additional data structures.

To state our bounds, let n_G denote the total size (vertices plus edges) of the graph G. We first show the following worst-case compression bound achieved by the top DAG.

Theorem 1. *Let T be any ordered tree with nodes labeled from an alphabet of size σ and let $\mathcal{TD}$ be the corresponding top DAG. Then, $n_{\mathcal{TD}} = O(n_T / \log_\sigma^{0.19} n_T)$.*

This worst-case performance of the top DAG should be compared to the optimal information-theoretic lower bound of $\Omega(n_T / \log_\sigma n_T)$. Note that with standard DAG compression the worst-case bound is $O(n_T)$ since a single path is incompressible using subtree repeats.

Secondly, we compare top DAG compression to standard DAG compression.

Theorem 2. *Let T be any ordered tree and let D and $\mathcal{TD}$ be the corresponding DAG and top DAG, respectively. For any tree T we have $n_{\mathcal{TD}} = O(\log n_T) \cdot n_D$ and there exist families of trees T such that $n_D = \Omega(n_T / \log n_T) \cdot n_{\mathcal{TD}}$.*

Thus, top DAG compression can be exponentially better than DAG compression and it is always within a logarithmic factor of DAG compression. To the best of our knowledge this is the first non-trivial bound shown for any tree compression scheme compared to the DAG.

Finally, we show how to represent the top DAG $\mathcal{TD}$ in $O(n_{\mathcal{TD}})$ space such that we can quickly answer a wide range of queries about T without decompressing.

Theorem 3. *Let T be an ordered tree with top DAG $\mathcal{TD}$. There is an $O(n_{\mathcal{TD}})$ space representation of T that supports* Access, Depth, Height, Size, Parent, Firstchild, NextSibling, LevelAncestor, *and* NCA *in $O(\log n_T)$ time. Furthermore, we can* Decompress *a subtree T' of T in time $O(\log n_T + |T'|)$.*

The Access, Depth, Height, Size, Parent, Firstchild, and NextSibling all take a node v in T as input and return its label, its depth, its height, the size of its subtree, its parent, its first child, and its immediate sibling, respectively. The LevelAncestor returns an ancestor at a specified distance from v, and NCA returns the nearest common ancestor to a given pair of nodes. Finally, the Decompress operation decompresses and returns any rooted subtree.

Related Work (Succinct Data Structures). Jacobson [16] was the first to observe that the naive pointer-based tree representation using $\Theta(n \log n)$ bits is wasteful. He showed that *unlabeled* trees can be represented using $2n + o(n)$ bits and support various queries by inspection of $\Theta(\lg n)$ bits in the bit probe model. This space bound is asymptotically optimal with the information-theoretic lower bound averaged over all trees. Munro and Raman [20] showed how to achieve the same bound in the RAM model while using only constant time for queries. Such representations are called *succinct data structures*, and have been generalized to trees with higher degrees [5] and to a richer set of queries such as subtree-size queries [20] and level-ancestor queries [15]. For *labeled* trees, Ferragina et al. [13] gave a representation using $2n \log \sigma + O(n)$ bits that supports basic navigational operations, such as find the parent of node v, the i'th child of v, and any child of v with label α.

All the above bounds for space are averaged over all trees and do not take advantage of the cases where the input tree contains many repeated substructures. The focus of this paper is achieving sublinear bounds in trees with many repeated substructures (i.e., highly compressible trees).

2 Top Trees and Top DAGs

Top trees were introduced by Alstrup et al. [2–4] for maintaining an uncompressed, unordered, and unlabeled tree under link and cut operations. We extend them to ordered and labeled trees, and then introduce top DAGs for compression. Our construction is related to well-known algorithms for top tree construction, but modified for our purposes. In particular, we need to carefully order the steps of the construction to guarantee efficient compression, and we disallow some combination of cluster merges to ensure fast navigation.

Clusters. Let v be a node in T with children $v_1, \ldots, v_k$ in left-to-right order. Define $T(v)$ to be the subtree induced by v and all proper descendants of v. Define $F(v)$ to be the forest induced by all proper descendants of v. For $1 \leq s \leq r \leq k$ let $T(v, v_s, v_r)$ be the tree pattern induced by the nodes $\{v\} \cup T(v_s) \cup T(v_{s+1}) \cup \cdots \cup T(v_r)$.

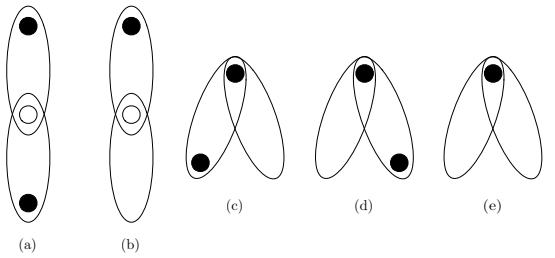


Fig. 2. Five ways of merging clusters. The • nodes are boundary nodes that remain boundary nodes in the merged cluster. The ∘ nodes are boundary nodes that become internal (non-boundary) nodes in the merged cluster.

A *cluster* with *top boundary node* v is a tree pattern of the form $T(v, v_s, v_r)$, $1 \leq s \leq r \leq k$. A *cluster* with *top boundary node* v and *bottom boundary node* u is a tree pattern of the form $T(v, v_s, v_r) \setminus F(u)$, $1 \leq s \leq r \leq k$, where u is a node in $T(v_s) \cup \cdots \cup T(v_r)$. Clusters can therefore have either one or two boundary nodes. For example, let $p(v)$ denote the parent of v then a single edge $(v, p(v))$ of T is a cluster where $p(v)$ is the top boundary node. If v is a leaf then there is no bottom boundary node, otherwise v is a bottom boundary node.

Two edge disjoint clusters A and B whose vertices overlap on a single boundary node can be *merged* if their union $C = A \cup B$ is also a cluster. There are five ways of merging clusters, as illustrated by Fig. 2. The original paper on top trees [2–4] contains more ways to merge clusters, but allowing these would lead to a violation of our definition of clusters as a tree pattern of the form $T(v, v_s, v_r) \setminus F(u)$, which we need for navigational purposes.

Top Trees. A *top tree* $\mathcal{T}$ of T is a hierarchical decomposition of T into clusters. It is an ordered, rooted, and binary tree and is defined as follows.

- The nodes of $\mathcal{T}$ correspond to clusters of T.
- The root of $\mathcal{T}$ is the cluster T itself.
- The leaves of $\mathcal{T}$ correspond to the edges of T. The label of each leaf is the pair of labels of the endpoints of the edges in T.
- Each internal node of $\mathcal{T}$ is a merged cluster of its two children. The label of each internal node is the type of merge it represents (out of the five merging options). The children are ordered so that the left child is the child cluster visited first in a preorder traversal of T.

Constructing the Top Tree. We now describe a greedy algorithm for constructing a top tree $\mathcal{T}$ of T that has height $O(\log n_T)$. The algorithm constructs the top tree $\mathcal{T}$ bottom-up in $O(\log n_T)$ iterations starting with the edges of T as the leaves of $\mathcal{T}$. During the construction, we maintain an auxiliary tree $\widetilde{T}$ initialized as $\widetilde{T} := T$. The edges of $\widetilde{T}$ will correspond to the nodes of $\mathcal{T}$ and to the clusters of T. In the beginning, these clusters represent actual edges $(v, p(v))$

of T. In this case, if v is not a leaf in T then v is the bottom boundary node of the cluster and $p(v)$ is the top boundary node. If v is a leaf then there is no bottom boundary node.

In each one of the $O(\log n_T)$ iterations, a constant fraction of $\widetilde{T}$'s edges (i.e., clusters of T) are merged. Each merge is performed on two overlapping edges (u, v) and (v, w) of $\widetilde{T}$ using one of the five types of merges from Fig. 2: If v is the parent of u and the only child of w then a merge of type (a) or (b) contracts these edges in $\widetilde{T}$ into the edge (u, w). If v is the parent of both u and w, and w or u are leaves, then a merge of type (c), (d), or (e) replaces these edges in $\widetilde{T}$ with either the edge (u, v) or (v, w). In all cases, we create a new node in $\mathcal{T}$ whose two children are the clusters corresponding to (u, v) and to (v, w).

This way, we get that a single iteration shrinks the tree $\widetilde{T}$ (and the number of parentless nodes in $\mathcal{T}$) by a constant factor. The process ends when $\widetilde{T}$ is a single edge. Each iteration is performed as follows:

Step 1: Horizontal Merges. For each node $v \in \widetilde{T}$ with $k \geq 2$ children $v_1, \ldots, v_k$, for $i = 1$ to $\lfloor k/2 \rfloor$, merge the edges (v, v_{2i-1}) and (v, v_{2i}) if v_{2i-1} or v_{2i} is a leaf. If k is odd and v_k is a leaf and both v_{k-2} and v_{k-1} are non-leaves then also merge (v, v_{k-1}) and (v, v_k).

Step 2: Vertical Merges. For each maximal path $v_1, \ldots, v_p$ of nodes in $\widetilde{T}$ such that v_{i+1} is the parent of v_i and $v_2, \ldots, v_{p-1}$ have a single child: If p is even merge the following pairs of edges $\{(v_1, v_2), (v_2, v_3)\}, \{(v_3, v_4), (v_4, v_5)\}, \ldots, (v_{p-2}, v_{p-1})\}$. If p is odd merge t$\{(v_1, v_2), (v_2, v_3)\}, \{(v_3, v_4), (v_4, v_5)\}, \ldots, (v_{p-3}, v_{p-2})\}$, and if (v_{p-1}, v_p) was not merged in Step 1 then also merge $\{(v_{p-2}, v_{p-1}), (v_{p-1}, v_p)\}$.

Lemma 1. *A single iteration shrinks $\widetilde{T}$ by a factor of $c \geq 8/7$.*

Proof. Suppose that in the beginning of the iteration the tree $\widetilde{T}$ has n nodes. Any tree with n nodes has at least $n/2$ nodes with less than 2 children. Consider the edges $(v_i, p(v_i))$ of $\widetilde{T}$ where v_i has one or no children. We show that at least half of these $n/2$ edges are merged in this iteration. This will imply that $n/4$ edges of $\widetilde{T}$ are replaced with $n/8$ edges and so the size of $\widetilde{T}$ shrinks to $7n/8$. To prove it, we charge each edge $(v_i, p(v_i))$ that is not merged to a unique edge $f(v_i, p(v_i))$ that is merged.

Case 1. Suppose that v_i has no children (i.e., is a leaf). If v_i has at least one sibling and $(v_i, p(v_i))$ is not merged it is because v_i has no right sibling and its left sibling v_{i-1} has already been merged (i.e., we have just merged $(v_{i-2}, p(v_{i-2}))$ and $(v_{i-1}, p(v_{i-1}))$ in Step 1 where $p(v_i) = p(v_{i-1}) = p(v_{i-2})$). We also know that at least one of v_{i-1} and v_{i-2} must be a leaf. We set $f(v_i, p(v_i)) = (v_{i-1}, p(v_{i-1}))$ if v_{i-1} is a leaf, otherwise we set $f(v_i, p(v_i)) = (v_{i-2}, p(v_{i-2}))$.

Case 2. Suppose that v_i has no children (i.e., is a leaf) and no siblings (i.e., $p(v_i)$ has only one child). The only reason for not merging $(v_i, p(v_i))$ with $(p(v_i), p(p(v_i)))$ in Step 2 is because $(p(v_i), p(p(v_i)))$ was just merged in Step 1. In this case, we set $f(v_i, p(v_i)) = (p(v_i), p(p(v_i)))$. Notice that we haven't already charged $(p(v_i), p(p(v_i))$ in *Case 1* because $p(v_i)$ is not a leaf.

Case 3. Suppose that v_i has exactly one child $c(v_i)$ and that $(v_i, p(v_i))$ was not merged in Step 1. The only reason for not merging $(v_i, p(v_i))$ with $(c(v_i), v_i)$ in Step 2 is if $c(v_i)$ has only one child $c(c(v_i))$ and we just merged $(c(v_i), v_i)$ with $(c(c(v_i)), c(v_i))$. In this case, we set $f(v_i, p(v_i)) = (c(v_i), v_i)$. Notice that we haven't already charged $(c(v_i), v_i)$ in *Case 1* because $c(v_i)$ is not a leaf. We also haven't charged $(c(v_i), v_i)$ in *Case 2* because v_i has only one child. □

Corollary 1. *Given a tree T, the greedy top tree construction creates a top tree of size $O(n_T)$ and height $O(\log n_T)$ in $O(n_T)$ time.*

The next lemma follows from the construction of the top tree and Lemma 1.

Lemma 2. *For any node c in the top tree corresponding to a cluster C of T, the total size of all clusters corresponding to nodes in the subtree $\mathcal{T}(c)$ is $O(|C|)$.*

Top Dags. The *top DAG* of T, denoted $\mathcal{TD}$, is the minimal DAG representation of the top tree $\mathcal{T}$. It can be computed in $O(n_T)$ time from $\mathcal{T}$ using the algorithm of [12]. The entire top DAG construction can thus be done in $O(n_T)$ time.

3 Compression Analysis

Worst-Case Bounds for Top Dag Compression. We now prove Theorem 1.

Let $\mathcal{T}$ be the top tree for T. Identical clusters in T are represented by identical complete subtrees in $\mathcal{T}$. Since identical subtrees in $\mathcal{T}$ are shared in $\mathcal{TD}$ we have the following lemma.

Lemma 3. *For any tree T, all clusters in the corresponding top DAG $\mathcal{TD}$ are distinct.*

Lemma 4. *Let T be any tree with n_T nodes labeled from an alphabet of size σ and let $\mathcal{T}$ be its top tree. The nodes of $\mathcal{T}$ correspond to at most $O(n_T / \log_\sigma^{0.19} n_T)$ distinct clusters in T.*

Proof. Consider the bottom-up construction of the top tree $\mathcal{T}$ starting with the leaves of $\mathcal{T}$ (the clusters corresponding to the edges of T). By Lemma 1 each level in the top tree reduces the number of clusters by a factor $c = 8/7$, while at most doubling the size of the current clusters. After round i we are therefore left with at most $O(n_T/c^i)$ clusters, each of size at most $2^i + 1$.

To bound the total number of distinct cluster, we partition the clusters into *small clusters* and *large clusters*. The small clusters are those created in rounds 1

to $j = \log_2(0.5 \log_{4\sigma} n_T)$ and the large clusters are those created in the remaining rounds from $j+1$ to h. The total number of large clusters is at most

$$\sum_{i=j+1}^{h} O(n_T/c^i) = O(n_T/c^{j+1}) = O(n_T/\log_\sigma^{0.19} n_T).$$

In particular, there are at most $O(n_T/\log_\sigma^{0.19} n_T)$ distinct clusters among these.

Next, we bound the total number of distinct small clusters. Each small cluster corresponds to a connected subgraph (i.e., a tree pattern) of T that is of size at most $2^j + 1$ and is an ordered and labeled tree. The total number of distinct ordered and labeled trees of size at most x is given by

$$\sum_{i=1}^{x} \sigma^i C_{i-1} = \sum_{i=1}^{x} \frac{\sigma^i}{i} \binom{2i-1}{i-1} = \sum_{i=1}^{x} O(4^i \sigma^i) = O\left((4\sigma)^{x+1}\right),$$

where C_i denotes the ith Catalan number. Hence, the total number of distinct small clusters is bounded by $O((4\sigma)^{2^j+2}) = O(\sigma^2\sqrt{n_T}) = O(n_T^{3/4})$. In the last equality we used the fact that $\sigma < n_T^{1/8}$. If $\sigma > n_T^{1/8}$ then the lemma trivially holds because $O(n_T/(\log_\sigma^{0.19} n_T)) = O(n_T)$. We get that the total number of distinct clusters is at most $O(n_T/\log_\sigma^{0.19} n_T + n_T^{3/4}) = O(n_T/\log_\sigma^{0.19} n_T)$. □

Combining Lemma 3 and 4 we obtain Theorem 1.

Comparison to Subtree Sharing. We now prove Theorem 2. To do so we first show two useful properties of top trees and top dags.

Let T be a tree with top tree $\mathcal{T}$. For any internal node z in T, we say that the subtree $T(z)$ is *represented* by a set of clusters $\{C_1, \ldots, C_\ell\}$ from $\mathcal{T}$ if $T(z) = C_1 \cup \cdots \cup C_\ell$. Since each edge in T is a cluster in $\mathcal{T}$ we can always trivially represent $T(z)$ by at most $|T(z)| - 1$ clusters. We prove that there always exists a set of clusters, denoted S_z, of size $O(\log n_T)$ that represents $T(z)$.

Let z be any internal node in T and let z_1 be its leftmost child. Since z is internal we have that z is the top boundary node of the leaf cluster $L = (z, z_1)$ in $\mathcal{T}$. Let U be the smallest cluster in $\mathcal{T}$ containing all nodes of $T(z)$. We have that L is a descendant leaf of U in $\mathcal{T}$. Consider the path P of cluster in $\mathcal{T}$ from U to L. An *off-path* cluster of P is a cluster C that is not on P, but whose parent cluster is on P. We define

$$S_z = \{C \mid C \text{ is off-path cluster of } P \text{ and } C \subseteq T(z)\} \cup \{L\}$$

Since the length of P is $O(\log n_T)$ the number of clusters in S_z is $O(\log n_T)$. We need to prove that $\cup_{C \in S_z} C = T(z)$. By definition we have that all nodes in $\cup_{C \in S_z} C$ are in $T(z)$. For the other direction, we first prove the following lemma. Let $E(C)$ denote the set of edges of a cluster C.

Lemma 5. *Let C be an off-path cluster of P. Then either $E(C) \subseteq E(T(z))$ or $E(C) \cap E(T(z)) = \emptyset$.*

Proof. We will show that any cluster in $\mathcal{T}$ containing edges from both $T(z)$ and $T \setminus T(z)$ contains both $(p(z), z)$ and (z, z_1), where z_1 is the leftmost child of z and $p(z)$ is the parent of z. Let C be a cluster containing edges from both $T(z)$ and $T \setminus T(z)$. Consider the subtree $\mathcal{T}(C)$ and let C' be the smallest cluster containing edges from both $T(z)$ and $T \setminus T(z)$. Then C' must be a merge of type (a) or (b), where the higher cluster A only contains edges from $T \setminus T(z)$ and the bottom cluster, B, only contains edges from $T(z)$. Also, z is the top boundary node of B and the bottom boundary node of A. Clearly, A contains the edge $(p(z), z)$, since all clusters are connected tree patterns. A merge of type (a) or (b) is only possible when B contains all children of its top boundary node. Thus B contains the edge (z, z_1).

We have $L = (z, z_1)$ and therefore all clusters in $\mathcal{T}$ containing (z, z_1) lay on the path from L to the root. The path P is a subpath of this path, and thus no off-path clusters of P can contain (z, z_1). Therefore no off-path clusters of P can contain edges from both $T(z)$ and $T \setminus T(z)$. □

Any edge from $T(z)$ (except (z, z_1)) contained in a cluster on P must be contained in an off-path cluster of P. Lemma 5 therefore implies that $T(z) = \cup_{C \in S_z} C$ and the following corollary.

Corollary 2. *Let T be a tree with top tree $\mathcal{T}$. For any node z in T, the subtree $T(z)$ can be represented by a set of $O(\log n_T)$ clusters in $\mathcal{T}$.*

Next we prove that our bottom-up top tree construction guarantees that two identical subtrees $T(z), T(z')$ are represented by two *identical* sets of clusters $S_z, S_{z'}$. Two sets of clusters are identical (denoted $S_z = S_{z'}$) when $C \in S_z$ iff $C' \in S_z$ such that C and C' are clusters corresponding to tree patterns in T that have the same structure and labels.

Lemma 6. *Let T be a tree with top tree $\mathcal{T}$. Let $T(z)$ and $T(z')$ be identical subtrees in T and let S_z and $S_{z'}$ be the corresponding representing set of clusters in $\mathcal{T}$. Then, $S_z = S_{z'}$.*

Proof. Omitted due to lack of space.

Theorem 4. *For any tree T, $n_{\mathcal{TD}} = O(\log n_T) \cdot n_D$.*

Proof. An edge is shared in the DAG if it is in a shared subtree of T. We denote the edges in the DAG D that are shared as *red* edges, and the edges that are not shared as *blue*. Let r_D and b_D be the number of red and blue edges in the DAG D, respectively.

A cluster in the top tree $\mathcal{T}$ is *red* if it only contains red edges from D, *blue* if it only contains blue edges from D, and *purple* if it contains both. Since clusters are connected subtrees we have the property that if cluster C is red (resp. blue), then all clusters in the subtree $\mathcal{T}(C)$ are red (resp. blue). Let r, b, and p be the number of red, blue, and purple clusters in $\mathcal{T}$, respectively. Since $\mathcal{T}$ is a binary

tree, where all internal nodes have 2 children and all leaves are either red or blue, we have $p \leq r + b$. It is thus enough to bound the number of red and blue clusters.

First we bound the number of red clusters in the DAG $\mathcal{TD}$. Consider a shared subtree $T(z)$ from the DAG compression. $T(z)$ is represented by at most $O(\log n_T)$ clusters in $\mathcal{T}$, and all these contain only edges from $T(z)$. It follows from Lemma 6 that all the clusters representing $T(z)$ (and their subtrees in $\mathcal{T}$) are identical for all copies of $T(z)$. Therefore each of these will appear only once in the top DAG $\mathcal{TD}$.

From Corollary 2 we have that each of the clusters representing $T(z)$ has size at most $O(|T(z)|)$. Thus, the total size of the subtrees of the clusters representing $T(z)$ is $O(|T(z)| \log n_T)$. This is true for all shared subtrees, and thus $r = O(r_D \log n_T)$.

To bound the number of blue clusters in the top DAG, we first note that the blue clusters form rooted subtrees in the top tree. Let C be the root of such a blue subtree in $\mathcal{T}$. Then C is a connected component of blue edges in T. It follows from Corollary 2 that $|\mathcal{T}(C)| = O(|C|)$. Thus the number of blue clusters $b = O(b_D)$. The number of edges in the $\mathcal{TD}$ is thus $b + r + p \leq 2(b + r) = O(b_D + r_D \log n_T) = O(n_D \log n_T)$. □

Lemma 7. *There exist trees T, such that $n_D = \Omega(n_T / \log n_T) \cdot n_{\mathcal{TD}}$.*

Proof. Caterpillars and paths have $n_{\mathcal{TD}} = O(\log n_T)$, whereas $n_D = n_T$. □

4 Supporting Navigational Queries

In this section we give a high-level sketch of how to perform the navigational queries. All details can be found in the full version [6]. Let T be a tree with top DAG $\mathcal{TD}$. To uniquely identify nodes of T we refer to them by their preorder numbers. For a node of T with preorder number x we want to support the following queries.

Access(x)**:** Return the label associated with node x.
Decompress(x)**:** Return the tree $T(x)$.
Parent(x)**:** Return the parent of node x.
Depth(x)**:** Return the depth of node x.
Height(x)**:** Return the height of node x.
Size(x)**:** Return the number of nodes in $T(x)$.
Firstchild(x)**:** Return the first child of x.
NextSibling(x)**:** Return the sibling immediately to the right of x.
LevelAncestor(x, i)**:** Return the ancestor of x whose distance from x is i.
NCA(x, y)**:** Return the nearest common ancestor of the nodes x and y.

The Data Structure. In order to enable the above queries, we augment the top DAG $\mathcal{TD}$ of T with some additional information. Consider a cluster C in $\mathcal{TD}$. Recall that if C is a leaf in $\mathcal{TD}$ then C is a single edge in T and C stores the labels of this edge's endpoints. For each internal cluster C in $\mathcal{TD}$ we save information about which type of merge that was used to construct C, the height and size of the tree pattern C, and the distance between the top of C to the top boundary nodes of its child clusters. From the definition of clusters and tree patterns it follows that the preorder numbers in T of all the nodes (except possibly the root) in a cluster/tree pattern C are either a consecutive sequence $[i_1, \ldots, i_r]$ or two consecutive sequences $[i_1, \ldots, i_r], [i_s, \ldots, i_t]$, where $r < s$. In the second case the nodes with preorder numbers $[i_{r+1}, \ldots, i_{s-1}]$ are nodes in the subtree rooted at the bottom boundary node of C. We augment our data structure with information that allows us to compute the sequence(s) of preorder numbers in C in constant time.

All of our queries are based on traversals of the augmented top DAG $\mathcal{TD}$. During the traversal we identify nodes by computing preorder numbers local to the cluster that we are currently visiting. The intervals of local preorder numbers can be computed in constant time for each cluster.

Our data structures uses constant space for each cluster of $\mathcal{TD}$, and thus the total space remains $O(n_{\mathcal{TD}})$.

Navigation. The navigational queries can be grouped into three types. Access and Depth are computed by a single top-down search of $\mathcal{TD}$ starting from its root and ending with the leaf cluster containing x. At each step the local preorder number of x is computed, along with some additional information in the case of Depth. The procedures Firstchild, LevelAncestor, Parent, and NCA are computed by a top-down search to find the local preorder number in a relevant cluster C, and then a bottom-up search to compute the corresponding preorder number in T. Since the depth of $\mathcal{TD}$ is $O(\log n_T)$ and the computation in each cluster of $\mathcal{TD}$ takes constant time the total time for each operation is $O(\log n_T)$.

To answer the queries Decompress, Height, Size, and NextSibling the key idea is to compute a small set of clusters representing $T(x)$ and all necessary information needed to answer the queries. The set is a subset denoted $\check{S}_x$ of the set S_x, where S_x is the set of $O(\log n_T)$ off-path clusters of P that represents $T(x)$ as defined in Section 3. Let M be the highest cluster on P that only contains edges from $T(x)$ and partition S_x into the set $\hat{S}_x$ that contains all clusters in S_x that are descendants of M and the set $\check{S}_x$ that contains the remaining clusters. The desired subset is the set $\check{S}_x$. We then show how to implement a procedure FindRepresentatives that efficiently computes $\check{S}_x$ in $O(\log n_T)$ time. This is done by a top-down traversal of $\mathcal{TD}$ to find M followed by a bottom-up traversal that exploits the structural properties of the top tree. Note that not all off-path clusters of P are in S_x. Therefore we carefully need to consider for each off-path cluster in the bottom-up traversal, if it is in S_x or not. With FindRepresentatives, the remaining procedures are straightforward to implement using the information from the clusters in $\check{S}_x$.

References

1. Adiego, J., Navarro, G., de la Fuente, P.: Lempel-Ziv compression of highly structured documents. J. Amer. Soc. Inf. Sci. and Techn. 58(4), 461–478 (2007)
2. Alstrup, S., Holm, J., de Lichtenberg, K., Thorup, M.: Minimizing diameters of dynamic trees. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 270–280. Springer, Heidelberg (1997)
3. Alstrup, S., Holm, J., Lichtenberg, K.D., Thorup, M.: Maintaining information in fully-dynamic trees with top trees. ACM Trans. Algorithms 1, 243–264 (2003)
4. Alstrup, S., Holm, J., Thorup, M.: Maintaining center and median in dynamic trees. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 46–56. Springer, Heidelberg (2000)
5. Benoit, D., Demaine, E., Munro, I., Raman, R., Raman, V., Rao, S.: Representing trees of higher degree. Algorithmica 43, 275–292 (2005)
6. Bille, P., Gørtz, I.L., Landau, G.M., Weimann, O.: Tree compression with top trees. Arxiv preprint arXiv:1304.5702 (2013)
7. Bille, P., Landau, G., Raman, R., Rao, S., Sadakane, K., Weimann, O.: Random access to grammar-compressed strings. In: Proc. 22nd SODA, pp. 373–389 (2011)
8. Buneman, P., Grohe, M., Koch, C.: Path queries on compressed XML. In: Proc. 29th VLDB, pp. 141–152 (2003)
9. Busatto, G., Lohrey, M., Maneth, S.: Grammar-based tree compression. Technical report, EPFL (2004)
10. Busatto, G., Lohrey, M., Maneth, S.: Efficient memory representation of XML document trees. Information Systems 33(4-5), 456–474 (2008)
11. Charikar, M., Lehman, E., Lehman, A., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Trans. Inform. Theory 51(7), 2554–2576 (2005)
12. Downey, P.J., Sethi, R., Tarjan, R.E.: Variations on the common subexpression problem. J. ACM 27, 758–771 (1980)
13. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and indexing labeled trees, with applications. J. ACM 57, 1–33 (2009)
14. Frick, M., Grohe, M., Koch, C.: Query evaluation on compressed trees. In: Proc. 18th LICS, pp. 188–197 (2003)
15. Geary, R., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. In: Proc. 15th SODA, pp. 1–10 (2004)
16. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th FOCS, pp. 549–554 (1989)
17. Lohrey, M., Maneth, S.: The complexity of tree automata and XPath on grammar-compressed trees. Theoret. Comput. Sci. 363(2) (2006)
18. Lohrey, M., Maneth, S., Mennicke, R.: Tree structure compression with repair. Arxiv preprint arXiv:1007.5406 (2010)
19. Maneth, S., Busatto, G.: Tree transducers and tree compressions. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 363–377. Springer, Heidelberg (2004)
20. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. 31(3), 762–776 (2001)

Noncommutativity Makes Determinants Hard*

Markus Bläser

Saarland University
mblaeser@cs.uni-saarland.de

Abstract. We consider the complexity of computing the determinant over arbitrary finite-dimensional algebras. We first consider the case that A is fixed. We obtain the following dichotomy: If $A/\operatorname{rad} A$ is noncommutative, then computing the determinant over A is hard. "Hard" here means #P-hard over fields of characteristic 0 and $\mathsf{Mod}_p\mathsf{P}$-hard over fields of characteristic $p > 0$. If $A/\operatorname{rad} A$ is commutative and the underlying field is perfect, then we can compute the determinant over A in polynomial time.

We also consider the case when A is part of the input. Here the hardness is closely related to the nilpotency index of the commutator ideal of A. The commutator ideal $\operatorname{com}(A)$ of A is the ideal generated by all elements of the form $xy - yx$ with $x, y \in A$. We prove that if the nilpotency index of $\operatorname{com}(A)$ is linear in n, where $n \times n$ is the format of the given matrix, then computing the determinant is hard. On the other hand, we show the following upper bound: Assume that there is an algebra $B \subseteq A$ with $B = A/\operatorname{rad}(A)$. (If the underlying field is perfect, then this is always true.) The center $Z(A)$ of A is the set of all elements that commute with all other elements. It is a commutative subalgebra. We call an ideal J a complete ideal of noncommuting elements if $B + Z(A) + J = A$. If there is such a J with nilpotency index $o(n/\log n)$, then we can compute the determinant in subexponential time. Therefore, the determinant cannot be hard in this case, assuming the counting version of the exponential time hypothesis.

Our results answer several open questions posed by Chien et al. [4].

1 Introduction

The determinant of a matrix $M = (m_{i,j}) \in k^{n\times n}$ over some field k is given by the well-known formula

$$\det M = \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma) m_{1,\sigma(1)} \cdots m_{n,\sigma(n)}.$$

The determinant plays a central role in linear algebra. It can be efficiently computed, for instance, by Gaussian elimination. In fact, there are even efficient

* Work supported by DFG grant BL 511/10-1 and by the Indo-German Max-Planck Center for Computer Science (IMPECS). A full version is available as a preprint: Markus Bläser, Noncommutativity makes determinants hard. Electronic Colloquium on Computational Complexity (ECCC) 19: 142 (2012).

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 172–183, 2013.

algorithms when the matrix M has entries from some commutative algebra, see [12] and the references given therein.

A related polynomial is the permanent of M, given by

$$\operatorname{per} M = \sum_{\sigma \in S_n} m_{1,\sigma(1)} \cdots m_{n,\sigma(n)}.$$

If M is $\{0,1\}$-valued, then $\operatorname{per} M$ is the number of perfect matchings of the bipartite graph defined by M. While the determinant is easy over commutative algebras, the permanent is hard already over the rationals. Valiant [15] showed that evaluating the $\{0,1\}$-permanent over the rationals is at least as hard as counting the number of satisfying assignments of a formula in 3-CNF.

Since the determinant and the permanent have similar formulas, it is tempting to try to modify algorithms for the determinant and use them to compute the permanent. Godsil and Gutman [9] used the determinant to approximate the permanent. They designed a matrix-valued random variable. In expectation, the square of the determinant of this random variable is the permanent. However, the variance is huge. Karmarkar et al. [11] showed how to lower the variance by extending the underlying field to the complex numbers. Chien et al. [6], building upon the work by Barvinok [2], showed that if one could compute the determinant of an $n \times n$-matrix the entries of which are themselves matrices of size $cn \times cn$ for some constant c, then there is a fully polynomial time randomized approximation scheme for the permanent of $\{0,1\}$-matrices. See [13] for further results in this direction. (Of course, there is a fully polynomial randomized approximation scheme based on Markov chains, see [10]. However, if we could evaluate noncommutative determinants as fast as commutative ones, then we would get much faster approximation schemes.)

Therefore, it is important to understand the complexity of the determinant over arbitrary finite-dimensional algebras, especially over noncommutative ones, and not only over fields or commutative algebras. The first to study this problem was Nisan [14]. He proved an exponential lower bound for the size of an algebraic branching program for computing the determinant over the free noncommutative algebra $k\langle X_{i,j}\rangle$. While the lower bound is strong, the setting is limited, because it only applies to a restricted circuit model and only to a very "powerful" algebra. Chien and Sinclair [5] extended these bounds to a wide range of "concrete" algebras by analysing their polynomial identities, for instance to matrix algebras and the Hamiltonian quaternions, albeit only in the algebraic branching program model.

Recently Arvind and Srinivasan [1] showed that the noncommutative determinant cannot have small circuits unless the permanent has small circuits. Finally, Chien et al. [4] made further progress by proving the #P-hardness and Mod_pP-hardness of the determinant for odd p for large classes of algebras.

The fundamental question behind these results is: Which properties of the algebra makes the determinant hard? In this work, we prove that this is exactly noncommutativity.

1.1 A Crash Course on the Structure of Algebras

An associative algebra A over some field k is a k-vector space together with a bilinear mapping $\cdot : A \times A \to A$, the multiplication in A. Multiplication is associative and distributes over addition. If $\lambda \in k$, then $\lambda(x \cdot y) = (\lambda x) \cdot y = x \cdot (\lambda y)$ for all $x, y \in A$. We will always assume that A is finite-dimensional (as a vector space) and contains a unit element, which we denote by 1.

A left (right, twosided) *ideal* of an algebra is a vector space that is closed under multiplication with arbitrary elements of A from the left (right, both sides). If S is a subset of A, then the left (right, twosided) ideal of A generated by S is the intersection of all left (right, twosided) ideals that contain S. Alternatively, it can be defined as the linear span generated by all elements xs (sy, xsy) with $x, y \in A$ and $s \in S$.

A left (right, twosided) ideal I is called *nilpotent*, if $I^s = \{0\}$ for some positive integer s. The nilpotency index of I is the smallest s such that $I^s = \{0\}$. If there is no such s, then the index is infinite.

The sum of all nilpotent left ideals of A is a nilpotent twosided ideal, which contains every nilpotent right ideal of A. This twosided ideal is called the *radical* of A and is denoted by $\operatorname{rad} A$. The quotient algebra $A/\operatorname{rad} A$ contains no nilpotent ideals other than the zero ideal. Since A is finite dimensional, we can alternatively define the radical of A as the intersection of all maximal twosided ideals. An ideal is maximal if it is not contained in any other ideal and is not equal to A.

We call an algebra A *semisimple*, if $\operatorname{rad} A = \{0\}$. By the above fact, $A/\operatorname{rad} A$ is semisimple. An algebra A is called *simple*, if there are no twosided ideals in A except the zero ideal and A itself. An algebra D is called a *division algebra*, if $D^\times = D \setminus \{0\}$. Here $D^\times$ is the set of all invertible elements in D. An algebra A is called *local*, if $A/\operatorname{rad} A$ is a division algebra.

The following fundamental theorem describes the structure of semisimple algebras.

Theorem 1 (Wedderburn). *Every finite dimensional semisimple k-algebra is isomorphic to a finite direct product of simple algebras. Every finite dimensional simple k-algebra A is isomorphic to an algebra $D^{n \times n}$ for an integer $n \geq 1$ and a k-division algebra D. The integer n and the algebra D are uniquely determined by A (the latter one up to isomorphism).*

For an introduction to associative algebras, we recommend [8].

1.2 Our Results

First we will consider the problem when the underlying algebra A is fixed: We are given a matrix $M \in A^{n \times n}$ as an input and our task is to compute $\det M$. We prove that the determinant over A is hard if $A/\operatorname{rad} A$ is noncommutative. If $A/\operatorname{rad} A$ is commutative, then the problem is polynomial time computable. That means, we get a complete dichotomy (Theorem 3). More precisely, we show that

- computing the determinant over A is #P-hard if $A/\operatorname{rad} A$ is noncommutative and the characteristic of k is 0.
- computing the determinant over A is $\mathsf{Mod}_p\mathsf{P}$-hard if $A/\operatorname{rad} A$ is noncommutative and the characteristic p of k is positive.

Chien et al. show that if $A/\operatorname{rad} A$ is commutative and the field k is perfect, then the determinant can be computed in polynomial time. A field is perfect if every irreducible polynomial over k has distinct roots. Any "reasonable" field is perfect, for instance, fields of characteristic zero are perfect, finite fields are perfect as well as algebraically closed fields.[1]

Our dichotomy extends the results of Chien et al. in two ways: First it works for arbitrary algebras A such that $A/\operatorname{rad} A$ is noncommutative. Chien et al. proved this only for algebras whose semisimple part $A/\operatorname{rad} A$ contained at least one matrix algebra. For instance, it did not apply to local algebras and in particular, division algebras like Hamiltonian quaternions. Second, we get $\mathsf{Mod}_2\mathsf{P}$-hardness, that is, $\oplus\mathsf{P}$-hardness, over fields of characteristic 2. The proof by Chien et al. did not work in this case.

Then we turn to the case when the algebra is given as a part of the input. Beside the matrix M, we also get a basis and the multiplication table of the algebra A from which the entries of M are taken. It seems to be natural that the dimension of A should be polynomial in the size of M. The setting above subsumes the case where we have a familiy of algebras A_n and our task is to compute the $n\times n$-determinant over A_n, for instance, computing the determinant of $n \times n$-matrices with upper triangular $n \times n$-matrices as entries. This setting is of interest because there could be a sequence of algebras each of which is noncommutative but still the determinant is easy to compute. This of course is only possible if $A_n/\operatorname{rad} A_n$ is commutative, by our first result.

We give evidence that the quantity that determines the hardness is the *nilpotency index* of the *commutator ideal* of A. The commutator ideal $\operatorname{com}(A)$ of an algebra A is the ideal generated by all elements of the form $xy-yx$ with $x, y \in A$. If the commutator ideal $\operatorname{com}(A) = \{0\}$, then A is commutative. If its nilpotency index is finite, then $A/\operatorname{rad} A$ is commutative. We prove that if the nilpotency index of the commutator ideal of A is at least linear in n, then computing the determinant of $n \times n$-matrices is as hard as counting the number of solutions of a formula in 3-CNF modulo the characteristic of k.

We prove an upper bound that is a little weaker in two ways: First we need that the nilpotency index of a somewhat larger ideal is bounded and second the upper bound does not fully match the lower bound from the hardness result. Assume that there is an algebra $B \subseteq A$ with $B \cong A/\operatorname{rad}(A)$. (If the underlying field is perfect, then this is always true.) The *center* $Z(A)$ of A is the set of all elements that commute with all other elements. It is a commutative subalgebra. We call

[1] What is actually needed by Chien et al. is that there is a subalgebra B of A such that $A = B \oplus \operatorname{rad} A$ (as vector spaces). This is true if the algebra A is separable. Over perfect fields, every algebra is separable. Any of these implications is often called the Wedderburn-Malcev Theorem. The existence of the algebra B is only needed for the upper bound and not for the hardness result.

an ideal J a *complete ideal of noncommuting elements* if $B + Z(A) + J = A$. If there is such a J with nilpotency index r, then we can compute the determinants of $n \times n$-matrices over A in time $n^{O(r)}$. Due to space limitations, the proof of the general case is omitted here, but can be found in the full version (see the footnote on the first page).

Over fields of characteristic 0 this result is almost tight assuming the counting version of the exponential time hypothesis #ETH as formulated by Dell et al. [7]. If $r = o(n/\log n)$, then computing the determinant over A cannot be #P-hard under #ETH.

The ideal J is a superset of $\mathrm{com}(A)$. It is currently not clear whether the condition that J has nilpotency index $o(n/\log n)$ can be replaced by $\mathrm{com}(A)$ has nilpotency index $o(n/\log n)$ in the upper bound. The main reason is that not too much is known about the structure of the radical. See the conclusions for some examples. In order to replace the $o(n/\log n)$ by a tight $o(n)$, we need a faster algorithm, for instance with running time $2^{O(r)}$ or $O(\binom{n}{r})$. The latter does not seem to be completely out of reach.

2 Determinants, Permanents, and Cycle Covers

Given an $n \times n$-matrix $M = (m_{i,j})$ the entries of which belong to an algebra A, the *(Cayley) determinant* of M is defined by

$$\det M = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) m_{1,\sigma(1)} \cdots m_{n,\sigma(n)}. \tag{1}$$

(Since A might be noncommutative, the order of multiplication makes a difference. When the order is by rows, then we get the Cayley determinant.) Similarly, the permanent of M is defined by

$$\mathrm{per}\, M = \sum_{\sigma \in S_n} m_{1,\sigma(1)} \cdots m_{n,\sigma(n)}. \tag{2}$$

We can interpret the matrix M as an edge-weighted digraph on the vertex set $V = \{1, \ldots, n\}$. There is an edge from i to j if $m_{i,j} \neq 0$ and the weight of this edge is $m_{i,j}$. We denote this graph by $G(M)$. A cycle cover C of a digraph is a subset of the edges such that every node has indegree and outdegree one in C. C encodes a unique permutation, which maps every node i to the node j where (i, j) is the unique edge in C leaving i. We set $C(i) := j$. In this way, we can interpret C as a permutation. It is easy to see that $\mathrm{sgn}(C) = (-1)^{n+c}$ where c is the number of cycles in C. The weight of a cycle cover is the product of the weights of the edges in C, that is, $m_{1,C(1)} \cdots m_{n,C(n)}$. Again the order is important, since the weights might not commute. For a digraph G, let $\mathrm{CC}(G)$ be the set of its cycle covers. Now we can rewrite (1) and (2) as

$$\det M = \sum_{C \in \mathrm{CC}(G(M))} \mathrm{sgn}(C) m_{1,C(1)} \cdots m_{n,C(n)} \tag{3}$$

and

$$\operatorname{per} M = \sum_{C \in \mathrm{CC}(G(M))} m_{1,C(1)} \cdots m_{n,C(n)}. \tag{4}$$

If G is an edge-weighted digraph, we will often write $\det G$ and $\operatorname{per} G$ for the determinant and permanent of its weighted adjacency matrix.

3 Hardness Proofs for the Permanent

#3-SAT is the following problem: Given a Boolean formula ϕ in 3-CNF with n variables and m clauses, count the number of satisfying assignments. #3-SAT is #P-complete. It even stays #P-complete if we assume that every variable appears as often unnegated as negated. We can achieve this by adding trivial clauses of the form $\bar{x} \vee x \vee x$ or $\bar{x} \vee \bar{x} \vee x$ for every variable x, if neccessary. This reduction increases the size of ϕ only by a constant factor. Note that thereafter, every assignment sets as many literals to true as to false.

We first briefly review the reduction by Dell et al. [7] of #3-SAT to the permanent, which is similar to the original construction by Valiant [15], but simpler and nicer. (It should go into any modern textbook.) The reduction by Dell et al. is itself derived from the reduction in [3]. Chien et al. [4] used the same approach; however, our gadgets can handle arbitrary noncommutative algebras and not only matrix algebras.

A given formula ϕ is mapped to a graph G_ϕ. This graph will have $O(m)$ edges. For every variable x, there is a selector gadget, see Figure 1 (left-hand side). There are two ways to cover this gadget by a cycle cover, taking the left-hand edge will correspond to setting x to zero and taking the right-hand edge will correspond to setting x to one.

For every clause, there is a clause gadget as depicted in Figure 1 (right-hand side). Each of the three outer edges corresponds to one literal of the clause. Taking one of the three outer edges corresponds to setting the literal to zero. It is easy to check that for every subset of the outer edges, except for the one consisting of all three outer edges, there is exactly one cycle cover. Call the graph constructed so far G'_ϕ. A cycle cover of G'_ϕ is called *consistent*, if the chosen edges in the selector gadgets and the clause gadgets are consistent, that is, whenever we chose the left-hand edge in the selector gadget for x (i.e, $x = 0$), then we choose all corresponding edges in the clause gadgets in which x appears positively and vice versa.

Fact 2. *Satisfying assignments of ϕ and consistent cycle covers of G'_ϕ stand in one-to-one correspondence.*

The last step is to get rid of inconsistent cycle covers. This is done by connecting the edge of a literal ℓ in a clause gadget by the edge in the selector gadget corresponding to setting $\ell = 0$ using an equality gadget, see Figure 2. The edge of the selector gadget and the edge of the clause gadget are subdivided, let x and z be the newly introduced vertices. These two vertices are connected as depicted

Fig. 1. Left-hand side: The selector gadget. In all figures, edges without explicitely stated weights have weight 1. Right-hand side: The clause gadget. In the gadget as it is, there is a double edge between the two nodes at the bottom. The lower edge is however subdivided when we introduce the equality gadgets.

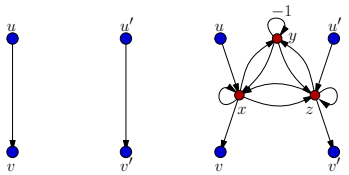


Fig. 2. The equality gadget. The pair of edges (u, v) and (u', v') of the left-hand side, one of them is an edge of the selector gadget and the other is the corresponding outer edge of a clause gadget, is connected as shown on the right-hand side.

in Figure 2. Since a literal appears in several clauses, the edge of the selector gadget is subdivided as many times.

Every consistent cycle cover of G'_ϕ can be extended to several cycle covers of G_ϕ. If the two edges connected by a equality gadget are both taken, then we take both path $u - x - v$ and $u' - z - v'$ in G_ϕ. The interior vertex y is covered by the self-loop, yielding a weight of -1. If both edges are not taken, then we take none of the corresponding paths. There are six possibilities to cover the interior nodes x, y, and z; four of them have weight 1, two of them have weight -1. This sums up to 2. (The six different covers, albeit with different weights, are shown in Figure 4.) Therefore, every consistent cycle cover is mapped to several cycle covers with a total weight of $(-1)^p 2^q$ where p is the number of literals set to zero and q is the number of literals set to one. Since we normalized ϕ, $p = q = 3m/2$.

There are also cycle covers that do not cover equality gadget consistently. This can either mean that the path $u - x - v$ is taken but not $u' - z - v'$ or that we enter the gadget via u but leave it via v'. One can prove that all cycle covers in which at least one equality gadget is not covered consistently sum up to zero. Altogether, we get that $\operatorname{per} G_\phi = (-2)^{3m/2} \cdot \#3\text{-SAT}(\phi)$, where $\#3\text{-SAT}(\phi)$ denotes the number of satisfying assignments of ϕ.

4 Hardness of the Noncommutative Determinant

We adapt the construction of the previous section to the determinant over noncommutative algebras. Note that now every cycle cover C is weighted by $\operatorname{sgn}(C)$

and the order in which the edge weights are multiplied is important. The selector gadgets stay the same. The clause gadgets stay almost the same, the only difference is that one edge gets weight -1 as is done by Chien et al. [4]. As before, for every proper subset of the outer edges, there is one cycle cover covering the clause gadget. The new -1 weight compensates the fact that some covers contain an odd number of cycles and some an even number. Let again G'_ϕ denote the resulting graph. Consistent cycle covers of G'_ϕ with sign stand in one-to-one correspondance with satisfying assignments of ϕ.

Note that since we are now working over some noncommutative algebra, the order of the vertices can be important: Up to now, we used only edge weights 1 or -1. Therefore, the order of the vertices does not matter so far.

The structure of the equality gadgets also stays the same, but we use different weights. To construct the weights, we use the following lemma.

Lemma 1. *Let A be an associative algebra. $A/\operatorname{rad} A$ is noncommutative if and only if there are invertible $i, j \in A$ such that $1 - iji^{-1}j^{-1}$ is not nilpotent.*

Proof. Assume that $A/\operatorname{rad} A$ is noncommutative and let $A/\operatorname{rad} A = A_1 \times \cdots \times A_t$ be its decomposition into simple algebras as given by Wedderburn's Theorem. One of these factors, say A_1, is either a matrix algebra of the form $B^{s\times s}$ with B being a division algebra and $s \geq 2$ or a noncommutative division algebra D. In the first case $A_1 = B^{s\times s}$, set

$$i' = \begin{pmatrix} 1\,1\,0\ldots\,0 \\ 0\,1\,0\ldots\,0 \\ 0\,0\,1\ldots\,0 \\ \vdots\,\vdots\,\vdots\,\ddots\,0 \\ 0\,0\,0\ldots\,1 \end{pmatrix} \quad \text{and} \quad j' = \begin{pmatrix} 0\,1\,0\ldots\,0 \\ 1\,0\,0\ldots\,0 \\ 0\,0\,1\ldots\,0 \\ \vdots\,\vdots\,\vdots\,\ddots\,0 \\ 0\,0\,0\ldots\,1 \end{pmatrix}$$

It is easy to check that

$$i'j' - j'i' = \begin{pmatrix} 1 & 0 & 0\ldots\,0 \\ 0 & -1 & 0\ldots\,0 \\ 0 & 0 & 0\ldots\,0 \\ \vdots & \vdots & \vdots\,\ddots\,0 \\ 0 & 0 & 0\ldots\,0 \end{pmatrix}.$$

$(i'j' - j'i')^2$ is idempotent in A_1 and therefore $i'j' - j'i'$ cannot be nilpotent. In the second case $A_1 = D$, we choose i' and j' to be noncommuting elements in D. $i'j' - j'i'$ is nonzero and therefore invertible in A_1, as D is a division algebra. The elements $i = (i', 1, \ldots, 1)$ and $j = (j', 1, \ldots, 1)$ are invertible in $A/\operatorname{rad} A$ and can be lifted to invertible elements of A. $ij - ji = (i'j' - j'i', 0, \ldots, 0)$ is not nilpotent. We have

$$1 - iji^{-1}j^{-1} = -(ij - ji) \cdot i^{-1}j^{-1},$$

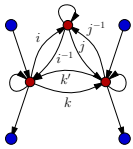


Fig. 3. The modified equality gadget. i and j are the elements constructed in the proof of Lemma 1, $k = ij$, and $k' = i^{-1}j^{-1}$. The edges between x and y have weight i and i^{-1}, between y and z weights j and j^{-1}, and between z and x weights k' and k.

which is not nilpotent, either.[2]

For the converse direction, note that $1 - iji^{-1}j^{-1} \notin \operatorname{rad} A$, since $1 - iji^{-1}j^{-1}$ is not nilpotent. Therefore the image of $1 - iji^{-1}j^{-1}$ in $A/\operatorname{rad} A$ under the canonical projection is nonzero and thus, $A/\operatorname{rad} A$ is not commutative. □

Let A be an algebra such that $A/\operatorname{rad} A$ is noncommutative. Choose i and j as constructed above. Let $k = ij$. The edges of the equality gadget get weights as depicted in Figure 3. The three new vertices x, y, and z of each gadget appear consecutively in the order x, y, z in the ordering of all the vertices. Besides this, the ordering of the new vertices can be arbitrary. Let G_ϕ denote the resulting graph.

Now we analyse what happens with a consistent cycle cover C of G'_ϕ when moving over to G_ϕ, see Figure 4. Note that consistent cycle covers of G'_ϕ have the same sign. If both paths in the equality gadget are taken, then we cover y by the self-loop. This adds one cycle to the cycle cover, which toggles the sign. If both paths are not taken, then there are six cycles covers. Two of them, have one cycle and signed weights[3] $-ijk' = -iji^{-1}j^{-1}$ and $-ki^{-1}j^{-1} = -iji^{-1}j^{-1}$. Three of them have two cycles and signed weights $ii^{-1} = 1$, $jj^{-1} = 1$, and $kk' = iji^{-1}j^{-1}$. Finally, there is one cycle cover with three cycles and signed weight -1. The total signed weight contribution is $1 - iji^{-1}j^{-1}$. Doing this for all equality gadgets, we get that every consistent cycle cover of G'_ϕ can be extended to consistent cycle covers of G_ϕ with total signed weight

$$(-1)^{3m/2}(1 - iji^{-1}j^{-1})^{3m/2}.$$

Recall that we normalized ϕ such that every assignment sets $3m/2$ literals to true and $3m/2$ literals to false. Since $1 - iji^{-1}j^{-1}$ is not nilpotent, this weight is nonzero.

[2] To not fall into the same trap as a STOC'12 referee, please note that this is not true in general. Here this holds because of the choice of i' and j'. Either A_1 is a noncommutative division algebra or A_1 is a matrix algebra. In the first case, being nonzero already means invertible. In the second case, note that $-(ij - ji) \cdot i^{-1}j^{-1}$ is a matrix with an invertible 2×2-matrix in the upper left corner and zeros elsewhere.

[3] The term signed weight also includes the change of sign induced by the parity change of the cycles.

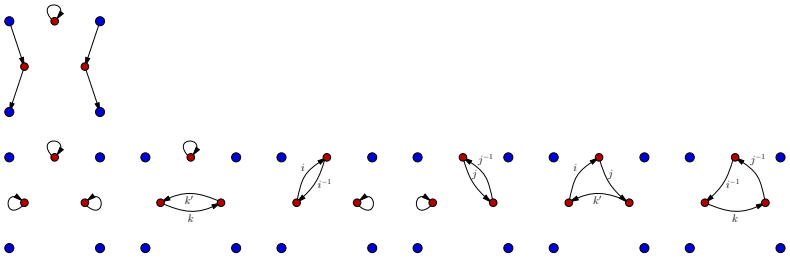


Fig. 4. First row: The one possible configuration if both edges are taken. Second row: The six possible configurations if none of the edges is taken.

It remains to analyse what happens with cycle covers of G_ϕ which are not consistent, that is, in which at least one equality gadget is not covered consistently. We will define an involution I without fixed points on the set of all inconsistent cycle covers of G_ϕ such that the weight of C and $I(C)$ cancel each other. From this it follows that the total contribution of the inconsistent cycle covers is zero. To define I, take an inconsistent cycle cover. We order the equality gadgets arbitrarily. Let C be an inconsistent cycle cover and consider the first inconsistent equality gadget. Then either C uses the path $u - x - v$ in this gadget but not $u' - z - v'$ or it enters the gadget via u and leaves it via v'. (The cases where $u' - z - v'$ is used but not $u - x - v$ or the gadget is entered via u' and left via v are symmetric.) Figure 5 shows how I pairs inconsistent cycle covers.

In the first case, C and $I(C)$ only differ in how y and z are covered. On the lefthand side, we use two cycles of weight 1, on the righthand side we use one cycle of weight $jj^{-1} = 1$. So the weights of the cycle covers are the same, but the signs differ, since the cycle cover on the lefthand side has one cycle more. (In the symmetric case, we get two cycles of weight 1 versus one cycle of weight $ii^{-1} = 1$.)

In the second case, we either use one edge of weight k and cover y by a cycle of weight 1 (lefthand side), or we use two edges of weight i and j. Since $k = ij$, the weight of both covers is the same, but again the signs differ, since the second cover has one cycle more. (In the symmetric case, we have one edge with weight $k' = i^{-1}j^{-1}$ and one additional cycle or two edges with weight $i^{-1}j^{-1}$.)

This finishs the proof that the contribution of the inconsistent cycle covers is 0.

Altogether, we get that

$$\det(G) = (-1)^{3m/2}(1 - iji^{-1}j^{-1})^{3m/2}\#\text{3-SAT}(\phi). \tag{5}$$

Note that $(-1)^{3m/2}(1-iji^{-1}j^{-1})^{3m/2}$ is a fixed nonzero element of A multiplied by the scalar $\#$3-SAT(ϕ).

Theorem 3. *Let k be a field of characteristic p. Let A be an associative algebra over k.*

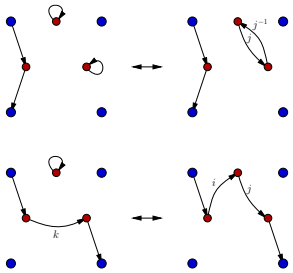


Fig. 5. The involution I. I maps the configuration on the left-hand side to the corresponding configuration on the right-hand side and vice versa.

1. *If* $A/\operatorname{rad} A$ *is noncommutative, then evaluating the determinant over* A *is* #P*-hard if* $p = 0$ *and* $\mathsf{Mod}_p\mathsf{P}$*-hard otherwise.*
2. *If* $A/\operatorname{rad} A$ *is commutative and* k *is perfect, then the determinant over* A *can be evaluated in polynomial time.*

Proof. The first part immediately follows from (5), since $(-1)^{3m/2}(1 - iji^{-1}j^{-1})^{3m/2}$ is a nonzero element of A by the choice of i and j. Note that if $p > 0$, then we get #3-SAT(ϕ), which is a scalar from k, only modulo p.

The second part follows from the fact that there is an algorithm with running time $n^{O(d)}$ for this problem, where d is the dimension of A [4]. Note that A is fixed, so d is a constant. □

5 Algebras as Part of the Input

If the algebra is part of the input, we have the following results. The proof can be found in the full version.

Theorem 4. *Let* k *be a field of characteristic* p *and* A *be an associative algebra over* k*.*

1. *If the nilpotency index of the commutator ideal of* A *is* $\Omega(n)$*, then evaluating the determinant over* A *is* #P*-hard if* $p = 0$ *and* $\mathsf{Mod}_p\mathsf{P}$*-hard otherwise, where* A *is part of the input.*
2. *If there is a complete ideal of noncommutaing elements* J *with nilpotency index* $o(n/\log n)$*, then the determinant over* A *can be computed in subexponential time over perfect fields.*

6 Conclusions

It is an interesting question whether the smallest ideal J can be much larger than $\operatorname{com}(A)$ and how much their nilpotency indices can differ. There seems to be no general answer, mainly because there is no analogue of Wedderburn's

theorem for the radical. For the algebra of upper triangular matrices, we have $J = \mathrm{com}(A) = \mathrm{rad}(A)$. For the free noncommutative algebra $k\langle x, y, z\rangle$ modulo the ideal of all monomials of degree d and the relations that make x commute with y and z, we have $\mathrm{rad}(A) \supsetneq J \supsetneq \mathrm{com}(A)$ for any J. More precisely, $\mathrm{rad}\, A$ is generated by x, y, and z, J is generated by y and z, and $\mathrm{com}(A)$ is generated by $yz - zy$. In our upper bound, we can take the minimum over all complete ideals J of noncommuting elements. Is there an easy characterisation of the best J?

Acknowledgement. I would like to thank Prahladh Harsha for drawing my attention to this problem.

References

1. Arvind, V., Srinivasan, S.: On the hardness of the noncommutative determinant. In: Proc. 42nd ACM Symp. on Theory of Comput (STOC), pp. 677–686 (2010)
2. Barvinok, A.: Polynomial time algorithms to approximate permanents and mixed discriminants within a simply exponential factor. Random Struct. Algorithms 14(1), 29–61 (1999)
3. Bläser, M., Dell, H.: Complexity of the cover polynomial. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 801–812. Springer, Heidelberg (2007)
4. Chien, S., Harsha, P., Sinclair, A., Srinivasan, S.: Almost settling the hardness of noncommutative determinant. In: Proc. 43rd ACM Symp. on Theory of Comput (STOC), pp. 499–508 (2011)
5. Chien, S., Sinclair, A.: Algebras with polynomial identities and computing the determinant. J. Comput. Sys. Sci. 67(2), 263–290 (2003)
6. Chien, S., Rasmussen, L., Sinclair, A.: Clifford algebras and approximating the permanent. J. Comput. Sys. Sci. 67(2), 263–290 (2003)
7. Dell, H., Husfeldt, T., Wahlén, M.: Exponential Time Complexity of the Permanent and the Tutte Polynomial. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 426–437. Springer, Heidelberg (2010)
8. Drozd, Y.A., Kirichenko, V.V.: Finite dimensional algebras. Springer (1994)
9. Godsil, C.D., Gutman, I.: On the matching polynomial of a graph. In: Lovász, L., Sós, V.T. (eds.) Algebraic Methods in Graph Theory, vol. 1, pp. 241–249. North-Holland (1981)
10. Jerrum, M., Sinclair, A., Vigoda, E.: A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. J. ACM 51(4), 671–697 (2004)
11. Kamarkar, N., Karp, R.M., Lipton, R.J., Lovász, L., Luby, M.: A Monte-Carlo algorithm for estimating the permanent. SIAM J. Comput. 22(2), 284–293 (1993)
12. Mahajan, M., Vinay, V.: Determinant: Old algorithms and new insights. SIAM J. Discrete Math. 12(4), 474–490 (1999)
13. Moore, C., Russell, A.: Approximating the permanent via nonabelian determinants, arXiv:0906.1702 (2009)
14. Nisan, N.: Lower bounds for noncommutative computation. In: Proc. 23rd ACM Symp. on Theory of Comput. (STOC), pp. 410–418 (1991)
15. Valiant, L.G.: The complexity of computing the permanent. Theoret. Comput. Sci. 8, 189–201 (1979)

Optimal Orthogonal Graph Drawing with Convex Bend Costs*

Thomas Bläsius, Ignaz Rutter, and Dorothea Wagner

Faculty of Informatics, Karlsruhe Institute of Technology (KIT), Germany
firstname.lastname@kit.edu

Abstract. Traditionally, the quality of orthogonal planar drawings is quantified by the total number of bends, or the maximum number of bends per edge. However, this neglects that in typical applications, edges have varying importance. We consider the problem OptimalFlexDraw that is defined as follows. Given a planar graph G on n vertices with maximum degree 4 (*4-planar graph*) and for each edge e a cost function $\text{cost}_e\colon \mathbb{N}_0 \longrightarrow \mathbb{R}$ defining costs depending on the number of bends e has, compute an orthogonal drawing of G of minimum cost.

In this generality OptimalFlexDraw is NP-hard. We show that it can be solved efficiently if 1) the cost function of each edge is convex and 2) the first bend on each edge does not cause any cost. Our algorithm takes time $O(n \cdot T_{\text{flow}}(n))$ and $O(n^2 \cdot T_{\text{flow}}(n))$ for biconnected and connected graphs, respectively, where $T_{\text{flow}}(n)$ denotes the time to compute a minimum-cost flow in a planar network with multiple sources and sinks. Our result is the first polynomial-time bend-optimization algorithm for general 4-planar graphs optimizing over all embeddings. Previous work considers restricted graph classes and unit costs.

1 Introduction

Orthogonal graph drawing is one of the most important techniques for the human-readable visualization of complex data. Since edges are required to be straight orthogonal lines—which automatically yields good angular resolution and short links—the human eye may easily adapt to the flow of an edge. The readability of orthogonal drawings can be further enhanced in the absence of crossings, i.e., if the underlying data exhibits planar structure. In order to be able to visualize all 4-planar graphs, we allow edges to have bends. Since bends decrease readability, we seek to minimize the number of bends.

We consider the problem OptimalFlexDraw whose input consists of a planar graph G with maximum degree 4 and for each edge e a cost function $\text{cost}_e\colon \mathbb{N}_0 \longrightarrow \mathbb{R}$ defining costs depending on the number of bends on e. We seek an orthogonal drawing of G with minimum cost. Garg and Tamassia [7] show that it is NP-hard to decide whether a 4-planar graph admits an orthogonal drawing with zero bends, directly implying that OptimalFlexDraw is NP-hard in

* Partly done within GRADR – EUROGIGA project no. 10-EuroGIGA-OP-003.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 184–195, 2013.

general. For special cases, namely planar graphs with maximum degree 3 and series-parallel graphs, Di Battista et al. [4] give an algorithm minimizing the total number of bends, optimizing over all planar embeddings. They introduce the concept of spirality that is similar to the concept of rotation we use. Bläsius et al. [2] consider the decision problem FlexDraw, where each edge has a *flexibility* specifying its allowed number of bends. They give a polynomial-time decision algorithm for the case that all flexibilities are positive. OptimalFlexDraw can be seen as the optimization version of FlexDraw since it allows to find a drawing that minimizes the number of bends exceeding the flexibilities.

As minimizing the total number of bends is NP-hard, many results initially fix the planar embedding. Tamassia [9] describes a flow network for minimizing the number of bends for a fixed planar embedding. The technique can be easily adapted to solve OptimalFlexDraw if the planar embedding is fixed. Biedl and Kant [1] show that every planar graph admits a drawing with at most two bends per edge except for the octahedron. Even though fixing an embedding allows to efficiently minimize the total number of bends, it neglects that this choice may have a huge impact on the number of bends in the resulting drawing.

Contribution and Outline. Our main result is the first polynomial-time bend-optimization algorithm for general 4-planar graphs optimizing over all embeddings. Previous work considers restricted graph classes and unit costs. We solve OptimalFlexDraw if 1) all cost functions are convex and 2) the first bend is for free. Note that convexity is quite natural, and without condition 2) OptimalFlexDraw is NP-hard. An interesting special case is the minimization of the total number of bends over all planar embeddings, where one bend is for free. Moreover, as every 4-planar graph has a drawing with at most two bends per edge [1], we can minimize the number of 2-bend edges in such a drawing.

To solve OptimalFlexDraw for biconnected graphs, we extend the notion "number of bends" to split components and use dynamic programming to compute their cost functions bottom-up in the SPQR-tree. In each step we use a flow network similar to the one described by Tamassia [9]. The major problem is that the cost functions for split components may be non-convex [3]. To overcome this problem, we show the existence of an optimal solution with at most three bends per edge except for a single edge per block with up to four bends. Due to an extension to split components, it suffices to consider their cost functions on the interval $[0, 3]$, and we show that, on this interval, they are convex.

We show in Section 3 that for biconnected graphs, the number of bends per edge can always be reduced to three and generalize this result to split components in Section 4. In Section 5 we show that the cost functions for split components are convex on the interval $[0, 3]$. This yields an algorithm for computing optimal drawings of biconnected graphs, which extends to connected graphs. Omitted proofs are in the appendix and in the full version of this paper [3].

2 Preliminaries

An instance of OptimalFlexDraw is a 4-planar graph G together with a cost function $\text{cost}_e : \mathbb{N}_0 \longrightarrow \mathbb{R} \cup \{\infty\}$ for each edge e assigning a cost to e depending

on the number of its bends. OptimalFlexDraw asks for an optimal orthogonal drawing, i.e., a drawing with minimum cost summed over all edges.

For a cost function $\text{cost}_e(\cdot)$ let $\Delta\,\text{cost}_e(\rho) = \text{cost}_e(\rho+1) - \text{cost}_e(\rho)$ be its *difference function*. A cost function is *monotone* if its difference function is greater or equal to 0. It is *convex*, if its difference function is monotone. The *base cost* of the edge e with monotone cost function is $b_e = \text{cost}_e(0)$. According to the decision problem FlexDraw, G is said to have *positive flexibility* if $\text{cost}_e(0) = \text{cost}_e(1)$ holds for every edge e. An instance G of OptimalFlexDraw is *positive-convex* if it has positive flexibility and each cost function is convex.

2.1 Connectivity and the SPQR-Tree

A graph is *connected* if there exists a path between any pair of vertices. A *separating k-set* is a set of k vertices whose removal disconnects the graph. Separating 1-sets and 2-sets are *cutvertices* and *separation pairs*, respectively. A connected graph is *biconnected* (*triconnected*) if it does not have a cutvertex (separation pair). The *cut components* with respect to a separating k-set S are the maximal subgraphs that are not disconnected by removing S.

The *SPQR-tree* $\mathcal{T}$ introduced by Di Battista and Tamassia [5,6] is a succinct representation of all planar embeddings of a biconnected planar graph G. It describes a decomposition of G along its *split pairs*, which are separation pairs or single edges, into triconnected components. It can be computed in linear time [8] and has linear size. Every node μ of $\mathcal{T}$ is associated with a multigraph $\text{skel}(\mu)$, called *skeleton*, on a subset of the vertices of G. Each inner node in $\mathcal{T}$ is an S-, P- or R-node, having a cycle, a bunch of parallel edges and a triconnected graph as skeleton, respectively. The edges in these skeletons are called *virtual edges*. The leaves of $\mathcal{T}$ are Q-nodes, their skeletons consist of an edge of G plus a parallel virtual edge. When two nodes μ_1 and μ_2 are adjacent in $\mathcal{T}$ this edge identifies a virtual edge in $\text{skel}(\mu_1)$ with a virtual edge in $\text{skel}(\mu_2)$, and each virtual edge in each node is associated with exactly one such neighbor.

Rooting the SPQR-tree in some node τ determines for each node $\mu \neq \tau$ a unique *parent edge* in $\text{skel}(\mu)$ that is associated with μ's parent. The *pertinent graph* $\text{pert}(\mu)$ of a node μ is recursively defined as follows. For a Q-node $\text{pert}(\mu)$ is the edge in G it corresponds to. For an inner node $\text{pert}(\mu)$ is the graph obtained from $\text{skel}(\mu)$ by deleting the parent edge and replacing each virtual edge by the pertinent graph of the corresponding child. The *expansion graph* of a virtual edge ε is the pertinent graph of the child of μ corresponding to ε when $\mathcal{T}$ is rooted at μ. The SPQR-tree represents all embeddings of G on a sphere, i.e., embeddings without a specific outer face, by allowing independent choices for the embeddings of all skeletons. For R-nodes the embedding is fixed up to a flip, for P-nodes one can choose an arbitrary order for the parallel edges, and for S- and Q-nodes there is no embedding choice.

Usually the SPQR-tree is assumed to be unrooted, as described above, or rooted at a Q-node, representing embeddings with the corresponding edge on the outer face. We consider the SPQR-tree to be rooted at an arbitrary node τ. This also restricts the choice of the outer face and the embedding choices are of the following kind. For every node $\mu \neq \tau$ one can choose an embedding for

skel(μ) with the parent edge on the outer face. For τ itself, the choice for the embedding of skel(τ) includes the choice of an outer face.

2.2 Orthogonal Representation

Two orthogonal drawings of a 4-planar graph G are *equivalent*, if they have the same planar embedding, and the same shape, i.e., the sequence of right and left turns is the same when traversing the faces of G. To make this precise, we define *orthogonal representations*, originally introduced by Tamassia [9], as equivalence classes of this relation. To ease the notation we only consider biconnected graphs.

Let Γ be an orthogonal drawing of a biconnected 4-planar graph G and let $\mathcal{E}$ be the planar embedding induced by it. We define the *rotation* of an edge e in an incident face f to be the number of bends to the right minus the number of bends to the left when traversing f in clockwise order (counter-clockwise if f is the outer face) and denote the resulting value by $\text{rot}(e_f)$. Similarly, we define the rotation of a vertex v in an incident face f, denoted by $\text{rot}(v_f)$, to be 1, -1 and 0 if there is a turn to the right, a turn to the left and no turn, respectively. The orthogonal representation $\mathcal{R}$ belonging to Γ consists of the planar embedding $\mathcal{E}$ of G and all rotation values of edges and vertices, respectively. It is easy to see that every orthogonal representation has the following properties.

(I) For every edge e with incident faces f_1, f_2 we have $\text{rot}(e_{f_1}) = -\text{rot}(e_{f_2})$.
(II) The sum over all rotations in a face is 4 (-4 for the outer face).
(III) The sum of rotations around a vertex v is $2 \cdot (\deg(v) - 2)$.

Tamassia showed that the converse is also true [9], i.e., if $\mathcal{R}$ satisfies the above properties, then it is the orthogonal representation of a class of drawings. In what follows we always neglect the exact geometry and work with orthogonal representations instead of drawings. In some cases we write $\text{rot}_{\mathcal{R}}(\cdot)$ instead of $\text{rot}(\cdot)$ to make clear which orthogonal representation we refer to. Moreover, the face in the subscript is omitted if it is clear from the context.

Let G be a 4-planar graph with orthogonal representation $\mathcal{R}$ and two vertices s and t incident to a common face f. We define $\pi_f(s,t)$ to be the path from s to t on the boundary of f, when traversing f in clockwise direction (counter-clockwise if f is the outer face). Let $s = v_1, \ldots, v_k = t$ be the vertices on the path $\pi_f(s,t)$. The rotation of $\pi(s,t)$ is defined as

$$\text{rot}(\pi(s,t)) = \sum_{i=1}^{k-1} \text{rot}(\{v_i, v_{i+1}\}) + \sum_{i=2}^{k-1} \text{rot}(v_i)\,.$$

Let G be a biconnected positive-convex instance of OptimalFlexDraw with optimal orthogonal representation $\mathcal{R}$ and let H be a split component with respect to $\{s,t\}$ such that the orthogonal representation $\mathcal{S}$ induced by H has s and t on its outer face. Then $\mathcal{S}$ is *tight* with respect to s and t if the rotations of s and t in internal faces are 1, i.e., they have 90°-angles in internal faces. The orthogonal representation of G is *tight* if every split component having its corresponding split pair on its outer face is tight. We can assume without loss of generality that all orthogonal representations are tight [2, Lemma 2].

2.3 Flow Network

A *flow network* is a tuple $N = (V, A, \text{COST}, \text{dem})$ where (V, A) is a directed (multi-)graph, COST is a set containing a *convex cost function* $\text{cost}_a \colon \mathbb{N}_0 \longrightarrow \mathbb{R} \cup \{\infty\}$ for each arc $a \in A$ and $\text{dem} \colon V \longrightarrow \mathbb{Z}$ is the *demand* of the vertices. A *flow* is a function $\phi \colon A \longrightarrow \mathbb{N}_0$ assigning a certain amount of flow to each arc. It is *feasible*, if the difference of incoming and outgoing flow at each vertex equals its demand. The *cost* of ϕ is $\text{cost}(\phi) = \sum_{a \in A} \text{cost}_a(\phi(a))$. An arc a has *capacity* c if $\text{cost}_a(\rho) = 0$ for $\rho \in [0, c]$ and $\text{cost}_a(\rho) = \infty$ otherwise.

The *parameterized flow network* with respect to two nodes $u, v \in V$ is defined the same as N but with a *parameterized demand* of $\text{dem}(u) - \rho$ for u and $\text{dem}(v) + \rho$ for v where ρ is a parameter. The *cost function* $\text{cost}_N(\rho)$ of N is defined to be $\text{cost}(\phi)$ of an optimal flow ϕ in N with respect to the demands determined by ρ. Increasing ρ by 1 can be seen as pushing one unit of flow from u to v.

Theorem 1. *The cost function of a parameterized flow network is convex on the interval* $[\rho_0, \infty]$*, where* $\rho_0 = \text{argmin}_{\rho \in \mathbb{Z}}\{\text{cost}_N(\rho)\}$.

3 Valid Drawings with Fixed Planar Embedding

In this section we consider the problem FlexDraw for biconnected planar graphs with fixed embedding. Given an arbitrary *valid orthogonal representation*, i.e., an orthogonal representation that respects the flexibilities, we show the existence of a valid orthogonal representation with the same angles around vertices, the same planar embedding, and at most three bends per edge except for possibly a single edge on the outer face with up to five bends.

Let G be a 4-planar graph with positive flexibility and valid orthogonal representation $\mathcal{R}$. If the number of bends of an edge e equals its flexibility, we orient e such that its bends are right bends (we always assume that edges are bent in only one direction). Otherwise, e remains undirected. A path $\pi = (v_1, \dots, v_k)$ in G is *directed*, if the edge $\{v_i, v_{i+1}\}$ (for $i \in \{1, \dots, k-1\}$) is either undirected or directed from v_i to v_{i+1}. It is *strictly directed*, if it is directed and does not contain undirected edges. These definitions extend to *(strictly) directed cycles*. The terms $\text{left}(C)$ and $\text{right}(C)$ denote the set of edges and vertices lying to the left and right of a (strictly) directed cycle C. A cut $(U, V \setminus U)$ is *directed* from U to $V \setminus U$, if every edge crossing the cut is undirected or directed from U to $V \setminus U$. It is *strictly directed* if it additionally does not contain undirected edges.

Lemma 1. *Let G be a graph with positive flexibility and vertices s and t such that $G + st$ is biconnected and 4-planar. Let further $\mathcal{R}$ be a valid orthogonal representation with s and t incident to a face f such that $\pi_f(t, s)$ is strictly directed from t to s. The following holds.*
(1) $\text{rot}_{\mathcal{R}}(\pi_f(s, t)) \leq -3$ *if f is the outer face and G is not a single path*
(2) $\text{rot}_{\mathcal{R}}(\pi_f(s, t)) \leq -1$ *if f is the outer face*
(3) $\text{rot}_{\mathcal{R}}(\pi_f(s, t)) \leq 5$

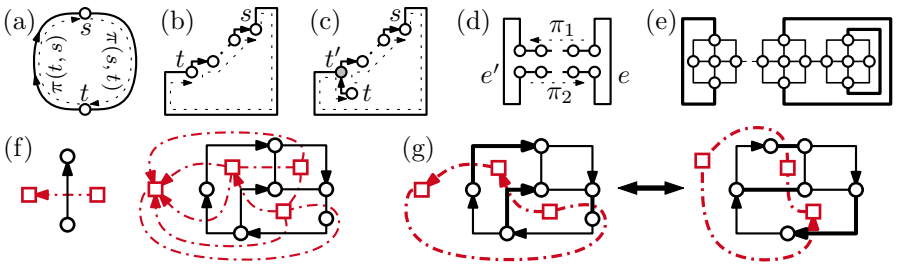


Fig. 1. (a–c) Illustration of Lemma 1. (d) Two edges on the outer face with four bends. (e) Example with $O(n)$ edges requiring four bends. (f) The flex graph of an orthogonal drawing. (g) Bending along a cycle in the flex graph. The resulting flex graph contains the same cycle directed in the opposite direction, so this operation can be reversed.

Proof (Sketch). We show case (3), where f is an internal face; see Fig. 1(a). The other cases work similarly. Since $\pi_f(t,s)$ is strictly directed, every edge on this path has at least one right bend (when traversing from t to s), yielding a rotation of at least 1. Moreover, every internal vertex in $\pi_f(t,s)$ may have a left bend, yielding a rotation of at most -1. As the number of internal vertices is one less than the number of edges in a path, $\mathrm{rot}(\pi_f(t,s)) \geq 1$ holds. We first assume that neither s nor t have degree 1; see Fig. 1(b). As the rotation around f is 4, we have $\mathrm{rot}(\pi_f(s,t)) = 4 - \mathrm{rot}(s_f) - \mathrm{rot}(t_f) - \mathrm{rot}(\pi_f(t,s))$. Moreover, we have $\mathrm{rot}(s_f), \mathrm{rot}(t_f) \geq -1$ (since $\deg(s), \deg(t) > 1$) and $\mathrm{rot}(\pi_f(t,s)) \geq 1$ (as seen above), yielding $\mathrm{rot}(\pi_f(s,t)) \leq 5$. If t (or s) has degree 1, $\mathrm{rot}(\pi_f(t,s)) \geq 2$ holds since an internal vertex t' (or s') of $\pi_f(t,s)$ has degree 3 and thus cannot have rotation -1, canceling out the rotation of -2 at t (or s); see Fig. 1(c). □

The *flex graph* $G^{\times}_{\mathcal{R}}$ of G with respect to a valid orthogonal representation $\mathcal{R}$ is the dual graph of G such that the dual edge $e^\star$ is directed from the face right of e to the face left of e (or undirected if e is undirected); see Fig. 1(f). Assume C is a simple directed cycle in the flex graph. Then *bending* along this cycle yields a new valid orthogonal representation $\mathcal{R}'$ defined as follows; see Fig. 1(g). For an edge $e^\star = (f_1, f_2)$ in C dual to e we decrease $\mathrm{rot}(e_{f_1})$ and increase $\mathrm{rot}(e_{f_2})$ by 1. It is easy to see that $\mathcal{R}'$ is an orthogonal representation. Moreover, no edge has more bends than allowed by its flexibility, as C is directed. The following lemma states that a high rotation along a path $\pi_f(s,t)$ for two vertices s and t sharing the face f can be reduced by 1 using a directed cycle in the flex graph.

Lemma 2. *Let G be a biconnected 4-planar graph with positive flexibility, valid orthogonal representation $\mathcal{R}$, and s and t on a common face f. The flex graph $G^{\times}_{\mathcal{R}}$ contains a directed cycle C such that $s \in \mathrm{left}(C)$ and $t \in \mathrm{right}(C)$ if one of the following conditions holds.*

(1) $\mathrm{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq -2$, *$f$ is the outer face and $\pi_f(s,t)$ is not strictly directed from t to s*
(2) $\mathrm{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 0$ *and f is the outer face*
(3) $\mathrm{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 6$

Proof (Sketch). Assume such a cycle C does not exist. By the duality of cycles and cuts, this implies that there is no directed cut (S, T) in G with $s \in S$ and $t \in T$. Thus for every partition $V = S \dot\cup T$ with $s \in S$ and $t \in T$ there is a directed edge with its source in T and its target in S. Iteratively applying this argument yields a path π in G strictly directed from t to s. For each of the conditions (1)–(3), we obtain a contradiction by applying Lemma 1 to the subgraph of G consisting of the strictly directed path π and the path $\pi_f(s, t)$. □

As edges with many bends imply the existence of paths with high rotation, we can use Lemma 2 to successively reduce the number of bends on edges with many bends. Since we only bend along cycles in the flex graph, neither the embedding nor the angles around vertices are changed.

Theorem 2. *Let G be a biconnected 4-planar graph with positive flexibility and valid orthogonal representation. Then G has a valid orthogonal representation with the same planar embedding, the same angles around vertices and at most three bends per edge, except for one edge on the outer face with up to five bends.*

Proof (Sketch). We iteratively bend along cycles in the flex graph to reduce the number of bends on edges with more than three bends. To ensure that the number of bends of an edge does not increase above three once it is below, we set its flexibility down to its current number of bends (but at least 1).

Let $e = \{s, t\}$ be an edge with more than three bends having its negative rotation in an internal face f, i.e., $\text{rot}(e_f) \leq -4$, and assume that $\pi_f(t, s)$ consists of e. As the rotation around the face f is 4, we have $\pi_f(s, t) = 4 - \text{rot}(s_f) - \text{rot}(t_f) - \text{rot}(e_f)$, yielding $\pi_f(s, t) \geq 6$. By Lemma 2 a cycle C separating s from t and thus containing $e^\star$ exists. Bending along this cycle reduces the number of bends on e. With a similar argument, the bends of edges having their negative rotation on the outer face can be reduced to 5. Moreover, if e is an edge with $\text{rot}(e_f) \leq -4$, where f is the outer face, and the boundary of the outer face contains a path with non-negative rotation, the bends of e can be reduced by case (1) of Lemma 2. This is the case if there is another edge e' with $\text{rot}(e'_f) \leq -4$; see Fig. 1(d) where one of the paths π_1 or π_2 must have non-negative rotation. Repeatedly applying this operation yields the theorem. □

If we allow the embedding to be changed slightly, we obtain an even stronger result. Assume the edge e lying on the outer face has five bends. Rerouting e in the opposite direction around the rest of the graph yields a drawing where e has only three bends. Thus, there might be a single edge with up to four bends in the worst case. Note that this result is restricted to biconnected graphs. For general graphs it implies that each block contains at most a single edge with up to four bends. Figure 1(e) illustrates an instance of FlexDraw with linearly many blocks and linearly many edges requiring four bends. We note that increasing the lower bound for the flexibilities from 1 to 2 in the above arguments yields a result similar to the existence of 2-bend drawings by Biedl and Kant [1].

Theorem 2 implies that it is sufficient to consider the flexibility of every edge to be at most 5, or in terms of costs, it is sufficient to store the cost function of

Fig. 2. Adding the safety edges (bold) to G and the effects on the dual graph

an edge only in the interval $[0, 5]$. However, there are two reasons why we need a stronger result. First, we want to compute cost functions of split components and thus we have to limit the number of "bends" they can have (we deal with this in the next section). Second, the cost function of a split component may already be non-convex on the interval $[0, 5]$ [3]. Fortunately, there may be at most a single edge with up to five bends, all remaining edges have at most three bends and thus we only need to consider the interval $[0, 3]$.

4 Flexibility of Split Components and Nice Drawings

Let G be a biconnected 4-planar graph with SPQR-tree $\mathcal{T}$ rooted at some node τ. Recall that we do not require τ to be a Q-node. A node $\mu \neq \tau$ of $\mathcal{T}$ has a unique parent and $\text{skel}(\mu)$ contains a unique virtual edge $\varepsilon = \{s, t\}$ that is associated with this parent. We call the split-pair $\{s, t\}$ a *principal split pair* and the pertinent graph $\text{pert}(\mu)$ with respect to the root τ a *principal split component*. The vertices s and t are the *poles* of this split component. Note that an edge (whose Q-node is not τ) is also a principal split component.

Let $\mathcal{R}$ be a valid orthogonal representation of G such that the planar embedding of $\mathcal{R}$ is represented by $\mathcal{T}$ rooted at τ. Consider a principal split component H with respect to the split pair $\{s, t\}$ and let $\mathcal{S}$ be the restriction of $\mathcal{R}$ to H. Note that the poles s and t are on the outer face f of $\mathcal{S}$. We define $\max\{|\operatorname{rot}_{\mathcal{S}}(\pi_f(s, t))|, |\operatorname{rot}_{\mathcal{S}}(\pi_f(t, s))|\}$ to be the *number of bends* of the split component H. With this terminology we can assign a *flexibility* $\text{flex}(H)$ to H and we define $\mathcal{R}$ to be *valid* if and only if H has at most $\text{flex}(H)$ bends. We say that the graph G has *positive flexibility* if the flexibility of every principal split component is at least 1. Note that this terminology extends the notion of bends and flexibility for edges.

To obtain a result similar to Lemma 2 we need to extend the flex graph such that it respects flexibilities of principal split components. As we cannot deal with principal split components with respect to different roots at the same time, we initially choose an arbitrary Q-node τ to be the root of the SPQR-tree $\mathcal{T}$. We then augment G for each principal split component H with two so-called *safety edges* between the poles; see Fig. 2. As a cycle in the flex graph of the augmented graph crosses H if and only if it crosses these safety edges, suitably orienting them ensures that bending along a cycle in the flex graph does not increase the number of bends of H above $\text{flex}(H)$. As we consider only principal split components, the safety edges can be added for all of them at the same time without losing planarity. Denote the resulting augmented graph by G^+ and call the resulting flex graph the *extended flex graph*. As a directed safety edge represents a path with rotation at least 1 along the outer face of its split

component, a strictly directed path in G^+ yields a path in G with positive rotation. All remaining arguments from the proof of Lemma 2 can be applied literally, yielding the following lemma.

Lemma 3. *Let G be a biconnected 4-planar graph with positive flexibility, valid orthogonal representation $\mathcal{R}$, and s and t sharing a face f. The extended flex graph contains a directed cycle C such that $s \in \text{left}(C)$ and $t \in \text{right}(C)$, if one of the following conditions holds.*
(1) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq -2$*, f is the outer face and $\pi_f(s,t)$ is not strictly directed from t to s*
(2) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 0$ *and f is the outer face*
(3) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 6$

We define a valid orthogonal representation of G to be *nice* if 1) it is tight, 2) every principal split component has at most three bends, and 3) the edge corresponding to the root τ of the SPQR-tree, in case τ is a Q-node, has at most five bends. The following statement extends Theorem 2. Moreover, it obviously extends from FlexDraw to OptimalFlexDraw, i.e., every positive-convex 4-planar graph has an optimal drawing that is nice.

Theorem 3. *Every biconnected 4-planar graph with positive flexibility having a valid orthogonal representation has a valid orthogonal representation with the same planar embedding and the same angles around vertices that is nice with respect to at least one node chosen as root of its SPQR-tree.*

Proof (Sketch). Similar to the proof in Theorem 2 we can use Lemma 3 to reduce the number of bends of split components having their negative rotation in an internal face down to three, while preserving this property once it is achieved by reducing the flexibilities. Similarly, the number of bends of split components with negative rotation in the outer face can be reduced to five. Assume we have an orthogonal representation where each principal split component with more than three bends has its negative rotation on the outer face. If there are two disjoint components of this type, a similar argument as for single edges can be used to reduce the number of bends of one of them. For the remaining nested principal split components of this type we can show that there is no need to reduce the number of bends further, as the drawing is already nice if we reroot the SPQR-tree at the node corresponding to the innermost of these split components. □

5 Optimal Drawings with Variable Planar Embedding

All results presented so far fix the planar embedding of the input graph. In the following we optimize over all embeddings of a biconnected 4-planar graph G. As we only consider positive-convex instances of OptimalFlexDraw, it suffices to consider nice drawings (Theorem 3). Whether a drawing is nice depends on the node chosen as the root for the SPQR-tree $\mathcal{T}$. For a node τ we call an orthogonal representation *τ-optimal* if it is optimal among all representations that are nice

with respect to the root τ. We say that it is $(\tau, \mathcal{E})$-*optimal* if it is optimal among all orthogonal representations that are nice with respect to τ and induce the planar embedding $\mathcal{E}$ on skel(τ). Computing a $(\tau, \mathcal{E})$-optimal solution for every planar embedding $\mathcal{E}$ of skel(τ) obviously yields a τ-optimal orthogonal representation. Moreover, the minimum of all τ-optimal solutions over all nodes of the SPQR-tree yields an overall optimal orthogonal representation. Since G has maximum degree 4, skel(τ) has $O(|\operatorname{skel}(\tau)|)$ embeddings (including the choice of an outer face), and hence the sum over all embeddings of all nodes of the SPQR-tree is in $O(n)$. Thus, an algorithm computing a $(\tau, \mathcal{E})$-optimal solution can be used to compute an overall optimal solution by applying it $O(n)$ times.

In the following we show how to compute a $(\tau, \mathcal{E})$-optimal solution efficiently, using a dynamic program computing cost functions of principal split components bottom-up in the SPQR-tree. We start by defining the cost function $\operatorname{cost}_H(\cdot)$ of a principal split component H with poles s and t. Recall that the number of bends of H with respect to an orthogonal representation $\mathcal{S}$ with s and t on the outer face f is defined to be $\max\{|\operatorname{rot}_{\mathcal{S}}(\pi_f(s,t))|, |\operatorname{rot}_{\mathcal{S}}(\pi_f(t,s))|\}$. This implies that there is a lower bound of $\ell_H = \lceil(\deg(s) + \deg(t) - 2)/2\rceil$ bends. For a number of bends $\rho \geq \ell_H$, we define $\operatorname{cost}_H(\rho)$ to be the cost of an orthogonal representation of H that is optimal among all nice representations with ρ bends. For $\rho \in [0, \ell_H)$ we formally set $\operatorname{cost}(\rho) = \operatorname{cost}(\ell_H)$. As we are only interested in nice drawings, it remains to compute $\operatorname{cost}_H(\rho)$ for $\rho \in [\ell_H, 3]$. One of the main results of this section is the following theorem.

Theorem 4. *If the poles of a principal split component do not both have degree 3, then its cost function is convex on the interval* $[0, 3]$.

We prove Theorem 4 later and first assume that it holds. The *base cost* b_H of a principal split component is the minimum of $\operatorname{cost}_H(\cdot)$. Due to Theorem 4 we have $b_H = \operatorname{cost}_H(0)$ except for the single case that $\deg(s) = \deg(t) = 3$, where $\operatorname{cost}_H(\cdot)$ may be non-convex. In this case $b_H = \operatorname{cost}_H(3)$.

In the following we assume that the cost function of every principal split component with respect to the root τ is already computed and show how this can be used to compute a $(\tau, \mathcal{E})$-optimal solution. To this end, we define a flow network on skel(τ), similar to Tamassia's flow network [9]. The cost functions computed for the children of τ will be used as cost functions on arcs in the flow network. Since only flow networks with convex costs can be solved efficiently, we have to deal with potentially non-convex cost functions in the case where both poles have degree 3. Our strategy is to simply ignore these subgraphs by contracting them into single vertices. The following lemma justifies this strategy.

Lemma 4. *Let G be a biconnected positive-convex instance of* OptimalFlexDraw *with τ-optimal orthogonal representation $\mathcal{R}$ and let H be a principal split component with non-convex cost function and base cost b_H. Let further G' be the graph obtained from G by contracting H into a single vertex and let $\mathcal{R}'$ be a τ-optimal orthogonal representation of G'. Then* $\operatorname{cost}(\mathcal{R}) = \operatorname{cost}(\mathcal{R}') + b_H$ *holds.*

Now we are ready to define the flow network $N^{\mathcal{E}}$ on $\text{skel}(\tau)$ with respect to its fixed embedding $\mathcal{E}$. For each vertex v, each virtual edge ε and each face f in $\text{skel}(\tau)$ the flow network $N^{\mathcal{E}}$ contains the nodes v, ε and f, called *vertex node*, *edge node* and *face node*, respectively. The network $N^{\mathcal{E}}$ contains the arcs (v, f) and (f, v) with capacity 1, called *vertex-face arcs*, if the vertex v and the face f are incident. For every virtual edge ε we add *edge-face arcs* (ε, f) and (f, ε) if f is incident to ε. We use $\text{cost}_H(\cdot) - b_H$ as cost function of the arc (f, ε), where H is the expansion graph of ε. The edge-face arcs (ε, f) in the opposite direction have infinite capacity with 0 cost. Every inner face has a demand of 4, the outer face has a demand of -4. An edge node ε stemming from the edge $\varepsilon = \{s, t\}$ with expansion graph H has demand $\deg_H(s) + \deg_H(t) - 2$, where $\deg_H(v)$ denotes the degree of v in H. The demand of a vertex node v is $4 - \deg_G(v) - \deg_{\text{skel}(\tau)}(v)$.

In the flow network $N^{\mathcal{E}}$, the flow entering a face node f via a vertex-face arc or an edge-face arc is interpreted as the rotation at this vertex or along the path between the poles of its expansion graph, respectively, where incoming flow is positive rotation. Thus, a feasible flow describes the shapes of all expansion graphs and the composition of their representations at vertices. Note that this composition is possible as we can assume them to be tight. Let $b_{H_1}, \dots, b_{H_k}$ be the base costs of the children of τ. We define the *total base costs* of τ to be $b_\tau = \sum_i b_{H_i}$. It can be shown that an optimal flow ϕ in $N^{\mathcal{E}}$ corresponds to a $(\tau, \mathcal{E})$-optimal orthogonal representation $\mathcal{R}$ of G, with costs differing by the total base costs, i.e., $\text{cost}(\mathcal{R}) = \text{cost}(\phi) + b_\tau$. We obtain the following lemma, where $T_{\text{flow}}(\cdot)$ is time necessary to compute an optimal flow.

Lemma 5. *Let G be a biconnected positive-convex instance of* OptimalFlexDraw*, let $\mathcal{T}$ be its SPQR-tree with root τ and let $\mathcal{E}$ be an embedding of* $\text{skel}(\tau)$*. If the cost function of every principal split component is known, a $(\tau, \mathcal{E})$-optimal solution can be computed in $O(T_{\text{flow}}(|\,\text{skel}(\tau)|))$ time.*

It remains to show that Theorem 4 holds. We make a structural induction over the SPQR-tree. For the leaves it obviously holds as edges are required to have convex costs. For inner nodes we show the following lemma.

Lemma 6. *If Theorem 4 holds for each principal split component corresponding to a child of the node μ in the SPQR-tree, then it also holds for* $\text{pert}(\mu)$*.*

Proof (Sketch). In an orthogonal representation $\mathcal{S}$ of $H = \text{pert}(\mu)$, the number of bends ρ is determined by the rotation along one of the two paths $\pi(s, t)$ or $\pi(t, s)$. We define the *partial cost function* $\text{cost}_H^{\mathcal{E}}(\cdot)$ with respect to the embedding $\mathcal{E}$ of $\text{skel}(\mu)$ to be the smallest possible cost of an orthogonal representation inducing the planar embedding $\mathcal{E}$ on $\text{skel}(\mu)$ with ρ bends such that $\pi_f(s, t)$ determines the number of bends. We show how to compute $\text{cost}_H^{\mathcal{E}}(\cdot)$ using a flow network similar to $N^{\mathcal{E}}$. It can be shown that these partial cost functions are convex, and that their minimum $\text{cost}_H(\cdot)$ defined by $\text{cost}_H(\rho) = \min_{\mathcal{E}} \text{cost}_H^{\mathcal{E}}(\rho)$ is convex.

We define $N^{\mathcal{E}}$ as before with two changes. First, the parent edge plays a special role as it should not occur in the resulting orthogonal representation. Removing some arcs and adjusting the demands accordingly yields a flow network such that

an optimal flow corresponds to an optimal orthogonal representation. Second, the flow network is parameterized as follows. The incoming flows at the two face-nodes corresponding to the faces incident to the parent edge are equal to the rotations along the paths $\pi(s,t)$ and $\pi(t,s)$ in a corresponding orthogonal representation. We parameterize $N^{\mathcal{E}}$ with respect to these two faces. It can then be shown that the cost function of the flow and the partial cost function of H coincide on the interval $[\ell_H, 3]$ up to the total base cost. Thus, convexity for the partial cost function follows from the convexity of the cost function of a parametrized flow network if $\text{cost}_{N^{\mathcal{E}}}(\rho)$ is minimal for $\rho = \ell_H$; see Theorem 1. We note that this is not obvious and not even true if $\deg(s) = \deg(t) = 3$. However, it is true for all other cases. Moreover, we can show that the minimum over all partial cost functions is convex on the interval $[\ell_H, 3]$. This is again not obvious and not even true for a larger interval [3]. □

The proof of Lemma 6 yields an algorithm computing a $(\tau, \mathcal{E})$-optimal solution bottom-up in the SPQR-tree. In each node μ a constant number of optimal flows in a network of size $|\,\text{skel}(\mu)|$ has to be computed, consuming overall $O(T_{\text{flow}}(n))$ time. Applying this algorithm $O(n)$ times yields an optimal drawing.

Theorem 5. OptimalFlexDraw *can be solved in* $O(n \cdot T_{\text{flow}}(n))$ *time for positive-convex biconnected instances.*

We can extend our algorithm to the case where G contains cutvertices (an extensive description is in the appendix), yielding the following theorem.

Theorem 6. OptimalFlexDraw *can be solved in* $O(n^2 \cdot T_{\text{flow}}(n))$ *time for positive-convex instances.*

References

1. Biedl, T., Kant, G.: A Better Heuristic for Orthogonal Graph Drawings. Comput. Geom. 9(3), 159–180 (1998)
2. Bläsius, T., Krug, M., Rutter, I., Wagner, D.: Orthogonal graph drawing with flexibility constraints. Algorithmica (2012), doi:10.1007/s00453-012-9705-8
3. Bläsius, T., Rutter, I., Wagner, D.: Optimal orthogonal graph drawing with convex bend costs. CoRR abs/1204.4997 (2012), `http://arxiv.org/abs/1204.4997`
4. Di Battista, G., Liotta, G., Vargiu, F.: Spirality and Optimal Orthogonal Drawings. SIAM J. Comput. 27(6), 1764–1811 (1998)
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.: On-Line Planarity Testing. SIAM J. Comput. 25(5), 956–997 (1996)
7. Garg, A., Tamassia, R.: On the Computational Complexity of Upward and Rectilinear Planarity Testing. SIAM J. Comput. 31(2), 601–625 (2001)
8. Gutwenger, C., Mutzel, P.: A Linear Time Implementation of SPQR-Trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
9. Tamassia, R.: On Embedding a Graph in the Grid with the Minimum Number of Bends. SIAM J. Comput. 16(3), 421–444 (1987)

Deterministic Single Exponential Time Algorithms for Connectivity Problems Parameterized by Treewidth*

Hans L. Bodlaender[1], Marek Cygan[2], Stefan Kratsch[3], and Jesper Nederlof[1]

[1] Utrecht University, The Netherlands
{H.L.Bodlaender,J.Nederlof}@uu.nl
[2] University of Warsaw, Poland
cygan@mimuw.edu.pl
[3] Technical University Berlin, Germany
stefan.kratsch@tu-berlin.de

Abstract. It is well known that many local graph problems, like Vertex Cover and Dominating Set, can be solved in $2^{\mathcal{O}(\mathtt{tw})}n^{\mathcal{O}(1)}$ time for graphs with a given tree decomposition of width $\mathtt{tw}$. However, for nonlocal problems, like the fundamental class of connectivity problems, for a long time it was unknown how to do this faster than $\mathtt{tw}^{\mathcal{O}(\mathtt{tw})}n^{\mathcal{O}(1)}$ until recently, when Cygan et al. (FOCS 2011) introduced the Cut&Count technique that gives randomized algorithms for a wide range of connectivity problems running in time $c^{\mathtt{tw}}n^{\mathcal{O}(1)}$ for a small constant c.

In this paper, we show that we can improve upon the Cut&Count technique in multiple ways, with two new techniques. The first technique (*rank-based approach*) gives deterministic algorithms with $O(c^{\mathtt{tw}}n)$ running time for connectivity problems (like Hamiltonian Cycle and Stei-ner Tree) *and* for *weighted* variants of these; the second technique (*determinant approach*) gives deterministic algorithms running in time $c^{\mathtt{tw}}n^{\mathcal{O}(1)}$ for *counting* versions, e.g., counting the number of Hamiltonian cycles for graphs of treewidth $\mathtt{tw}$.

The rank-based approach introduces a new technique to speed up dynamic programming algorithms which is likely to have more applications. The determinant-based approach uses the Matrix Tree Theorem for deriving closed formulas for counting versions of connectivity problems; we show how to evaluate those formulas via dynamic programming.

1 Introduction

It is known since the 1980s that many ($\mathcal{NP}$-hard) problems allow algorithms with a running time of the type $f(\mathtt{tw})n^c$ on graphs with n vertices and a tree decomposition of width $\mathtt{tw}$. It is a natural question how we can optimize on

* The second author is partially supported by NCN grant DEC-2012/05/D/ST6/03214 and Foundation for Polish Science. Part of the work of the third author was done at Utrecht University supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), project: 'KERNELS'. The fourth author is supported by the NWO project 'Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms'.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 196–207, 2013.

the dependency of the treewidth, i.e., we first aim at obtaining a small growing (but, since we assume $\mathcal{P} \neq \mathcal{NP}$, exponential) function $f(\mathtt{tw})$ and second a small exponent c. For problems with locally checkable certificates, that is, certificates assigning a constant number of bits per node that can be checked by a cardinality check and iteratively looking at all neighborhoods of the input graph[1], it quickly became clear that $f(\mathtt{tw})$ only needs to be single exponential. See [14,20] for sample applications to the Independent Set/Vertex Cover problems. From the work of [13] and known Karp-reductions between problems it follows that this dependence cannot be improved to subexponential algorithms unless the Exponential Time Hypothesis (ETH) fails. In [17] it was shown that under the Strong ETH (SETH) the current algorithms are optimal even with respect to the bases of the exponential dependence on the treewidth, that is, problems with current best running time $c^{\mathtt{tw}}n^{\mathcal{O}(1)}$ cannot be solved in $(c-\epsilon)^{\mathtt{tw}}n^{\mathcal{O}(1)}$ for positive ϵ where, e.g., $c = 2$ for Independent Set and $c = 3$ for Dominating Set.

A natural class of problems that does not have locally checkable certificates are connectivity problems such as Hamiltonian Cycle and Steiner Tree (see for example [11, Section 5]), begging the question whether these can be solved within single exponential dependence on $\mathtt{tw}$ as well. Early results on the special case of H-minor-free graphs were given in [9]. A positive answer to the question was found by Cygan et al. [8] using a randomized approach termed "Cut & Count": It provided a transformation of the natural certificates to "cut-certificates" transforming the connectivity requirement into a locally checkable requirement. The transformation is only valid modulo 2, but by a standard technique introducing randomization [19], the decision variant can be reduced to the counting modulo 2 variant. This result was considered surprising since in the folklore $2^{\mathcal{O}(\mathtt{tw}\log\mathtt{tw})}n^{\mathcal{O}(1)}$ dynamic programming routines for connectivity problems all information stored seemed needed: Given two partial solutions inducing different connectivity properties, one can be extendable to a solution while the other one can not (this resembles the notion of Myhill-Nerode equivalence classes [12]).

The Cut & Count approach is one of the dynamic programming algorithms using a modulo 2 based transformation [3,4,16,15,18,23]. These algorithms give the smallest running times currently known, but have several disadvantages compared to traditional dynamic programming algorithms: (a) They are randomized. (b) The dependence on the inputs weights in weighted extensions is pseudo-polynomial. (c) They do not extend to counting the number of witnesses. (d) They do not give intuition for the optimal substructure / equivalence classes. An additional disadvantage of the Cut & Count approach of [8], compared to traditional dynamic programming algorithms on tree decompositions, is that their dependence in terms of the input graph is superlinear. Our work shows that each of these disadvantages can be overcome, with two different approaches, both giving deterministic algorithms for connectivity problems that are single exponential in the treewidth.

[1] E.g., for the odd cycle transversal problem that asks to make the input graph bipartite by removing at most k vertices, a locally checkable certificate would be a solution set combined with a proper two-coloring of the remaining graph.

Table 1. Our results for some famous computational problems. The second column gives running times when a path decomposition of width $\mathtt{pw}$ is given in the input; the third column gives running times when a tree decomposition of width $\mathtt{tw}$ is given in the input. Rows 1–4 use the rank-based approach; Rows 5–7 the determinant approach. ω denotes the matrix multiplication exponent (currently it is known that $\omega < 2.3727$ [24]).

1	Weighted Steiner Tree	$n(1+2^{\omega})^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}$	$n(1+2^{\omega+1})^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}$
2	Traveling Salesman	$n(2+2^{\omega/2})^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}$	$n(5+2^{(\omega+2)/2})^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}$
3	k-Path	$n(2+2^{\omega/2})^{\mathtt{pw}}(k+\mathtt{pw})^{\mathcal{O}(1)}$	$n(5+2^{(\omega+2)/2})^{\mathtt{tw}}(k+\mathtt{tw})^{\mathcal{O}(1)}$
4	Feedback Vertex Set	$n(1+2^{\omega})^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}$	$n(1+2^{\omega+1})^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}$
5	# Hamiltonian Cycle	$\tilde{\mathcal{O}}(6^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}n^2)$	$\tilde{\mathcal{O}}(15^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}n^2)$
6	# Steiner Tree	$\tilde{\mathcal{O}}(5^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}n^3)$	$\tilde{\mathcal{O}}(10^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}n^3)$
7	Feedback Vertex Set	$\tilde{\mathcal{O}}(5^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}n^3)$	$\tilde{\mathcal{O}}(10^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}n^3)$

Our Contribution. We present the "Rank based" and "Squared determinant" approaches. For a number of key problems, we state the results obtained by applying these approaches in Table 1.

The approaches can be used to quickly and deterministically solve weighted and counting versions of problems solved by the Cut & Count approach. Additional advantages of the rank based approach are that it gives a more intuitive insight in the optimal substructure / equivalence classes of a problem and that is has only a linear dependence on the input graph in the running time. The only disadvantage of both approaches when compared to the Cut & Count approach is that the dependence on the treewidth or pathwidth in the running time is slightly worse. However, although we did not manage to overcome it, this disadvantage might not be inherently due to the new methods. Due to the generality of our key ideas, as one might expect, the approaches can be applied to other connectivity problems, such as all problems mentioned in [8]. However, our methods may inspire future work not involving tree decompositions as well.

Since one of the main strengths of the treewidth concept seems to be its ubiquity, it is perhaps not surprising that our results improve, simplify, generalize or unify a number of seemingly unrelated results. E.g., we unify algorithms for Feedback Vertex Set [6] and k-Path [2] and generalize algorithms for restricted inputs such as H-minor-free (e.g. [9]) or bounded degree graphs (e.g. [10]). A more detailed discussion is postponed to the full version.

For space reasons, many details (and material that might be considered 'more than details') are omitted from this extended abstract. For all such material, we refer to the full paper, available at arXiv.org [5].

2 Preliminaries

Partitions and the Partition Lattice. Given a base set U, we use $\Pi(U)$ for the set of all partitions of U. It is known that, together with the coarsening relation $\sqsubseteq$, $\Pi(U)$ gives a lattice, with the minimum element being $\{U\}$ and the maximum element being the partition into singletons. We denote $\sqcap$ for the

meet operation and $\sqcup$ for the join operation in this lattice; these operators are associative and commutative. We use $\Pi_2(U) \subset \Pi(U)$ to denote the set of all partitions of U in blocks of size 2, or equivalently, the set of perfect matchings over U. Given $p \in \Pi(U)$, we let $\#\texttt{blocks}(p)$ denote the number of blocks of p. If $X \subseteq U$ we let $p_{\downarrow X} \in \Pi(X)$ be the partition obtained by removing all elements not in X from it, and analogously we let for $U \subseteq X$ denote $p_{\uparrow X} \in \Pi(X)$ for the partition obtained by adding singletons for every element in $X \setminus U$ to p. Also, for $X \subseteq U$, we let $U[X]$ be the partition of U where one block is $\{X\}$ and all other blocks are singletons. If $a, b \in U$ we shorthand $U[ab] = U[\{a, b\}]$. The empty set, vector and partition are all denoted by $\emptyset$.

Tree Decompositions and Treewidth. A *tree decomposition* [22] of a graph G is a tree $\mathbb{T}$ in which each node x has an assigned set of vertices $B_x \subseteq V$ (called a *bag*) such that $\bigcup_{x \in \mathbb{T}} B_x = V$ with the following properties: (i) for any $uv \in E$, there exists an $x \in \mathbb{T}$ such that $u, v \in B_x$, (ii) if $v \in B_x$ and $v \in B_y$, then $v \in B_z$ for all z on the (unique) path from x to y in $\mathbb{T}$. In a *nice tree decomposition*, $\mathbb{T}$ is rooted, and each bag is of one of the following types:

- **Leaf Bag:** a leaf x of $\mathbb{T}$ with $B_x = \emptyset$.
- **Introduce Vertex Bag:** an internal vertex x of $\mathbb{T}$ with one child vertex y for which $B_x = B_y \cup \{v\}$ for some $v \notin B_y$.
- **Introduce Edge Bag:** an internal vertex x of $\mathbb{T}$ labeled with an edge $uv \in E$ with one child bag y for which $u, v \in B_x = B_y$.
- **Forget Bag:** an internal vertex x of $\mathbb{T}$ with one child bag y for which $B_x = B_y \setminus \{v\}$ for some $v \in B_y$.
- **Join Bag:** an internal vertex x with two child vertices l and r with $B_x = B_r = B_l$.

We require that every edge in E is introduced exactly once. This variant of nice tree decompositions was also used by Cygan et al. [8].

A nice tree decomposition is a *nice path decomposition* if it does not contain join bags. The *width* $tw(\mathbb{T})$ of a (nice) tree decomposition $\mathbb{T}$ is the size of the largest bag of $\mathbb{T}$ minus one, and the treewidth (pathwidth) of a graph G can be defined as the minimum treewidth over all nice tree decompositions (nice path decompositions) of G.

In this paper, we will always assume that nice tree decompositions of the appropriate width are given. To each bag x in a nice tree decomposition $\mathbb{T}$, we associate the graph $G_x = (V_x, E_x)$ with V_x the union of all bags B_y with y a descendant of x, and E_x the set of all edges introduced in an descendant of x.

Further Notation. For two integers a, b we use $a \equiv b$ to indicate that a is even if and only if b is even. We use $\mathbb{N}$ to denote the set of all non-negative integers. We use Iverson's bracket notation: if p is a predicate we let $[p]$ be 1 if p is true and 0 otherwise. If $\omega : U \to \{1, \ldots, N\}$, we shorthand $\omega(S) = \sum_{e \in S} \omega(e)$ for $S \subseteq U$. For a function/vector s by $s[v \to \alpha]$ we denote the function $s \setminus \{(v, s(v))\} \cup \{(v, \alpha)\}$. Note that this definition works regardless of whether $s(v)$ is already defined or not. We use either $s|_X$ or $s_{|X}$ to denote the function obtained by restricting the domain to X.

3 The Rank Based Approach

3.1 Main Ideas of the Approach

Recall that a dynamic programming algorithm fixes a way to decompose certificates into 'partial certificates', and builds partial certificates in a bottom-up manner while storing only their essential information. Given some language $L \subseteq \{0,1\}^*$, this is typically implemented by defining an equivalence $\sim$ on partial certificates $x, y \in \{0,1\}^k$ such that $x \sim y$ if $xz \in L \leftrightarrow yz \in L$, for every extension $z \in \{0,1\}^l$. For connectivity problems on treewidth, the number of non-equivalent certificates can be seen to be $2^{\Theta(\mathtt{tw} \cdot \lg \mathtt{tw})}$. See for example [21] for a lower bound in communication complexity.

We will use however, that sometimes we can represent the joint essential information for sets of partial certificates more efficiently than naively representing essential information for every partial certificate separately. The rank based approach achieves this as follows: Given a dynamic programming algorithm, consider the matrix A whose rows and columns are indexed by partial certificates, with $A[x, y] = 1$ if and only if $xy \in L$. Then observe that if a set of rows $X \subseteq \{0,1\}^n$ is linearly dependent (modulo 2), any partial certificate $x \in X$ is redundant in the sense that if $xz \in L$, there will be $y \in X, y \neq x$ with $yz \in L$. Hence, the essential information can be reduced to $\mathrm{rk}(A)$ partial certificates.

The second ingredient to our approach are proofs that for the considered problems, the rank (working in GF(2)) of such a matrix A is single exponential in the treewidth, and moreover, we can give explicit bases.

Now, the approach is as follows: take the 'usual' dynamic programming algorithm for the problem at hand, but add the following step: after each computation of a table at a bag node, form the submatrix of A with for each entry in the table a row and for each element of the basis a column; find a row-basis of this matrix and continue with only the partial certificates in this basis, of which there are not more than the rank of the certificate matrix. This is easily extended to weighted problems using a minimum weighted row-basis. For getting this approach to work for connectivity problems we require an upper bound on the rank of the matrix $\mathcal{M}$ defined as follows: Fix a ground set U and let p and q be partitions of U (p and q represent connectivity induced by partial solutions), define $\mathcal{M}[p, q]$ to be 1 if and only if the meet of p and q is the trivial partition, that is, if the union of the partial solutions induce a connected solution. Although this matrix has dimensions of order $2^{\theta(|U| \log |U|)}$, we exploit a simple factorization in GF(2) of matrices with inner dimension $2^{|U|}$ using an idea of [8]. To avoid creating a series of ad hoc results for single problems, we introduce a collection of operations on sets of weighted partitions, such that our results apply to any dynamic programming (DP) formulation that can be expressed using these operators only (see Section 3.2). In this extended abstract, we only state and illustrate the main building blocks of the approach.

3.2 Operators on Sets of Weighted Partitions

Recall that $\Pi(U)$ denotes the set of all partitions of some set U. A *set of weighted partitions* is a set $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$, i.e., a family of pairs, each consisting of a partition of U and a non-negative integer weight.

We now define a collection of operators on sets of weighted partitions. The operators naturally apply to connectivity problems by allowing, e.g., gluing of connected components (i.e., different sets in a partition), or joining of two partial solutions by taking the meet operation $\sqcap$ on the respective partitions.

An important reason of interest in these operators is the following: if the recurrences in a dynamic programming algorithm on a tree decomposition only use these operators, then the naive algorithm evaluating the recurrence can be improved beyond the typical $2^{\Omega(\mathtt{tw}\cdot\log \mathtt{tw})}$ that comes from the high number of different possible partial solutions; and we typically get a running time of $O(c^{\mathtt{tw}}n)$.

For notational ease, we let $\mathtt{rmc}(\mathcal{A})$ denote the set obtained by removing non-minimal weight copies, i.e., $\mathtt{rmc}(\mathcal{A}) = \{(p,w) \in \mathcal{A} \mid \nexists (p,w') \in \mathcal{A} \wedge w' < w\}$.

Definition 1. *Let U be a set and $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$.*

- ***Union.*** *For $\mathcal{B} \subseteq \Pi(U) \times \mathbb{N}$, define $\mathcal{A} \uplus \mathcal{B} = \mathtt{rmc}(\mathcal{A} \cup \mathcal{B})$.*
- ***Insert.*** *For $X \cap U = \emptyset$, define $\mathtt{ins}(X, \mathcal{A}) = \{(p_{\uparrow U \cup X}, w) \mid (p,w) \in \mathcal{A}\}$.*
- ***Shift.*** *For $w' \in \mathbb{N}$ define $\mathtt{shft}(w', \mathcal{A}) = \{(p, w+w') \mid (p,w) \in \mathcal{A}\}$.*
- ***Glue.*** *For u, v, let $\hat{U} = U \cup \{u,v\}$ and define $\mathtt{glue}(uv, \mathcal{A}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$ as $\mathtt{glue}(uv, \mathcal{A}) = \mathtt{rmc}(\{(\hat{U}[uv] \sqcap p_{\uparrow \hat{U}}, w) \mid (p,w) \in \mathcal{A}\})$. Also, if $\omega : \hat{U} \times \hat{U} \to \mathbb{N}$, let $\mathtt{glue}_\omega(uv, \mathcal{A}) = \mathtt{shft}(\omega(u,v), \mathtt{glue}(uv, \mathcal{A}))$.*
- ***Project.*** *For $X \subseteq U$ let $\overline{X} = U \setminus X$, and define $\mathtt{proj}(X, \mathcal{A}) \subseteq \Pi(\overline{X}) \times \mathbb{N}$ as $\mathtt{proj}(X, \mathcal{A}) = \left\{ (p_{\downarrow \overline{X}}, w) \,\middle|\, (p,w) \in \mathcal{A} \wedge \forall e \in X : \exists e' \in \overline{X} : p \sqsubseteq U[ee'] \right\}$.*
- ***Join.*** *For $\mathcal{B} \subseteq \Pi(U') \times \mathbb{N}$ let $\hat{U} = U \cup U'$ and define $\mathtt{join}(\mathcal{A}, \mathcal{B}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$ as $\mathtt{join}(\mathcal{A}, \mathcal{B}) = \mathtt{rmc}(\left\{ (p_{\uparrow \hat{U}} \sqcap q_{\uparrow \hat{U}}, w_1 + w_2) \,\middle|\, (p,w_1) \in \mathcal{A} \wedge (q, w_2) \in \mathcal{B} \right\})$.*

See the full version for a description of the operators in words. Using straightforward implementation each of the operations union, shift, insert, glue and project can be performed in $S|U|^{\mathcal{O}(1)}$ time where S is the size of the input of the operation. Given $\mathcal{A}$ and $\mathcal{B}$, $\mathtt{join}(\mathcal{A}, \mathcal{B})$ can be computed in time $|\mathcal{A}| \cdot |\mathcal{B}| \cdot |U|^{\mathcal{O}(1)}$.

3.3 Representing Collections of Partitions

The key idea for getting a faster dynamic programming algorithm is to follow the naive DP, but to consider only small representative sets of weighted partitions instead of all weighted partitions that would be considered by the naive DP. Intuitively, a representative (sub)set of partial solutions should allow us to always extend to an optimal solution provided that one of the complete set of partial solutions extends to it. Let us define this formally.

Definition 2. *Given a set of weighted partitions $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$ and a partition $q \in \Pi(U)$, define $\mathtt{opt}(q, \mathcal{A}) = \min\{w \mid (p,w) \in \mathcal{A} \wedge p \sqcap q = \{U\}\}$. For another set of weighted partitions $\mathcal{A}' \subseteq \Pi(U) \times \mathbb{N}$, we say that $\mathcal{A}'$* represents *$\mathcal{A}$ if for all $q \in \Pi(U)$ it holds that $\mathtt{opt}(q, \mathcal{A}') = \mathtt{opt}(q, \mathcal{A})$.*

Note that the definition of representation is symmetric, i.e., if $\mathcal{A}'$ represents $\mathcal{A}$ then $\mathcal{A}$ also represents $\mathcal{A}'$. However, we will only be interested in the special case where $\mathcal{A}' \subseteq \mathcal{A}$ and where we have a size guarantee for finding a small such subset $\mathcal{A}'$.

Definition 3. *A function* $f : 2^{\Pi(U)\times\mathbb{N}} \times Z \to 2^{\Pi(U')\times\mathbb{N}}$ *is said to* preserve representation *if for every* $\mathcal{A}, \mathcal{A}' \subseteq \Pi(U) \times \mathbb{N}$ *and* $z \in Z$ *it holds that if* $\mathcal{A}'$ *represents* $\mathcal{A}$ *then* $f(\mathcal{A}', z)$ *represents* $f(\mathcal{A}, z)$*. (Note that* Z *stands for any combination of further inputs.)*

The following lemma and theorem are our main building blocks and establish that the operations needed for the DP preserve representation, and, crucially, that we can always find a reasonably small representative set of weighted partitions. The proofs are deferred to the full version.

Lemma 1. *The union, insert, shift, glue, project, and join operations from Definition 1 preserve representation.*

Theorem 1. *There is an algorithm* `reduce()` *that given set of weighted partitions* $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$ *takes time* $|\mathcal{A}|2^{(\omega-1)|U|}|U|^{\mathcal{O}(1)}$ *and outputs a set of weighted partitions* $\mathcal{A}' \subseteq \mathcal{A}$ *such that* $\mathcal{A}'$ *represents* $\mathcal{A}$ *and* $|\mathcal{A}'| \leq 2^{|U|}$*, where* ω *denotes the matrix multiplication exponent (it is known that* $\omega < 2.3727$ *[24]).*

With help of Lemma 1 and Theorem 1, we can now handle problems in the following way: we give recurrences for the different types of bags (leaf, introduce, etc.) in a nice tree (or path) decompositions that use only union, insert, shift, glue, project, and join operations. Correctness of an algorithm that interleaves the computation of the recurrences with the (Gaussian elimination based) reduction step of Theorem 1 follows. Inspection of the resulting algorithms is still needed for establishing the precise time bounds.

3.4 Application to Steiner Tree

We now sketch how to solve the Steiner Tree problem via a dynamic programming formulation that requires only the operators introduced in Section 3.4.

Steiner Tree
Input: A graph $G = (V, E)$ weight function $\omega\colon E \to \mathbb{N} \setminus \{0\}$, a terminal set $K \subseteq V$ and a nice tree decomposition $\mathbb{T}$ of G of width `tw`.
Question: The minimum of $\omega(X)$ over all subsets $X \subseteq E$ of G such that $G[X]$ is connected and $K \subseteq V(G[X])$.

We now describe the 'folklore' dynamic programming algorithm for Steiner Tree on nice tree decompositions. For each bag x, we compute a table A_x. A_x has an entry for each $\mathbf{s} \in \{0,1\}^{B_x}$; this entry is a set of pairs consisting of a partition of $s^{-1}(1)$ and a weight value. The intuition is as follows: a 'partial solution' for the Steiner tree problem is a forest F in G_x that contains all vertices in K and each tree in the forest has a nonempty intersection with B_x; now $\mathbf{s}$

denotes which vertices in B_x belong to F, i.e., $v \in B_x$ belongs to F, iff $s(v) = 1$; and the partition tells which vertices in $s^{-1}(1)$ belong to the same tree in F. The weight value gives the total weight of all edges in F; for given $\mathbf{s}$ and partition p, we only store the minimum weight over all applicable forests.

Formally, we define, for a bag x, and $\mathbf{s} \in \{0,1\}^{B_x}$

$$A_x(\mathbf{s}) = \left\{ \left(p, \min_{X \in \mathcal{E}_x(p,\mathbf{s})} \omega(X) \right) \middle| \, p \in \Pi(s^{-1}(1)) \wedge \mathcal{E}_x(p,\mathbf{s}) \neq \emptyset \right\}$$

with $\mathcal{E}_x(p,\mathbf{s}) = \{X \subseteq E_x |\ \forall v \in B_x : v \in V(G[X]) \vee v \in K \rightarrow s(v) = 1 \wedge \forall v_1, v_2 \in s^{-1}(1) : v_1 v_2$ are in same block in $p \leftrightarrow v_1, v_2$ connected in $G[X] \wedge \#\mathtt{blocks}(p) = \mathtt{cc}(G[X])\}$.

The table A_x can be computed from the tables A_y of all the children of y, using only the union, insert, shift, glue, project, and join operations. For each of the different types of bags (leaf, forget, etc.), we can give such a recurrence. We only state here the recurrence for the Introduce edge bag; for the other recurrences, we refer to the full version.

Consider a bag x with child y introducing edge $e = uv$. One can show the following recurrence, which is, by the results above, representation preserving.

$$A_x(\mathbf{s}) = \begin{cases} A_y(\mathbf{s}) & \text{if } s(u) = 0 \vee s(v) = 0, \\ A_y(\mathbf{s}) \uplus \mathtt{glue}_\omega(uv, A_y(\mathbf{s})), & \text{otherwise.} \end{cases}$$

Theorem 2. *There exist algorithms that solve* Steiner Tree *in time* $n(1 + 2^\omega)^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}$ *time if a path decomposition of width* $\mathtt{pw}$ *of* G *is given, and in time* $n(1 + 2^{\omega+1})^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}$ *time if a tree decomposition of width* $\mathtt{tw}$ *of* G *is given.*

Proof. The algorithm is the following: use the above dynamic programming formulation as discussed to compute A_r (where r is the child of the root, as discussed), but after evaluation of any entry A_x, use Theorem 1 to obtain and store $A'_x = \mathtt{reduce}(A_x)$ rather than A_x. Since $A_x = \mathtt{reduce}(\mathcal{A})$ represents $\mathcal{A}$ and the recurrence uses only the operators defined in Definition 1 which all preserve representation by Lemma 1, we have as invariant that for every $x \in \mathbb{T}$ the entry A'_x stored for A_x represents A_x by Lemma 1. In particular, the stored value $A'_r(\mathbf{s})$ represents $A_r(\mathbf{s})$ and hence we can safely read off the answer to the problem from this stored value as done from $A_r(\mathbf{s})$ in the folklore dynamic programming algorithm. The time analysis can be found in the full version. □

3.5 Further Results

A large number of other problems can be handled with the rank based approach. In Table 1 we give a number of key results. Details are in the full version. The results on Hamiltonian Cycle and Traveling Salesman are obtained by combining our approach with the following result.

Theorem 3 ([7]). *Let* $\mathcal{H}$ *be the submatrix of* $\mathcal{M}$ *restricted to all matchings. Then* $\mathcal{H}$ *can be factorized into two matrices whose entries can be computed in time* $|U|^{\mathcal{O}(1)}$*, where the inner dimension of the factorization is* $2^{|U|/2-1}$.

An interesting corollary of our work is the following.

Theorem 4. *There is an algorithm solving* Traveling Salesman *on cubic graphs in* $1.2186^n n^{\mathcal{O}(1)}$ *time.*

4 Determinant Approach

In this section we will present the determinant approach that can be used to solve counting versions of connectivity problems on graphs of small treewidth. Throughout this section, we will assume a graph G along with a path/tree decomposition $\mathbb{T}$ of G of width `pw` or `tw` is given.

4.1 Main Ideas of the Approach

The determinant approach gives a generic transformation of counting connected objects to a more local transformation. In [8] the existence of such a transformation in GF(2) was already given. For extending this to counting problems, we will need three key insights. The first insight is that (a variant of) Kirchhoff's Matrix Tree Theorem gives a reduction from counting connected objects to computing a sum of determinants. However, we cannot fully control the contribution of a connected object (it will appear to be either 1 or -1). To overcome this we ensure that every connected object contributed exactly once, we compute a sum of *squares of determinants*. The last obstacle is that the computation of a determinant is not entirely local (in the sense that we can verify a contribution by iteratively considering its intersection with all bags) since we have to account for the number of inversions of a permutation in every summand of the determinant. To overcome this obstacle, we show that this computation becomes a local computation once we have fixed the order of the vertices in a proper way.

Formally, let A be an incidence matrix of an orientation of G, that is $A = (a_{i,j})$ is a matrix with n rows and m columns. Each row of A is indexed with a vertex and each column of A is indexed with an edge. The entry $a_{v,e}$ is defined to be 0 if $v \notin e$; -1 if $e = uv$ and $u < v$; or 1 if $e = uv$ and $u > v$. We assume, that all the vertices are ordered with respect to the post-ordering of their forget nodes in the given tree decomposition, that is vertices forgotten in a left subtree are smaller than vertices forgotten in the right subtree, and a vertex forgotten in a bag x is smaller than a vertex forgotten in a bag which is an ancestor of x. Similarly we order edges according to the post-ordering of the introduce edge nodes in the tree decomposition.

Let v_1 be an arbitrary fixed vertex and let F be the matrix A with the row corresponding to v_1 removed. For a subset $S \subseteq E$ let F_S be the matrix with $n-1$ rows and $|S|$ columns, whose columns are those of F with indices in S. The following folklore lemma is used in the proof of the Matrix Tree Theorem (see for example [1, Page 203] where our matrix is denoted by N).

Lemma 2. *Let $S \subseteq E$ be a subset of size $n-1$. If (V, S) is a tree, then $|\det(F_S)| = 1$ and $\det(F_S) = 0$ otherwise.*

We are up to compute the number of connected edgesets X such that $X \in \mathcal{F}$ where $\mathcal{F}$ is some implicitly defined set family. Our main idea is to use Lemma 2 to reduce this task to computing the quantity $\sum_{X \in \mathcal{F}} \det(F_X)^2$ instead, and to ensure that if $X \in \mathcal{F}$ is connected, then it is a tree.

For two (not necessarily disjoint) subsets V_1, V_2 of an ordered set let us define $\mathrm{inv}(V_1, V_2) = |\{(u, v) : u \in V_1, v \in V_2, u > v\}|$. If X, Y are ordered sets, recall that for a permutation $f : X \overset{1-1}{\to} Y$ we have that the sign equals $\mathrm{sgn}(f) = (-1)^{|\{(e_1,e_2):e_1,e_2 \in S \wedge e_1 < e_2 \wedge f(e_1) > f(e_2)\}|}$. The following proposition will be useful:

Proposition 1. *Let $X_l, X_r \subseteq V$ and $Y_l, Y_r \subseteq E$ such that $X_l \cap X_r = \emptyset$ and $Y_l \cap Y_r = \emptyset$, and for every $e_1 \in Y_l$ and $e_2 \in Y_r$ we have that $e_1 < e_2$. Suppose $f_l : Y_l \overset{1-1}{\to} X_l$ and $f_r : Y_r \overset{1-1}{\to} X_r$. Denote $f = f_l \cup f_r$, that is, $f(v) = f_l(v)$ if $v \in Y_l$ and $f(v) = f_r(v)$ if $v \in Y_r$. Then it holds that $\mathrm{sgn}(f) = \mathrm{sgn}(f_1)\mathrm{sgn}(f_2) \cdot (-1)^{\mathrm{inv}(X_l, X_r)}$.*

To see that the proposition is true, note that from the definition of sgn, the pairs e_1, e_2 with $e_1, e_2 \in Y_1$ or $e_1, e_2 \in Y_2$ are already accounted for in the part $\mathrm{sgn}(f_1)\mathrm{sgn}(f_2)$ so it remains to show that $|\{(e_1, e_2) : e_1 \in Y_1, e_2 \in Y_2 \wedge e_1 < e_2 \wedge f(e_1) > f(e_2)\}|$ indeed equals $\mathrm{inv}(X_l, X_r)$, which follows easily from the assumption that $e_1 < e_2$.

4.2 Counting Hamiltonian Cycles

For our first application to counting Hamiltonian cycles, we derive the following formula which expresses the number of Hamiltonian cycles of a graph. (We use that a 2-regular graph has n subtrees on $n-1$ edges if it is connected and 0 otherwise, and Lemma 2.)

$$\begin{aligned}
&\sum_{X \subseteq E; \forall_{v \in V} \deg_X(v) = 2} [X \text{ is a Hamiltonian cycle}] \\
&= \frac{1}{n} \cdot \sum_{X \subseteq E; \forall_{v \in V} \deg_X(v) = 2} \sum_{S \subseteq X, |S| = n-1} [(V, S) \text{ is a tree}] \\
&= \frac{1}{n} \cdot \sum_{X \subseteq E; \forall_{v \in V} \deg_X(v) = 2} \sum_{S \subseteq X, |S| = n-1} \det(F_S)^2.
\end{aligned}$$

By plugging in the permutation definition of a determinant, we obtain the following expression for the number of Hamiltonian cycles of a graph:

$$\begin{aligned}
&\frac{1}{n} \sum_{X \subseteq E; \forall_{v \in V} \deg_X(v) = 2} \sum_{S \subseteq X; |S| = n-1} \Big(\sum_{f : S \overset{1-1}{\to} V \setminus \{v_1\}} \mathrm{sgn}(f) \prod_{e \in S} a_{f(e), e} \Big)^2 \\
=&\frac{1}{n} \sum_{\substack{X \subseteq E \\ \forall_{v \in V} \deg_X(v) = 2}} \sum_{S \subseteq X} \sum_{f_1, f_2 : S \overset{1-1}{\to} V \setminus \{v_1\}} \mathrm{sgn}(f_1)\mathrm{sgn}(f_2) \prod_{e \in S} a_{f_1(e), e} a_{f_2(e), e}.
\end{aligned}$$

Note that in the last equality we dropped the assumption $|S| = n - 1$, as it follows from the fact that f_1 (and f_2) is a bijection.

Our goal is to compute the formula by dynamic programming over some nice tree decomposition $\mathbb{T}$. To this end, let us define a notion of "partial sum" of the above formula, that we will store in our dynamic programming table entries. For every bag $x \in \mathbb{T}$, $s_{\deg} \in \{0,1,2\}^{B_x}$, $s_1 \in \{0,1\}^{B_x}$ and $s_2 \in \{0,1\}^{B_x}$ define $A_x(s_{\deg}, s_1, s_2) =$

$$\sum_{\substack{X \subseteq E_x \\ \forall_{v \in (V_x \setminus B_x)} \deg_X(v)=2 \\ \forall_{v \in B_x} \deg_X(v)=s_{\deg}(v)}} \sum_{S \subseteq X} \sum_{\substack{f_1 : S \overset{1-1}{\to} (V_x \setminus \{v_1\}) \setminus s_1^{-1}(0) \\ f_2 : S \overset{1-1}{\to} (V_x \setminus \{v_1\}) \setminus s_2^{-1}(0)}} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \prod_{e \in S} a_{f_1(e),e} a_{f_2(e),e}.$$

Intuitively in $s_{\deg}$ we store the degrees of vertices of B_x in $G[X]$, whereas s_1 (and s_2) specify whether a vertex of B_x was already used as a value of the bijection f_1 (and f_2).

Showing that the terms $A_x(s_{\deg}, s_1, s_2)$ can be computed in the claimed time for the different types of nodes in a nice tree decompositions requires an intricate proof, for which we refer to the full version.

Theorem 5. *There exist algorithms that given a graph G solve* #Hamiltonian Cycle *in $\tilde{\mathcal{O}}(6^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}n^2)$ time if a path decomposition of width* `pw` *is given, and in time $\tilde{\mathcal{O}}(15^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}n^2)$ time if a tree decomposition of width* `tw` *is given.*

For other results that can be obtained with this approach, see the full paper.

5 Conclusions

In this paper, we have given deterministic algorithms for connectivity problems on graphs of small treewidth, with the running time only single exponential in the treewidth. We have given two different techniques. Each technique solves the standard versions, but for the counting and weighted variants, only one of the techniques appears to be usable. The rank-based approach gives a new twist to the dynamic programming approach, in the sense that we consider the "algebraic structure" of the partial certificates in a quite novel way. This suggests a study of this algebraic structure for dynamic programming algorithms for other problems.

In related work [7], an approach similar to the rank-based one, but focused on perfect matchings instead of partitions, is used to obtain a faster randomized algorithm for Hamiltonicity parameterized by pathwidth; the algorithm is showed to be tight under SETH, but does not apply to counting or the weighted case. The present results, while slower for the special case of Hamiltonicity, give deterministic algorithms that apply also to problems where connectivity of partial solutions does not appear to allow encoding via perfect matchings (e.g., Steiner Tree and Feedback Vertex Set), and to counting and weighted versions.

Acknowledgements. We thank Piotr Sankowski for pointing us to relevant literature on matrix-multiplication algorithms and Marcin Pilipczuk and Łukasz Kowalik for helpful discussions at an early stage of the paper.

References

1. Aigner, M., Ziegler, G.: Proofs from the book, Berlin, Germany (2010)
2. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM 42(4), 844–856 (1995)
3. Björklund, A.: Determinant sums for undirected Hamiltonicity. In: FOCS, pp. 173–182 (2010)
4. Björklund, A., Husfeldt, T., Taslaman, N.: Shortest cycle through specified elements. In: Rabani, Y. (ed.) SODA, pp. 1747–1753. SIAM (2012)
5. Bodlaender, H.L., Cygan, M., Kratsch, S., Nederlof, J.: Solving weighted and counting variants of connectivity problems parameterized by treewidth deterministically in single exponential time. CoRR, abs/1211.1505 (2012)
6. Cao, Y., Chen, J., Liu, Y.: On feedback vertex set new measure and new structures. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 93–104. Springer, Heidelberg (2010)
7. Cygan, M., Kratsch, S., Nederlof, J.: Fast Hamiltonicity checking via bases of perfect matchings (2012) (to appear at STOC 2013)
8. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: FOCS, pp. 150–159 (2011)
9. Dorn, F., Fomin, F.V., Thilikos, D.M.: Catalan structures and dynamic programming in H-minor-free graphs. J. Comput. Syst. Sci. 78(5), 1606–1622 (2012)
10. Eppstein, D.: The traveling salesman problem for cubic graphs. J. Graph Algorithms Appl. 11(1), 61–81 (2007)
11. Göös, M., Suomela, J.: Locally checkable proofs. In: PODC, pp. 159–168 (2011)
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. (2001)
13. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. 63(4), 512–530 (2001)
14. Kleinberg, J., Tardos, É.: Algorithm Design (2005)
15. Koutis, I.: Faster algebraic algorithms for path and packing problems. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 575–586. Springer, Heidelberg (2008)
16. Koutis, I., Williams, R.: Limits and applications of group algebras for parameterized problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 653–664. Springer, Heidelberg (2009)
17. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: SODA, pp. 777–789 (2011)
18. Lovász, L.: On determinants, matchings, and random algorithms. In: FCT, pp. 565–574 (1979)
19. Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. Combinatorica 7(1), 105–113 (1987)
20. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms (2002)
21. Raz, R., Spieker, B.: On the "log rank"-conjecture in communication complexity. Combinatorica 15(4), 567–588 (1995)
22. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. J. Comb. Theory 36(1), 49–64 (1984)
23. Williams, R.: Finding paths of length k in $\mathcal{O}^\star(2^k)$ time. Inf. Process. Lett. 109(6), 315–318 (2009)
24. Williams, V.V.: Multiplying matrices faster than coppersmith-winograd. In: STOC, pp. 887–898 (2012)

On the Complexity of Higher Order Abstract Voronoi Diagrams*

Cecilia Bohler[1], Panagiotis Cheilaris[2], Rolf Klein[1], Chih-Hung Liu[1], Evanthia Papadopoulou[2], and Maksym Zavershynskyi[2]

[1] University of Bonn, Institute of Computer Science I, D-53113 Bonn, Germany
[2] University of Lugano, Faculty of Informatics, CH-6904 Lugano, Switzerland

Abstract. Abstract Voronoi diagrams [15, 16] are based on bisecting curves enjoying simple combinatorial properties, rather than on the geometric notions of sites and circles. They serve as a unifying concept. Once the bisector system of any concrete type of Voronoi diagram is shown to fulfill the AVD properties, structural results and efficient algorithms become available without further effort. For example, the first optimal algorithms for constructing nearest Voronoi diagrams of disjoint convex objects, or of line segments under the Hausdorff metric, have been obtained this way [20].

In a concrete order-k Voronoi diagram, all points are placed into the same region that have the same k nearest neighbors among the given sites. This paper is the first to study *abstract Voronoi diagrams of arbitrary order* k. We prove that their complexity is upper bounded by $2k(n-k)$. So far, an $O(k(n-k))$ bound has been shown only for point sites in the Euclidean and L_p plane [18, 19], and, very recently, for line segments [23]. These proofs made extensive use of the geometry of the sites.

Our result on AVDs implies a $2k(n-k)$ upper bound for a wide range of cases for which only trivial upper complexity bounds were previously known, and a slightly sharper bound for the known cases.

Also, our proof shows that the reasons for this bound are combinatorial properties of certain permutation sequences.

Keywords: Abstract Voronoi diagrams, computational geometry, distance problems, higher order Voronoi diagrams, Voronoi diagrams.

1 Introduction

Voronoi diagrams are useful structures, known in many areas of science. The underlying idea goes back to Descartes [11]. There are sites p, q that exert influence on their surrounding space, M. Each point of M is assigned to that site p (resp. to those sites $p_1, \ldots, p_k$) for which the influence is strongest. Points assigned to the same site(s) form *Voronoi regions*.

The nature of the sites, the measure of influence, and space M can vary. The *order*, k, can range from 1 to $n-1$ if n sites are given. For $k = 1$ the standard

* This work was supported by the European Science Foundation (ESF) in the EUROCORES collaborative research project EuroGIGA/VORONOI.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 208–219, 2013.

nearest Voronoi diagram results, while for $k = n-1$ the *farthest* Voronoi diagram is obtained, where all points of M having the same farthest site are placed in the same Voronoi region. In this paper we are interested in values of k between 1 and $n-1$; here an *order-k* Voronoi region contains all points that have the same k nearest sites. See the surveys and monographs [5, 7, 8, 10, 12, 22].

A lot of attention has been given to nearest Voronoi diagrams in the plane. Many concrete cases have the following features in common. The locus of all points at identical distance to two sites p, q is an unbounded curve $J(p, q)$. It bisects the plane into two domains, $D(p, q)$ and $D(q, p)$; domain $D(p, q)$ consists of all points closer to p than to q. Intersecting all $D(p, q)$, where $q \neq p$ for a fixed p, results in the Voronoi region $\mathrm{VR}(p, S)$ of p with respect to site set S. It equals the set of all points with unique nearest neighbor p in S. If geodesics exist, Voronoi regions are pathwise connected, and the union of their closures covers the plane, since each point has at least one nearest neighbor in S.

In *abstract Voronoi diagrams* (AVDs, for short) no sites or distance measures are given. Instead, one takes unbounded curves $J(p, q) = J(q, p)$ as primary objects, together with the domains $D(p, q)$ and $D(q, p)$ they separate. *Nearest abstract Voronoi regions* are defined by

$$\mathrm{VR}(p, S) := \bigcap_{q \in S \setminus \{p\}} D(p, q),$$

and now one *requires* that the following properties hold true for each nonempty subset S' of S.

(A1) Each nearest Voronoi region $VR(p, S')$ is pathwise connected.
(A2) Each point of the plane belongs to the closure of a nearest Voronoi region $VR(p, S')$.

Two more, rather technical, assumptions on the curves $J(p, q)$ are stated in Definition 1 below. It has been shown that the resulting nearest AVDs—the plane minus all Voronoi regions— are planar graphs of complexity $O(n)$. They can be constructed, by randomized incremental construction, in $O(n \log n)$ many steps [16,17,20]. Moreover, properties (A1) and (A2) need only be checked for all subsets S' of size three [16]. This makes it easier to verify that a concrete Voronoi diagram is under the roof of the AVD concept. Examples of such applications can be found in [1–3, 9, 14, 20]. *Farthest abstract Voronoi diagrams* consist of regions $\mathrm{VR}^*(p, S) := \bigcap_{q \in S \setminus \{p\}} D(q, p)$. They have been shown to be trees of complexity $O(n)$, computable in expected $O(n \log n)$ many steps [21].

In this paper we consider, for the first time, general *order-k abstract Voronoi regions*, defined by

$$\mathrm{VR}^k(P, S) := \bigcap_{p \in P,\ q \in S \setminus P} D(p, q),$$

for each subset P of S of size k. The order-k abstract Voronoi diagram $V^k(S)$ is defined to be the complement of all order-k Voronoi regions in the plane; it equals

the collection of all edges that separate order-k Voronoi regions (Lemma 4). In addition to properties (A1) and (A2) we shall assume the following for each nonempty subset S' of S.

(A3) No nearest Voronoi region $VR(p, S')$ is empty.

In Lemma 1 we prove that property (A3) need only be tested for all subsets S' of size four. Clearly, (A3) holds in all concrete cases where each nearest region contains its site.

Figure 1 shows two concrete order-2 diagrams of points and line segments under the Euclidean metric. We observe that the order-2 Voronoi region of line segments s_1, s_2 is disconnected. In general, a Voronoi region in $V^2(S)$ can have $n-1$ connected components. Figure 2 depicts a curve system fulfilling all properties required, and the resulting abstract Voronoi diagrams of orders 1 to 4. An index p placed next to a curve indicates on which side the domain $D(p, q)$ lies, if q denotes the opposite index. The order-2 region of p_1 and p_2 consists of four connected components.

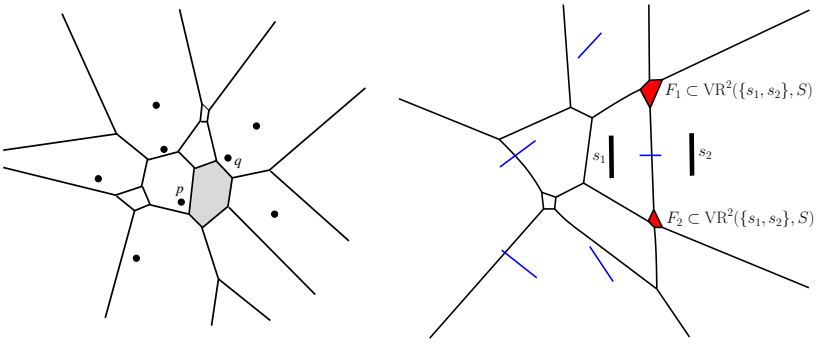


Fig. 1. Order-2 diagrams of points and line segments

In this paper we are proving the following result on the number of 2-dimensional faces of order-k abstract Voronoi diagrams.

Theorem 1. *The abstract order-k Voronoi diagram $V^k(S)$ has at most $2k(n-k)$ many faces.*

So far, an $O(k(n-k))$ bound was known only for points in L_2 and in the L_p-plane [18,19]. Quite recently, it has been shown for line segments in the Euclidean plane [23], too. The proofs of these results depend on geometric arguments using k-sets[1], k-nearest neighbor Delaunay triangulations, and point-line duality, respectively. None of these arguments applies to abstract Voronoi diagrams.

[1] We call a subset of size k of n points a *k-set* if it can be separated by a line passing through two other points. Such k-sets correspond to unbounded order-$(k+1)$ Voronoi edges.

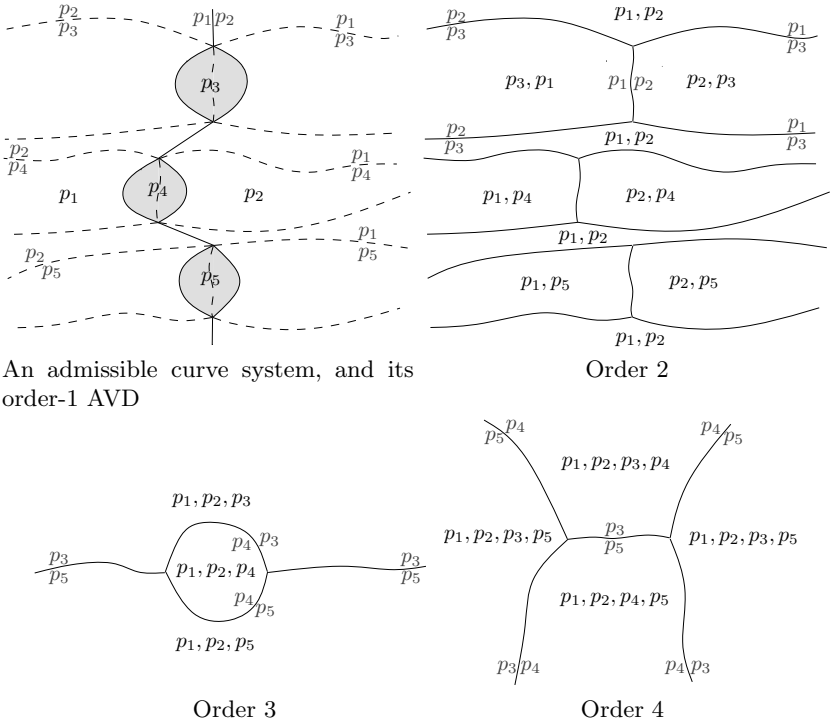


Fig. 2. AVD of 5 sites in all orders

However, the upper bound on k-sets established in [4] had a combinatorial proof; it was obtained by analyzing the cyclic permutation sequences that result when projecting n point sites onto a rotating line. In such a sequence, consecutive permutations differ by a switch of adjacent elements, and permutations at distance $\binom{n}{2}$ are inverse to each other.

In this paper we traverse the unbounded edges of higher order AVDs, and obtain a strictly larger class of cyclic permutation sequences, where consecutive permutations differ by switches and any two elements switch exactly twice. Our proof is based on a tight upper bound to the number of switches that can occur among the first $k+1$ elements; see Lemma 9. It is interesting to observe that in our class, each permutation sequence can be realised by an AVD (Lemma 10), while a similar statement does not hold for the sequences obtained by point projection [13].

To avoid technical complications we are assuming, in this paper, that any two input curves $J(p, q)$ intersect in a finite number of points, and that Voronoi vertices are of degree 3. How to get rid of the first assumption has been shown for the case of nearest AVDs in [16]. The perturbation technique of [15] can be used to obtain degree 3 vertices.

Theorem 1 implies a $2k(n-k)$ upper complexity bound on a wide range order-k Voronoi diagrams for which no good bounds were previously known. For example, sites may be disjoint convex objects of constant complexity in L_2 or under the Hausdorff metric. For point sites, distance can be measured by any metric d satisfying the following conditions: points in general position have unbounded bisector curves; d-circles are of constant algebraic complexity; each d-circle contains an L_2-circle and vice versa; for any two points $a \neq c$ there is a third point $b \neq a, c$ such that $\mu(a,c) = \mu(a,b) + \mu(b,c)$ holds. This includes all convex distance functions of constant complexity, but also the Karlsruhe metric where motions are constrained to radial or circular segments with respect to a fixed center point. A third example are point sites with additive weights a_p, a_q that satisfy $|a_p - a_q| < |p - q|$, for any two sites $p \neq q$; see [7] for a discussion of these examples.

The rest of this paper is organized as follows. In Section 2 we present some basic facts about AVDs. Then, in Section 3, permutation sequences will be studied, in order to establish an upper bound to the number of unbounded Voronoi edges of order at most k. This will lead, in Section 4, to a tight upper bound for the number of faces of order k.

2 Preliminaries

In this section we present some basic facts on abstract Voronoi diagrams of various orders.

Definition 1. *A curve system* $J := \{J(p,q) : p \neq q \in S\}$ *is called* admissible *if it fulfills, besides axioms (A1), (A2), (A3) stated in the introduction, the following axioms.*

(A4) Each curve $J(p,q)$, where $p \neq q$, is unbounded. After stereographic projection to the sphere, it can be completed to a closed Jordan curve through the north pole.

(A5) Any two curves $J(p,q)$ and $J(r,t)$ have only finitely many intersection points, and these intersections are transversal.

Fortunately, verification of these axioms can be based on constant size examples.

Lemma 1. *To verify axioms (A1) and (A2) it is sufficient to check all subsets S' of size 3, and for (A3), of size 4.*

Proofs for (A1) and (A2) can be found in [16, Section 4.3]. For (A3), the proof is given in the Appendix.

The following fact will be very useful in the sequel. Its proof can be found in [16, Lemma 5].

Lemma 2. *For all p, q, r in S, $D(p,q) \cap D(q,r) \subseteq D(p,r)$ holds.*

Consequently, for each $x \notin \bigcup_{p,q \in S} J(p,q)$ a global ordering of the site set S is given by

$$p <_x q :\Longleftrightarrow x \in D(p,q).$$

Informally, one can interpret $p <_x q$ as "x is closer to p than to q". We will write $p < q$ if it is clear which $x \in \mathbb{R}^2$ we are referring to.

As a direct consequence we show that property (A2) also holds for abstract order-k Voronoi regions.

Lemma 3. *Let $J = \{J(p,q) : p \neq q \in S\}$ be an admissible curve system. Then for each $k \in \{1, \ldots, n-1\}$*

$$\mathbb{R}^2 = \bigcup_{P \subseteq S, |P|=k} \overline{VR^k(P,S)}.$$

Proof. Let $x \in \mathbb{R}^2$. If x is not contained in any bisecting curve $J(p,q)$ then it belongs to the order-k region $\mathrm{VR}^k(P,S)$, where $P = \{p_1, \ldots, p_k\}$ are the k smallest elements of S with respect to the ordering $<_x$. Otherwise, x lies on the boundary of a domain $D \subset \mathbb{R}^2 \setminus \bigcup_{p \neq q \in S} J(p,q)$, and D fully belongs to an order-k region. □

The proofs of the following Lemmata 4 and 5 are similar to the proof of Lemma 3.

Lemma 4.

$$V^k(S) = \bigcup_{\substack{P \neq P' \subset S \\ |P|=|P'|=k}} \overline{VR^k(P,S)} \cap \overline{VR^k(P',S)}$$

Lemma 5. *If the intersection $E := \overline{VR^k(P,S)} \cap \overline{VR^k(P',S)}$ is not empty, there are sites $p \in P$ and $p' \in P'$ such that $P \setminus \{p\} = P' \setminus \{p'\}$, and $E \subseteq J(p,p')$ holds. For each point $x \in VR^k(P,S)$ near E, index p is the k-th with respect to $<_x$, while for points x' in $VR^k(P',S)$ index p' appears at position k.*

In particular, $D(p,p')$ lies on the same side of $J(p,p')$ as $\mathrm{VR}^k(P,S)$ does.

If F, F' are connected components (*faces*) of $\mathrm{VR}^k(P,S)$ and $\mathrm{VR}^k(P',S)$, respectively, the intersection $\overline{F} \cap \overline{F'}$ can be empty, or otherwise be of dimension 0 (*Voronoi vertices*) or 1 (*Voronoi edges*).

For the next lemma we assume that all vertices are of degree 3. As in concrete order-k Voronoi diagrams [18] there are two types of vertices that can be distinguished by the nature of sets $P_1, P_2, P_3 \subset S$ which define the adjacent order-k Voronoi regions. In the first case there exists a set $H \subset S$ of size $k-1$ and three more indices $p, q, r \in S$ satisfying

$$P_1 = H \cup \{p\}, \quad P_2 = H \cup \{q\}, \quad P_3 = H \cup \{r\};$$

a vertex where such regions $\mathrm{VR}^k(P_i,S)$ meet is called *new* in $V^k(S)$, or *of nearest type*. In the second case, there is a subset $K \subset S$ of size $k-2$ and three more sites $p, q, r \in S$ such that

$$P_1 = K \cup \{p,q\}, \quad P_2 = K \cup \{p,r\}, \quad P_3 = K \cup \{q,r\}.$$

A vertex adjacent to such regions is called *old* in $V^k(S)$, or *of furthest type*. The proof of the following lemma follows quite directly from these definitions.

Lemma 6. *Let v be a new vertex in $V^k(S)$. Then v is an old vertex of $V^{k+1}(S)$, and v lies in the interior of a face of $V^{k+2}(S)$, i. e., v is not a vertex of $V^{k+2}(S)$. Furthermore, every edge of $V^k(S)$ is included in a face of $V^{k+1}(S)$.*

Already in [21] it has been shown that farthest abstract Voronoi diagrams are trees, under a slightly different definition of admissible curves. In the Appendix we give a short alternative proof of this fact based on our axioms (A1)–(A5).

Lemma 7. *The farthest abstract Voronoi diagram $V^*(S)$ is a tree.*

3 Bounding the Number of Unbounded Edges of $V^{\leq k}(S)$

Let Γ be a circle in $\mathbb{R}^2$ large enough such that no pair of bisectors cross on or outside of Γ (axiom (A5)) and each bisector crosses Γ transversally exactly twice (axiom (A4)).
If we walk around Γ the ordering $<_x$ on S changes whenever we cross a bisector $J(p, q)$. Here indices p and q switch their places in the ordering. Because of the construction of Γ there can be only one switch at a time and each pair of sites switches exactly two times while walking one round around Γ, resulting in $n(n-1)$ switches altogether.

Lemma 8. *Suppose that two sites p and q switch in the ordering. Then they are adjacent to each other just before and after the switch.*

Proof. Let $p_1 < \ldots < p_n$, and assume that we cross $J(p_i, p_j)$, $i < j$, which means that p_i and p_j switch their places in the ordering. Suppose $j > i + 1$; then $x \in D(p_{i+1}, p_j)$ before the switch and $x \in D(p_j, p_{i+1})$ after the switch, but $J(p_{i+1}, p_j)$ has not been crossed—a contradiction. □

Every time a switch among the first $k+1$ elements of the ordering occurs, there is an unbounded edge of a Voronoi diagram of order $\leq k$. This means that *the maximum number of unbounded edges of all diagrams of order $\leq k$ is equal to the maximum number of switches among the first $k+1$ elements in the ordering.*

Permutation sequences and estimates for the maximum number of switches among the first k elements have been used in [4] to bound the number of k-sets of n points in the plane. These sequences resulted from projecting n points in general position onto a rotating line. Hence, they were of length $2N$, where $N = \binom{n}{2}$, and they had the following properties. Adjacent permutations differ by a transposition of adjacent elements, and any two permutations a distance N apart are inverse to each other. It has been shown in [13] that not every permutation of this type can be realized by a point set.

In the following lemma we introduce a larger class of permutation sequences that fits the AVD framework.

Lemma 9. *Let $P(S)$ be a cyclic sequence of permutations $P_0, \ldots, P_N = P_0$ such that*

(i) P_{i+1} differs from P_i by an adjacent switch;
(ii) each pair of sites $p, q \in S$ switches exactly two times in $P(S)$.

Then the number of switches occuring in $P(S)$ among the first $k+1$ sites is upper bounded by $k(2n-k-1)$. Furthermore, this bound is tight.

Proof. Call a switch *good* if it involves at least one of the k first sites of a permutation; otherwise call it *bad*. Consider the permutation $P_0 = (p_1, p_2, \ldots, p_n)$. For $i \in \{k+2, \ldots, n\}$, define B_i as the set of bad switches where p_i is switching with a site in $\{p_1, \ldots, p_{i-1}\}$. We remark that the sets B_i, for $i \in \{k+2, \ldots, n\}$, are pairwise disjoint. If p_i is not involved in a good switch, then all its $2i-2$ switches with sites in $\{p_1, \ldots, p_{i-1}\}$ are bad. Otherwise, for p_i to be involved in a good switch, it must first be involved in at least $i-k-1$ bad switches with sites in $\{p_1, \ldots, p_{i-1}\}$, in order to reach the first $k+1$ positions of a permutation, and since $P_0 = P_N$, it has to be involved in as many bad switches in order to return to its original place in P_N. In both cases, $|B_i| \geq 2(i-k-1)$. Because of (ii), the total number of switches is $N = 2\binom{n}{2}$. Therefore the number of good switches is at most

$$2\binom{n}{2} - \sum_{i=k+2}^{n} |B_i| \leq 2\binom{n}{2} - 2\sum_{j=1}^{n-k-1} j = k(2n-k-1),$$

where $j = i-k-1$.

To show that the bound is tight, let $P_0 = (p_1, \ldots, p_n)$. We will switch each p_i with all p_j having a place before p_i in P_0 in consecutive order until p_i is the first element and then in inverse order back. Start with $i = 2$ and continue until $i = n$. Then the number of switches among the first $k+1$ sites is exactly $2\binom{n}{2} - 2\sum_{j=1}^{n-k-1} j$. □

In contradistinction to the result in [13], each such permutation sequence can be realized by an AVD. The following Lemma 10 will be used for proving that the upper bound shown in Lemma 11 is tight. The proof of Lemma 10 is given in the Appendix.

Lemma 10. *Let $P(S)$ be a sequence of permutations as in Lemma 9. Then there exists an abstract Voronoi diagram where the ordering of the sites along Γ changes according to $P(S)$.*

Let S_i be the number of unbounded edges in $V^i(S)$. If an edge e has got two unbounded endpieces, i. e. the edge e bounding a p- and q-region is the whole bisector $J(p, q)$, then e is counted twice as an unbounded edge.

Lemma 11. *Let $k \in \{1, \ldots, n-1\}$. Then,*

$$k(k+1) \leq \sum_{i=1}^{k} S_i \leq k(2n-k-1).$$

Both bounds can be attained.

Proof. The second bound follows directly from Lemma 9. The first bound follows from the fact that the minimum number of switches among the first $(k+1)$ sites is greater or equal to the total number of switches, $n(n-1)$, minus the maximum number of switches among the last $(n-k)$ sites, which again is equal to the maximum number of switches among the first $(n-k)$ sites. Using Lemma 9 this implies

$$\sum_{i=1}^{k} S_i \geq n(n-1) - (n-k-1)(2n-(n-k-1)-1) = k(k+1).$$

The tightness of the bounds follows from Lemma 10. □

4 Bounding the Number of Faces of $V^k(S)$

In the following we assume that each Voronoi vertex is of degree 3. The following two lemmata give combinatorial proofs for facts that were previously shown by geometric arguments [18, 23].

Lemma 12. *Let H be a subset of S of size $k+1$ and F a face of $VR^{k+1}(H,S)$. The portion of $V^k(S)$ enclosed in F is exactly the farthest Voronoi diagram $V^*(H)$ intersected with F.*

Proof. "⇒": Let $x \in F$ and suppose $x \in \mathrm{VR}^k(H', S)$ for $H' \subset S$ of size k. Since $F \subseteq \mathrm{VR}^{k+1}(H,S)$ it follows that $x \in D(p,q)$ for all $p \in H$ and $q \in S \setminus H$, implying $H' \subset H$. Let $H \setminus H' = \{r\}$, then $x \in D(p,r)$ for all $p \in H'$ and hence $x \in \mathrm{VR}^*(r,H)$.
"⇐": Let $x \in F$ and $x \in \mathrm{VR}^*(r,H)$. Then $x \in D(p,q)$ for all $p \in H$ and $q \in S \setminus H$ and $x \in D(p,r)$ for all $p \in H \setminus \{r\}$. This implies $x \in \mathrm{VR}^k(H \setminus \{r\}, S)$. □

Lemma 13. *Let F be a face of $VR^k(H,S)$, $H \subseteq S$, $|H| = k \geq 2$. Then $V^*(H) \cap F$ is a nonempty tree.*

Proof. First we show that $V^*(H) \cap F$ is not empty by assuming the opposite. Then there is a $p \in H$ such that $F \subseteq \mathrm{VR}^*(p,H)$. Let $F' \subseteq \mathrm{VR}^k(H',S)$ be a face of $V^k(S)$ adjacent to F along an edge e. By Lemma 5, we have $H = U \cup \{q\}$ and $H' = U \cup \{q'\}$, where q, q' are different and not contained in U. Also, $e \subseteq J(q,q')$ holds. If p were in U, we would obtain $F' \subseteq D(p,q)$ and $F \subseteq V^*(p,H) \subseteq D(q,p)$, hence $e \subseteq J(p,q)$—a contradiction to axiom (A5). Thus, $p \notin U$, which means $p = q$. Now Lemma 5 implies that each edge on the boundary of F has to be part of a curve $J(p,q_j)$ such that $D(p,q_j)$ lies on the F-side. Let $q_1, \ldots, q_i$ be the sites for which there is such an edge e on the boundary of F. Then $\mathrm{VR}^1(p, \{p, q_1, \ldots, q_i\}) = F$, because nearest Voronoi regions are connected thanks to axiom (A1). But from $F \subseteq V^*(p,H)$ it follows that $\mathrm{VR}^1(p,H) \subseteq \mathbb{R}^2 \setminus F$ and hence $\mathrm{VR}^1(p,S) \subseteq F \cap \mathbb{R}^2 \setminus F = \emptyset$, a contradiction to axiom (A3).

Next we show that $V^*(H) \cap F$ is a tree. Because of Lemma 7 it is clear that it is a forest. So it remains to prove that it is connected. Otherwise, there would

be a domain $D \subset F$, bounded by two paths $P_1, P_2 \subset F$ of $V^*(H)$ and two disconnected parts e_1 and e_2 on the boundary of F. There is an index $p \in H$ such that $D \subseteq \mathrm{VR}^*(p, H)$. Since $V^*(H)$ is a tree, by Lemma 7, the upper (or: the lower) two endpoints of P_1 and P_2 must be connected by a path P in $V^*(H)$ that belongs to the boundary of $\mathrm{VR}^*(p, H)$; see Figure 3. Here path P connects the endpoints of e_1; both curves together encircle a domain D', which is part of $\mathrm{VR}^*(p, H)$. By definition of the farthest Voronoi diagram, there are $q_1, \ldots, q_i$, such that $e_1 \cup P$ is part of $J(p, q_1), \ldots, J(p, q_i)$, and all $D(p, q_j)$ are situated outside of D'; compare Lemma 5. But then $\mathrm{VR}^*(p, \{p, q_1, \ldots, q_i\})$ would be bounded, a contradiction to Lemma 7. □

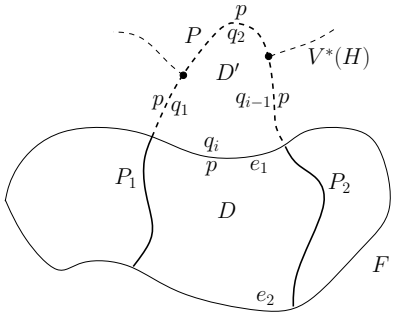


Fig. 3. The intersection of an order-k face F and the farthest Voronoi diagram of its defining sites must be a tree

Lemma 14. *Let F be a face of $VR^{k+1}(H, S)$ and m the number of Voronoi vertices of $V^k(S)$ enclosed in its interior. Then F encloses $2m + 1$ Voronoi edges of $V^k(S)$.*

Proof. See Lemmata 12 and 13. □

The next two lemmata are from [23]. The proofs are given in the Appendix, for completeness.

Lemma 15. *Let F_k, E_k, V_k and S_k denote, respectively, the number of faces, edges, vertices, and unbounded edges in $V^k(S)$. Then,*

$$E_k = 3(F_k - 1) - S_k \tag{1}$$

$$V_k = 2(F_k - 1) - S_k. \tag{2}$$

Lemma 16. *The number of faces in an AVD of order k is*

$$F_k = 2kn - k^2 - n + 1 - \sum_{i=1}^{k-1} S_i.$$

Theorem 2. *The number of faces F_k in an AVD of order k is bounded as follows*

$$n - k + 1 \leq F_k \leq 2k(n-k) + k + 1 - n \in O(k(n-k)).$$

Both bounds can be attained.

Proof. Lemma 11 implies tight bounds $k(k-1) \leq \sum_{i=1}^{k-1} S_i \leq (k-1)(2n-k)$. Together with Lemma 16 this proves the theorem. □

5 Concluding Remarks

A natural question is if weaker axioms than (A1)–(A5) can still imply Theorem 2. In the case of nearest abstract Voronoi diagrams, it could be shown, with some technical effort, that (A5) is dispensable [16]. A big challenge will be to design an efficient algorithm for constructing abstract Voronoi diagrams of order k. Even for the special case of points in the Euclidean metric, no optimal algorithm is known for computing a single higher order Voronoi diagram.

References

1. Abellanas, M., Hurtado, F., Palop, B.: Transportation Networks and Voronoi Diagrams. In: Proceedings International Symposium on Voronoi Diagrams in Science and Engineering (2004)
2. Ahn, H.-K., Cheong, O., van Oostrum, R.: Casting a Polyhedron with Directional Uncertainty. Computational Geometry: Theory and Applications 26(2), 129–141 (2003)
3. Aichholzer, O., Aurenhammer, F., Palop, B.: Quickest Paths, Straight Skeletons, and the City Voronoi Diagram. Discrete and Computational Geometry 31(7), 17–35 (2004)
4. Alon, N., Györi, E.: The number of Small Semispaces of a Finite Set of Points in the Plane. Journal of Combinatorial Theory, Ser. A 41(1), 154–157 (1986)
5. Aurenhammer, F.: Voronoi Diagrams: A Survey of a Fundamental Geometric Data Structure. ACM Computing Surveys 23(3), 345–405 (1991)
6. Aurenhammer, F., Drysdale, R., Krasser, H.: Farthest Line Segment Voronoi Diagrams. Information Processing Letters 100(6), 220–225 (2006)
7. Aurenhammer, F., Klein, R.: Voronoi Diagrams. In: Sack, J.R., Urrutia, G. (eds.) Handbook on Computational Geometry, pp. 201–290. Elsevier (1999)
8. Aurenhammer, F., Klein, R., Lee, D.-T.: Voronoi Diagrams and Delaunay Triangulations. World Scientific Publishing Company (to appear, 2013)
9. Bae, S.W., Chwa, K.-Y.: Voronoi Diagrams for a Transportation Network on the Euclidean Plane. International Journal on Computational Geometry and Applications 16, 117–144 (2006)
10. Boissonnat, J.D., Wormser, C., Yvinec, M.: Curved Voronoi Diagrams. In: Boissonnat, J.D., Teillaud, M. (eds.) Effective Computational Geometry for Curves and Surfaces. Springer, Mathematics and Visualization (2006)
11. Descartes, R.: Principia Philosophiae. Ludovicus Elzevirius, Amsterdam, 1644
12. Fortune, S.: Voronoi Diagrams and Delaunay Triangulations. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, ch. 20, pp. 377–388. CRC Press LLC (1997)

13. Goodman, J.E., Pollack, R.: On the Combinatorial Classification of Non-Degenerate Configurations in the Plane. Journal of Combinatorial Theory, Ser. A 29, 220–235 (1980)
14. Karavelas, M.I., Yvinec, M.: The Voronoi Diagram of Planar Convex Objects. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 337–348. Springer, Heidelberg (2003)
15. Klein, R.: Concrete and Abstract Voronoi Diagrams. LNCS, vol. 400. Springer, Heidelberg (1989)
16. Klein, R., Langetepe, E., Nilforoushan, Z.: Abstract Voronoi Diagrams Revisited. Computational Geometry: Theory and Applications 42(9), 885–902 (2009)
17. Klein, R., Mehlhorn, K., Meiser, S.: Randomized Incremental Construction of Abstract Voronoi Diagrams. Computational Geometry: Theory and Applications 3, 157–184 (1993)
18. Lee, D.-T.: On k-Nearest Neighbor Voronoi Diagrams in the Plane. IEEE Trans. Computers 31(6), 478–487 (1982)
19. Liu, C.-H., Papadopoulou, E., Lee, D.T.: An Output-Sensitive Approach for the L_1/L_∞ k-Nearest-Neighbor Voronoi Diagram. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 70–81. Springer, Heidelberg (2011)
20. Mehlhorn, K., Meiser, S., Ó'Dúnlaing, C.: On the Construction of Abstract Voronoi Diagrams. Discrete and Computational Geometry 6, 211–224 (1991)
21. Mehlhorn, K., Meiser, S., Rasch, R.: Furthest Site Abstract Voronoi Diagrams. International Journal of Computational Geometry and Applications 11(6), 583–616 (2001)
22. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley Series in Probability and Statistics (2000)
23. Papadopoulou, E., Zavershynskyi, M.: On Higher Order Voronoi Diagrams of Line Segments. In: Chao, K.-M., Hsu, T.-s., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 177–186. Springer, Heidelberg (2012)

A Pseudo-Polynomial Algorithm for Mean Payoff Stochastic Games with Perfect Information and a Few Random Positions

Endre Boros[1], Khaled Elbassioni[2], Vladimir Gurvich[1], and Kazuhisa Makino[3]

[1] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003
{boros,gurvich}@rutcor.rutgers.edu
[2] Masdar Institute of Science and Technology, Abu Dhabi, UAE
kelbassioni@masdar.ac.ae
[3] Research Institute for Mathematical Sciences (RIMS) Kyoto University, Kyoto 606-8502, Japan
makino@kurims.kyoto-u.ac.jp

Abstract. We consider two-person zero-sum stochastic mean payoff games with perfect information, or BWR-games, given by a digraph $G = (V, E)$, with local rewards $r : E \to \mathbb{R}$, and three types of vertices: black V_B, white V_W, and random V_R forming a partition of V. It is a long-standing open question whether a polynomial time algorithm for BWR-games exists, or not. In fact, a pseudo-polynomial algorithm for these games would already imply their polynomial solvability. In this paper, we show that BWR-games with a constant number of random nodes can be solved in pseudo-polynomial time. That is, for any such game with a few random nodes $|V_R| = O(1)$, a saddle point in pure stationary strategies can be found in time polynomial in $|V_W| + |V_B|$, the maximum absolute local reward R, and the common denominator of the transition probabilities.

1 Introduction

We consider two-person zero-sum stochastic games with perfect information and mean payoff: Let $G = (V, E)$ be a digraph whose vertex-set V is partitioned into three subsets $V = V_B \cup V_W \cup V_R$ that correspond to black, white, and random positions, controlled respectively, by two players, *Black* - the *minimizer* and *White* - the *maximizer*, and by nature. We also fix a *local reward* function $r : E \to \mathbb{Z}$, and probabilities $p(v, u)$ for all arcs (v, u) going out of $v \in V_R$. Vertices $v \in V$ and arcs $e \in E$ are called *positions* and *moves*, respectively. In a position $v \in V_W$ or $v \in V_B$ the corresponding player White or Black selects an arc (v, u), while in a random position $v \in V_R$ a move (v, u) is chosen with the given probability $p(v, u)$. From a given initial position $v_0 \in V$ the game produces an infinite walk (called a *play*). White's objective is to maximize the *limiting mean payoff*

$$c = \liminf_{n \to \infty} \frac{\sum_{i=0}^{n} b_i}{n+1}, \tag{1}$$

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 220–231, 2013.

where b_i is the expected reward incurred at step i of the play, while the objective of BLACK is the opposite, that is, to minimize c. For this model it was shown in [17, 28] (see also [9]) that a *saddle point* exists in *pure positional uniformly optimal* strategies. Here "pure" means that the choice of a move (v, u) in a personal position $v \in V_B \cup V_W$ is deterministic; "positional" means that this choice depends solely on v, not on previous positions or moves; finally, "uniformly optimal" means that the optimal strategies do not depend on the initial position v_0, either.

This class of *BWR-games* was introduced[1] in [19]; see also [10]. It was recently shown in [9] that the BWR-games and classical Gillette games [17] with perfect information are polynomially equivalent. The special case when $V_R = \emptyset$, so called *BW-games*, is also known as *cyclic*, or *mean payoff*. They were introduced for the complete bipartite digraphs in [31], for all (not necessarily complete) bipartite digraphs in [16], and for arbitrary digraphs[2] in [19]. A more special case was considered extensively in the literature under the name of *parity games* [2, 3, 11, 21, 23, 25], and later generalized also to include random nodes in [10]. A BWR-game is reduced to a *minimum mean cycle problem* in case $V_W = V_R = \emptyset$, see, e.g., [26]. If one of the sets V_B or V_W is empty, we obtain a *Markov decision process* (MDP), which can be expressed as a linear program; see, e.g., [30]. Finally, if both are empty $V_B = V_W = \emptyset$, we get a *weighted Markov chain*. In the special case when all rewards are zero except at special nodes called terminals, each of which only has is a single outgoing arc forming a self-loop, we get a *stochastic terminal payoff* game, and when the self-loops have 0/1 payoffs, and every random node has only two outgoing arcs with probability 1/2 each, we obtain the so-called *simple stochastic games* (SSG's), introduced by Condon [13, 14] and considered in several papers [18, 21]. In the latter games, the objective of WHITE is to maximize the probability of reaching the terminal, while BLACK wants to minimize this probability. Recently, it was shown that Gillette games with perfect information (and hence BWR-games by [9]) are equivalent to SSG's under polynomial-time reductions [1]. Thus, by recent results of Björklund, Vorobyov [4], and Halman [21], all these games can be solved in randomized strongly subexponential time $2^{O(\sqrt{n_d \log n_d})}$, where $n_d = |V_B| + |V_W|$ is the number of *deterministic* vertices. For BW-games many pseudo-polynomial and subexponential algorithms are known [2, 3, 5, 19–21, 27, 32, 34, 35]; see also [25] for parity games. Besides their many applications (see e.g., [24, 29]), all these games are of interest to Complexity Theory: Karzanov and Lebedev [27] (see also [35]) proved that the decision problem "whether the value of a BW-game is positive" is in the intersection of NP and co-NP. Yet, no polynomial algorithm is known for these games, see e.g., the recent survey by Vorobyov [34]. A similar complexity claim can be shown to hold for SSG's and BWR-games, see [1, 9].

[1] The results in [17, 28] were for a slightly different but (polynomially) equivalent formulation.

[2] In fact, a BW-game on an arbitrary digraph can be reduced to a BW-game on a bipartite graph; see [9].

1.1 Main Result

While there are numerous pseudo-polynomial algorithms known for the BW-case, it is a challenging open question whether a pseudo-polynomial algorithm exists for BWR-games. In fact, the problem of developing an efficient algorithm for stochastic games with perfect information was mentioned as an open problem already in the survey [33]. Our main result can be viewed as a partial solution of this problem for the case when the number of random nodes is fixed.

For a BWR-game $\mathcal{G}$ let us denote by $n = |V_W| + |V_B| + |V_R|$ the number of nodes, by $k = |V_R|$ the number of random nodes, and assume that all local rewards are integral with maximum absolute value R, and all transition probabilities are rational with common denominator D.

Theorem 1. *A BWR-game $\mathcal{G}$ can be solved in $(nDk)^{O(k)}R \cdot \operatorname{polylog} R$ time.*

We remark that our proof of Theorem 1 actually gives the following stronger bound. Let $\mathcal{G}'$ be a sub-game of $\mathcal{G}$, and $\mathcal{G}'(R)$ be a game obtained from $\mathcal{G}'$ by contracting all random nodes into a single deterministic node with a self-loop with reward R. Define $\nu(\mathcal{G}'(R))$ to be the number of distinct optimal values achieved by nodes in $\mathcal{G}'(R)$, and $\nu = \nu(\mathcal{G})$ be the maximum of $\nu(\mathcal{G}'(R))$ over all sub-games $\mathcal{G}'$ of $\mathcal{G}$. Then, the algorithm of Theorem 1 solves any BWR-game in time $(\nu Dk)^{O(k)}R \cdot \operatorname{poly}(n, \log R)$. For a BWR-game $\mathcal{G}$, it is immediate that $\nu(\mathcal{G}) \leq n$, while for a stochastic terminal payoff game $\mathcal{G}$ with t terminals of distinct rewards, we can show (Lemma 8) that $\nu \leq t + 1$, implying a running time of $(tDk)^{O(k)}R \cdot \operatorname{poly}(n, \log R)$ for these games, and of $k^{O(k)} \operatorname{poly}(n)$ for simple stochastic games. Thus, Theorem 1 extends the results in [12, 15, 18, 22] and unifies them with the pseudo-polynomial algorithms for deterministic mean payoff games [19, 32, 35].

We observe that any improvement of this result to a polynomial dependence on k would yield a polynomial-time algorithm for solving BWR-games in general. Indeed, by the results of Andersson and Miltersen [1], solving any BWR-game can be reduced to solving the β-discounted version of the same game with discount factor $\beta = 1 - [(n!)^2 2^{2n+3} D^{2n^2} R]^{-1}$. Furthermore, it is also shown in [1] that solving a $(1 - \frac{1}{B})$-discounted BWR-game with maximum reward R, common denominator of transition probabilities D, and integral B, can be reduced to solving a stochastic terminal payoff game with 0/1 local rewards and rational transition probabilities with common denominator BD, adding only two additional terminal nodes. Finally, Zwick and Paterson [35] showed that any such game can be reduced to a simple stochastic game at the cost of increasing the number of random nodes by $\log(BD) = O(n(\log n + n \log D) + \log R)$. Thus, a pseudo-polynomial algorithm, that depends polynomially on k would solve the original BWR-game in (weakly) polynomial time. Similarly, an improvement to a poly-logarithmic dependence on R would imply that BW-games can be solved in polynomial time.

It was shown in [6] that a pseudo-polynomial algorithm for BWR-games with constant number of random nodes can be used to obtain a polynomial-time approximation scheme for approximating the optimal values. Thus, we obtain the following result as a corollary of Theorem 1.

Corollary 1. *For any $\varepsilon > 0$, there is an algorithm that returns, for any given BWR-game with non-negative integral rewards, and rational transition probabilities with minimum value $p_{\min}$, a pair of strategies that approximates the value from a given initial position within a* relative *error of $1 + \varepsilon$. The running-time of the algorithm is bounded by* $\mathrm{poly}(n, 1/p_{\min}, \log R, 1/\varepsilon)$.

1.2 Main Idea of the Proof

Our approach relies heavily on the fact proved in [7] that *ergodic* BWR-games, i.e., those in which the (optimal) value does not depend on initial position, can be solved in $(Dk)^{O(k)} R \cdot \mathrm{poly}(n, \log R)$ time. For a non-ergodic game, this algorithm can be used to find the classes of nodes with the largest and smallest values, but cannot find the other classes since they interact via the random nodes. We follow the idea in [18] in guessing the order of values of the random nodes; however, unlike in the case for SSG's, the resulting game after fixing this order is still very complicated, and in fact, even if we know the actual values of all the nodes, it is not clear how to find optimal strategies realizing these values. Nevertheless, we show that if the values are known, then the problem can be reduced to solving the ergodic case. Then to utilize the algorithm of [7], we have to overcome the following main challenge: while the accuracy needed for solving an ergodic game can be shown to be $\frac{1}{D^{O(k)}}$, this is not true in the general case. In fact, there is an example with $k = 1$, in which the accuracy needed is exponentially small in n; see [8]. To resolve this issue, we employ the idea of *parametric search* to reduce the search space for all values into a set of products of intervals on the real line of cardinality at most $\nu(\mathcal{G})^k$. Using such a set of intervals we can iteratively replace random nodes by self-loops, on which the local reward is a guessed value selected in a binary search manner. In more details:

1. We iterate steps 2, 4 and 5 below over the random nodes in the guessed order, keeping only the nodes with highest rank (and hence having the same optimal value), and deleting all the other random nodes. We iterate until no random nodes remain, in which case we solve a BW-game.
2. We consider the situation when all the kept random nodes are replaced by a self-loop with local reward parameter x; we show (Corollary 2) that the value of any node in the resulting game defines an interval on the real line, as x changes from $-\infty$ to $+\infty$.
3. We extend the result in [7] for discovering ergodicity to the case when there is a single self-loop with reward x (Theorem 2 below). We show that the running time for such a procedure *does not depend* on x.
4. We use binary search combined with our aforementioned extension of the result in [7] (applied to a parametrized BW-game) to identify a set of at most $\nu(\mathcal{G})$ intervals, in each of which, the dependence of values of different nodes on x does not vary.
5. Since we do not know the real value of x, we guess among the identified intervals one that contains x; for the guessed interval, we show that a non-empty part of the game, namely the nodes that have values above the lower

bound of the interval, can be solved by a call to a parametrized BWR-game using again our extended procedure above.

6. The number of guesses is bounded by $(k\nu(\mathcal{G}))^k$; eventually, we make a correct guess which yields a pair of strategies that can be verified for optimality by solving two MDP's.

2 Preliminaries

2.1 BWR-Games

A BWR-game $\mathcal{G} = (G, p, r)$ is given by a digraph $G = (V, E)$, where $V = V_W \cup V_B \cup V_R$ is a partition of the vertices, that may have loops and multiple arcs, but no terminal nodes[3], i.e., nodes of out-degree 0; a set of probability distributions $p : E \cap (V_R \times V) \to [0, 1]$ specifying the probability $p(v, u)$ of a move from $v \in V_R$ to u; and a local reward function $r : E \to \mathbb{R}$. We assume that $\sum_{u \mid (v,u)\in E} p(v, u) = 1$ for all $v \in V_R$; for convenience we will also assume that $p(v, u) > 0$ whenever $(v, u) \in E$ and $v \in V_R$, and set $p(v, u) = 0$ for $(v, u) \notin E$.

Standardly, we define a strategy $s_W \in S_W$ (respectively, $s_B \in S_B$) as a mapping that assigns a move $(v, u) \in E$ to each position $v \in V_W$ (respectively, $v \in V_B$). A pair of strategies $s = (s_W, s_B)$ is called a *situation*. Given a BWR-game $\mathcal{G} = (G, p, r)$ and situation $s = (s_B, s_W)$, we obtain a weighted Markov chain $\mathcal{G}_s = (P_s, r)$ with transition matrix P_s in the obvious way:

$$p_s(v, u) = \begin{cases} 1 & \text{if } (v \in V_W \text{ and } u = s_W(v)) \text{ or } (v \in V_B \text{ and } u = s_B(v)); \\ 0 & \text{if } (v \in V_W \text{ and } u \neq s_W(v)) \text{ or } (v \in V_B \text{ and } u \neq s_B(v)); \\ p(v, u) & \text{if } v \in V_R. \end{cases}$$

In the obtained Markov chain $\mathcal{G}_s = (P_s, r)$, we define the limiting (mean) effective payoff $\mu_{\mathcal{G}_s}(v) = \mu_s(v)$ as

$$\mu_s(v) = \sum_{w \in V} p^*_s(v, w) \sum_u p_s(w, u) r(w, u), \tag{2}$$

where $p^*_s(v, w)$ is the limit probability in $\mathcal{G}_s$ to be at node w when the initial node is v. Doing this for all possible strategies of Black and White, we obtain a matrix game $M(v) : S_W \times S_B \to \mathbb{R}$, with entries $\mu_{(s_W,s_B)}(v)$ defined by (2). It is known that every such game has a saddle point in pure strategies [17, 28]. Moreover, there are optimal strategies (s^*_W, s^*_B) that do not depend on the starting position v, so-called uniformly optimal strategies. In contrast, the value of the game $\mu_{\mathcal{G}}(v) = M_{(s^*_W,s^*_B)}(v)$ may depend on v.

2.2 Basic Lemmas

We will use the following notation throughout. For $x \in \mathbb{R}$ and a set $Y \subseteq V$ such that every $v \in V_W \cup V_B$ has some arc (v, u) with $u \in Y$, we denote by $\mathcal{G}[Y](x)$ the

[3] This assumption is without loss of generality since otherwise one can add a loop to each terminal node.

sub-game obtained from $\mathcal{G}$ in the following way. We delete all nodes in $V \setminus Y$ and all arcs (v,u) where $v \notin Y$ or $u \notin Y$. For $v \in V_R \cap Y$ such that $\sum_{u \notin Y} p(v,u) > 0$, we add a new deterministic node $u(v)$ with a self-loop $(u(v), u(v))$ with local reward x, and an arc from v to $u(v)$ with local reward 0 and transition probability $p(v, u(v)) := \sum_{u \notin Y} p(v,u)$; all other arcs and nodes remain the same. When $\mathcal{G}$ is a BW-game, then $\mathcal{G}[Y]$ simply means the restriction of $\mathcal{G}$ on Y. Note that $\mathcal{G}[Y]$ may have terminals in general. In the sequel we consider only subsets Y for which $\mathcal{G}[Y]$ has no terminals. For a situation $s = (s_W, s_B)$ such that $s(v) \in Y$ for all $v \in Y$, we denote by the $s[Y] := (s_W[Y], s_B[Y])$ the restriction of s on Y. For brevity in what follows, we will call (v,u) a black, white, or random arc, depending on whether $v \in V_B$, $v \in V_W$, or $v \in V_R$, respectively.

Lemma 1. *(i) Given a BWR-game $\mathcal{G} = (G = (V = V_B \cup V_W \cup V_R, E), P, r)$, let $\widehat{\mathcal{G}}$ be the BW-game obtained from $\mathcal{G}$ by replacing each random node $v \in V_R$ with a terminal deterministic node (black or white, arbitrarily) with a local reward $\mu_{\mathcal{G}}(v)$ on the self-loop (v,v). Then $\mu_{\widehat{\mathcal{G}}}(v) = \mu_{\mathcal{G}}(v)$ for all $v \in V$.*

(ii) Let $\widehat{\mathcal{G}}$ be as above and $U \subseteq V$ be such that $\mu_{\mathcal{G}}(v) \neq \mu_{\mathcal{G}}(u)$ for all $v \in U$ and $u \in V \setminus U$. Then $\mu_{\widehat{\mathcal{G}}[U]}(v) = \mu_{\mathcal{G}}(v)$ for all $v \in U$.

Let $\mathcal{G} = (G = (V = V_B \cup V_W \cup V_R, E), p, r)$ be a BWR-game. In what follows we will use the following notation. For a $\theta \in \mathbb{R}$, let $S(\theta) := \{v \in V \mid \mu_{\mathcal{G}}(v) = \theta\}$. If $S(\theta) \neq \emptyset$, we refer to it as an *ergodic class*. If $S(\theta) = V$ the game $\mathcal{G}$ is said to be ergodic. Let $\theta_1 < \theta_2 < \cdots < \theta_\ell$ be such that $S(\theta_i) \neq \emptyset$ for all $i \in [\ell]$ and $\bigcup_{i=1}^{\ell} S(\theta_i) = V$.

Ergodic classes necessarily satisfy the following properties.

Proposition 1. *(i) There exists no arc $(v,u) \in E$ such that $v \in V_W \cap S(\theta_i)$, $u \in S(\theta_j)$, and $j > i$;*

(ii) there exists no arc $(v,u) \in E$ such that $v \in V_B \cap S(\theta_i)$, $u \in S(\theta_j)$, and $j < i$;

(iii) for every $v \in V_W \cap S(\theta_i)$, there exists an arc $(v,u) \in E$ such that $u \in S(\theta_i)$;

(iv) for every $v \in V_B \cap S(\theta_i)$, there exists an arc $(v,u) \in E$ such that $u \in S(\theta_i)$;

(v) there exists no arc $(v,u) \in E$ such that $v \in V_R \cap S(\theta_1)$, and $u \notin S(\theta_1)$;

(vi) there exists no arc $(v,u) \in E$ such that $v \in V_R \cap S(\theta_\ell)$, and $u \notin S(\theta_\ell)$.

For $i \in [\ell]$, define $\mathcal{G}[\theta_i]$ to be the game $\mathcal{G}[S(\theta_i)](\theta_i)$. Proposition 1 guarantees that the game $\mathcal{G}[\theta_i]$ is well-defined, that is, for every $v \in S(\theta_i)$ there is at least one arc going out of v in $\mathcal{G}[\theta_i]$.

Lemma 2. *For all $i \in [\ell]$ and $v \in S(\theta_i)$, it holds that $\mu_{\mathcal{G}[\theta_i]}(v) = \theta_i$.*

Lemma 3. *For $i \in [\ell]$, let $s^i := (s^i_W, s^i_B)$ be a pair of optimal strategies in $\mathcal{G}[\theta_i]$. Then the situation $s^* = (s^*_W, s^*_B)$ obtained by concatenating all these strategies together (that is, $s^*_W(v) := s^i_W(v)$ for $v \in V_W \cap S(\theta_i)$ and $s^*_B(v) := s^i_B(v)$ for $v \in V_B \cap S(\theta_i)$) is optimal in $\mathcal{G}$.*

Remark 1. Lemma 3 states that, if we the know values of all the nodes, then we can get uniformly optimal strategies by solving ℓ ergodic different games. It should be noted however that, if we use such reduction directly, then a pseudo-polynomial algorithm for the ergodic case, as the one described in [7], does not yield in general a pseudo-polynomial algorithm for solving the whole game. The reason is that in our reduction in the proof of Lemma 3, we introduce local rewards on self-loops (v, v) of value $\mu_{\mathcal{G}}(v)$, which might be exponentially small in n, even for games with a single random node; see, e.g., [8].

For the following claims, let us consider a BW or BWR-game $\mathcal{G}$ in which (w, w) is a self-loop, with no other arcs leaving w. For $x \in \mathbb{R}$, let us denote by $\mathcal{G}(x)$ be the game obtained from $\mathcal{G}$ by setting $r(w, w) = x$.

Lemma 4. *Let v be a node in a BW-game $\mathcal{G}$, and $x, y \in \mathbb{R}$.*

(i) If $\mu_{\mathcal{G}(x)}(v) < x$, then for any $y \geq \mu_{\mathcal{G}(x)}(v)$, $\mu_{\mathcal{G}(y)}(v) = \mu_{\mathcal{G}(x)}(v)$;
(ii) if $\mu_{\mathcal{G}(x)}(v) > x$, then for any $y \leq \mu_{\mathcal{G}(x)}(v)$, $\mu_{\mathcal{G}(y)}(v) = \mu_{\mathcal{G}(x)}(v)$;
(iii) if $\mu_{\mathcal{G}(x)}(v) = x$, then for any $y > x$, $y \geq \mu_{\mathcal{G}(y)}(v) \geq x$;
(iv) if $\mu_{\mathcal{G}(x)}(v) = x$, then for any $y < x$, $y \leq \mu_{\mathcal{G}(y)}(v) \leq x$.

Corollary 2. *For any node $v \in V$ of a BW-game $\mathcal{G}$ there is an interval $I(v) := [\theta_l(v), \theta_u(v)]$, such that*

(i) $\mu_{\mathcal{G}(x)}(v) = \theta_u(v)$ if $x \geq \theta_u(v)$;
(ii) $\mu_{\mathcal{G}(x)}(v) = \theta_l(v)$ if $x \leq \theta_l(v)$;
(iii) $\mu_{\mathcal{G}(x)}(v) = x$ if $x \in I(v)$.

Lemma 5. *For a BWR-game $\mathcal{G}$, the set $I(\mathcal{G}(x)) := \{x \in \mathbb{R} : \mathcal{G}(x)$ is ergodic$\}$ forms a closed (if not empty) interval in $\mathbb{R}$.*

Lemma 6. *In a BWR-game $\mathcal{G}$, let $\theta_1 \leq \theta_2$ be two real numbers in $I(\mathcal{G}(x))$, and $s^*_W \in S_W$ and $s^*_B \in S_B$ be optimal* White *and* Black *strategies in the games $\mathcal{G}(\theta_2)$ and $\mathcal{G}(\theta_1)$, respectively. Then (s^*_W, s^*_B) is a pair of optimal strategies in $\mathcal{G}(x)$ for all $x \in [\theta_1, \theta_2]$.*

3 Algorithm

For a node set $S \subseteq V$, we define the *black closure* $\mathrm{cl}_B(S)$ (respectively, the *black semi-closure* $\mathrm{cl}'_B(S)$) of S to be the node set which is recursively obtained from S by adding

(1) a node $v \in V_B$ (respectively, $v \in V_B \cup V_R$), if some arc $(v, u) \in E$ satisfies $u \in S$, or
(2) a node $v \in V_W \cup V_R$ (respectively, $v \in V_W$), if all arcs $(v, u) \in E$ satisfy $u \in S$.

In words, $\mathrm{cl}_B(S)$ (respectively, $\mathrm{cl}'_B(S)$) is the set of nodes to which Black can force a move with probability 1 (respectively, with some positive probability).

The white closure and semi-closure of S, $\mathrm{cl}_W(S)$ and $\mathrm{cl}'_W(S)$, are defined analogously. Associated with a black closure (respectively white closure $\mathrm{cl}_W(S)$) is a (partial) strategy $s_B(\mathrm{cl}_B(S))$ (respectively, $s_W(\mathrm{cl}_W(S))$) which guarantees Black (respectively, White) a move into S. Similar strategies are defined with respect to semi-closures.

3.1 Discovering Ergodicity in BWR-Games with a Special Self-loop

We first recall the following result on the accuracy needed for solving a rational BWR-game.

Proposition 2 ([7]). *Let $\mathcal{G}$ be a BWR-game with n nodes, k random nodes, local integral rewards of maximum absolute value R, and rational transition probabilities with common denominator D. Then for any node v,*

(i) the value $\mu_{\mathcal{G}}(v)$ is a rational number p/q with $p, q \in \mathbb{Z}$ and $|p| \leq (2D)^{(k+1)n}R$ and $|q| \leq (2D)^{(k+1)n}$;
(ii) for any situation s, if v belongs to some absorbing class C in $\mathcal{G}_s$, the value $\mu_{\mathcal{G}_s}(v)$ is a rational number p/q with $p, q \in \mathbb{Z}$ and $|p| \leq D^{k+1}R$ and $|q| \leq D^{k+1}$. In particular, if the game is ergodic, then the value $\mu_{\mathcal{G}}(v)$ satisfies the latter condition.

The following result strengthens the results in [7, 19, 32, 35], and can be obtained from the pumping algorithm and its analysis in [7].

Theorem 2. *Let $\mathcal{G}$ be a BWR-game with n nodes, k random nodes, rational transition probabilities with common denominator D, and assume that all local rewards are integral of maximum absolute R, except on a single self-loop (w, w), where the reward can be arbitrary.*

(i) There is an algorithm BWR-IsErgodic($\mathcal{G}$) that, given any such BWR-game $\mathcal{G}$, finds whether or not $\mathcal{G}$ is ergodic in time $(nD)^{O(k)}R\log R$; if $\mathcal{G}$ is ergodic, the algorithm finds also an optimal situation; otherwise, it finds a situation certifying non-ergodicity.
(ii) If the given game $\mathcal{G}$ is also a BW-game, then there is an algorithm BW-Solve($\mathcal{G}$) that finds an optimal strategy in $\mathcal{G}$ in time $\mathrm{poly}(n) \cdot R\log R$.

3.2 Parametric Search

Our algorithm uses the following auxiliary routines:

BW-FindIntervals($\mathcal{G}(x)$): Let $\mathcal{G}(x)$ be a BW-game with a self-loop of reward x. By Corollary 2, the end-points of the intervals $I(v)$, for $v \in V$, partition the range $[-R, R]$ into a set of at most $2\nu(\mathcal{G})+1 \leq 2n+1$ intervals $\mathcal{I} := \mathcal{I}(\mathcal{G}(x))$, such that the "structure" of the game with respect to the dependence of the nodes' values on x is fixed over each interval. That is, for each $I = [\theta_l(I), \theta_u(I)] \in \mathcal{I}$, there is a uniquely defined partition $S^l(I) \cup S^m(I) \cup S^u(I) = V$, such that $\mu_{\mathcal{G}(x)}(v) = \theta_u(v)$ for all $v \in S^l(I)$ (where $\theta_u(v) < \theta_l(I)$), $\mu_{\mathcal{G}(x)}(v) = x$ for all $v \in S^m(I)$, and $\mu_{\mathcal{G}(x)}(v) = \theta_l(v)$ for all $v \in S^u(I)$ (where $\theta_l(v) > \theta_u(I)$) (indeed, $S^m(I)$ is defined as the set S such that $I = \bigcap_{v \in S} I(v)$). BW-FindIntervals($\mathcal{G}(x)$) finds this set of intervals $\mathcal{I}(\mathcal{G}(x))$ together with the partitions $S^l, S^m, S^u : \mathcal{I} \to 2^V$, as follows. Given an interval $I := [\theta_l, \theta_u]$, we apply binary search to find the first point $\theta_1 \in [\theta_l, \theta_u]$ such that there is a node $v \in V$ with $\mu_{\mathcal{G}(\theta_1)}(v) \neq \mu_{\mathcal{G}(\theta_l)}(v)$. Each binary search step involves calling BW-Solve($\mathcal{G}(x)$) for some $x \in [-R, R]$

which, in view of Proposition 2, can be chosen rational of the form p/q with $p, q \in \mathbb{Z}$ and $|p| \leq D^{O((k+1)n)}R$ and $|q| \leq D^{O((k+1)n)}$. It follows that the total number of binary search steps needed until we find θ_1 is $O(nk \log D + \log R)$. Once θ_1 is found, we can find the next point $\theta_2 \in [\theta_1, R]$ at which the distribution of values changes, and so on until the whole set of intervals $\mathcal{I}(\mathcal{G}(x))$ is found.

BWR-FindErgodicityInterval($\mathcal{G}(x)$): Let $\mathcal{G}(x)$ be a BWR-game with a self-loop of reward x. By Lemma 5, the set of values of $x \in [-R, R]$ for which $\mathcal{G}(x)$ is ergodic is an interval $I_e(\mathcal{G}(x))$. Given $\mathcal{G}(x)$, BWR-FindErgodicityInterval($\mathcal{G}(x)$) finds $I_e(\mathcal{G}(x))$ by employing binary search, calling in each step the procedure BWR-isErgodic($\mathcal{G}(x)$). Suppose that we start the search on the interval $[\theta_l, \theta_u]$. If the game is non-ergodic, the procedure will return a node $v \in V$, and either a strategy $s_B \in S_B$ certifying that $\mu_{\mathcal{G}(x)}(v) < x$ or a strategy $s_W \in S_W$ certifying that $\mu_{\mathcal{G}(x)}(v) > x$. In the former case, we reduce the search into the interval $[\theta_l, x]$, and in the latter case, we reduce the search into the interval $[x, \theta_u]$. Again, we need only to consider polynomially many search steps by Proposition 2.

3.3 Algorithm Description

For a BWR-game $\mathcal{G}$ and a parameter $x \in \mathbb{R}$, define $\widehat{\mathcal{G}}(x)$ to be the BW-game obtained from $\mathcal{G}$ by replacing each random node $v \in V_R$ in $\mathcal{G}$ with a terminal deterministic node (black or white, arbitrarily) with a local reward of value x on the self-loop (v, v).

For a node $v \in V_R$, we define $\text{rank}(v) = |\{\mu_{\mathcal{G}}(u) \mid u \in V_R,\ \mu_{\mathcal{G}}(u) > \mu_{\mathcal{G}}(v)\}|+1$. For each $v \in V_R$, our algorithm guesses its rank as $g(v)$ (there are at most k^k possible guesses).

For each such guess $g : V_R \to [k]$, we call procedure BWR-Solve($\mathcal{G}, U, g, \ell$) with $U = V$ and $\ell = 1$. At any point in time, U represents the set of nodes for which the value is not yet fixed by the procedure.

The procedure returns a situation s^* which we check for optimality using linear programming (see, e.g., [30]). We now describe this procedure BWR-Solve($\cdot$). For an integer $\ell \in [k]$, define $\mathcal{G}^\ell$ to be the BWR-game obtained from $\mathcal{G}$ by removing all nodes in the black closure $\text{cl}_B(\bigcup_{h>\ell}\{v \in V_R :\ g(v) = h\})$. We first form the game $\widehat{\mathcal{G}}^\ell(x)$ defined from $\mathcal{G}^\ell$ by contracting all random nodes into a single node w with self-loop (w, w) of reward x. Then we find the set of intervals $\mathcal{I}(\widehat{\mathcal{G}}^\ell(x))$ using the routine BW-FindIntervals($\widehat{\mathcal{G}}^\ell(x)$) described above. Then for each such interval $I = [\theta_l(I), \theta_u(I)]$, we consider three sub-games, defined by the sets $S^u(I)$, $S^m(I)$, and $S^l(I)$. By the definition of $S^u(I)$, the first sub-game $\mathcal{G}[S^u(I)]$ is a BW-game that does not depend on x, since all random nodes have either been replaced by a self-loop with reward x or have been deleted since their guessed rank is strictly larger than ℓ. Hence, the optimal strategy $s^*[S^u(I)]$ in $\mathcal{G}[S^u(I)]$ can be obtained by calling BW-Solve($\mathcal{G}[S^u(I)]$). The nodes in the second sub-game $\mathcal{G}[S^m(I)]$ have the same value x. Although we do not know what the exact value of x is, we can find the interval of ergodicity of the BWR-game $\mathcal{G}[S^m(I)](x)$ by calling procedure BWR-FindErgodicityInterval($\mathcal{G}[S^m(I)](x)$) (step 7). Once we determine this interval $I_e = [\theta_1, \theta_2]$, we solve the two ergodic games $\mathcal{G}[S^m(I)](\theta_1)$

Algorithm 1. BWR-Solve($\mathcal{G}, U, g, \ell$)

Input: A BWR-game $\mathcal{G} = (G = (V = (V_B \cup V_W \cup V_R), E), p, r)$, a set of nodes $U \subseteq V$, a vector of rank guesses $g : V_r \to [k]$, and an integer ℓ
Output: Either an optimal situation s^* in $\mathcal{G}$ or "INVALID guess"

if $V_R = \emptyset$ **then**
  s :=BW-Solve($\mathcal{G}[U]$)
else
  $(\mathcal{I}, S^l, S^m, S^u)$:=BW-FindIntervals($\widehat{\mathcal{G}}^\ell[U](x)$)
  for each $I = [\theta_l(I), \theta_u(I)] \in \mathcal{I}$ **do**
    $s^*[S^u(I)]$:=BW-Solve($\mathcal{G}[S^u(I)]$)
    $I_e = [\theta_1, \theta_2]$:=BWR-FindErgodicityInterval($\mathcal{G}[S^m(I)](x)$)
    s^1 :=BWR-IsErgodic($\mathcal{G}[S^m(I)](\theta_1)$)
    s^2 :=BWR-IsErgodic($\mathcal{G}[S^m(I)](\theta_2)$)
    $s^*[S^m(I)] := (s^2_W, s^1_B)$
    $s^*[U \setminus (S^u(I) \cup S^m(I))]$:=BWR-Solve($\mathcal{G}, U \setminus (S^u(I) \cup S^m(I)), g, \ell + 1$)
    if s^* is optimal in $\mathcal{G}$ **then**
      return s^*
    else
      return "INVALID guess"
    end if
  end for
end if

and $\mathcal{G}[S^m(I)](\theta_2)$ using procedure BWR-IsErgodic($\cdot$), and then combine the strategies according to Lemma 6 to obtain an optimal situation for $\mathcal{G}[S^m(I)]$. Finally, the rest of the game is solved by calling the procedure recursively with $\mathcal{G} := \mathcal{G}[U \setminus (S^u(I) \cup S^m(I))]$ and $\ell := \ell + 1$.

The following lemma states that if the guess is correct, then the procedure actually solves the game.

Lemma 7. *Let $\mathcal{G}$ be a BWR-game. If procedure BWR-Solve($\mathcal{G}, U, g, \ell$) is called with $g(v) = \mathrm{rank}(v)$ for all $v \in V_R$, $U = V$ and $\ell = 1$, then it returns an optimal situation s^*.*

Lemma 8. *$\nu(\mathcal{G}) \leq n$ for a BWR-game, and $\nu(\mathcal{G}) \leq t + 1$ for a stochastic terminal payoff game with t terminals of distinct rewards.*

Note that the depth of the recursion tree is at most k. The number of intervals tried at each recursion level is at most $2\nu(\mathcal{G}) + 1$. The claimed bound on the running time follows.

Acknowledgments. The first author thanks the National Science Foundation for partial support (Grants CMMI-0856663 and IIS-1161476.) The fourth author was partially supported by Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

1. Andersson, D., Miltersen, P.B.: The complexity of solving stochastic games on graphs. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 112–121. Springer, Heidelberg (2009)
2. Beffara, E., Vorobyov, S.: Adapting gurvich-karzanov-khachiyan's algorithm for parity games: Implementation and experimentation. Technical Report 2001-020, Department of Information Technology, Uppsala University (2001), https://www.it.uu.se/research/reports/#2001
3. Beffara, E., Vorobyov, S.: Is randomized gurvich-karzanov-khachiyan's algorithm for parity games polynomial? Technical Report 2001-025, Department of Information Technology, Uppsala University (2001), https://www.it.uu.se/research/reports/#2001
4. Björklund, H., Vorobyov, S.: Combinatorial structure and randomized subexponential algorithm for infinite games. Theoretical Computer Science 349(3), 347–360 (2005)
5. Björklund, H., Vorobyov, S.: A combinatorial strongly sub-exponential strategy improvement algorithm for mean payoff games. Discrete Applied Mathematics 155(2), 210–229 (2007)
6. Boros, E., Elbassioni, K., Fouz, M., Gurvich, V., Makino, K., Manthey, B.: Stochastic mean payoff games: Smoothed analysis and approximation schemes. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 147–158. Springer, Heidelberg (2011)
7. Boros, E., Elbassioni, K., Gurvich, V., Makino, K.: A pumping algorithm for ergodic stochastic mean payoff games with perfect information. In: Eisenbrand, F., Shepherd, F.B. (eds.) IPCO 2010. LNCS, vol. 6080, pp. 341–354. Springer, Heidelberg (2010)
8. Boros, E., Elbassioni, K., Gurvich, V., Makino, K.: Discounted approximations of undiscounted stochastic games and markov decision processes are already poor in the almost deterministic case. Operations Research Letters (to appear, 2013)
9. Boros, E., Elbassioni, K., Gurvich, V., Makino, K.: On canocical forms for zero-sum stochastic mean payoff games. In: Dynamic Games and Applications (2013), doi:10.1007/s13235-013-0075-x; Special volume dedicated to the 60th anniversary of Shapley's 1953 paper on stochastic games
10. Chatterjee, K., Henzinger, T.A.: Reduction of stochastic parity to stochastic mean-payoff games. Inf. Process. Lett. 106(1), 1–7 (2008)
11. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Quantitative stochastic parity games. In: SODA 2004, pp. 121–130. Society for Industrial and Applied Mathematics, Philadelphia (2004)
12. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Termination criteria for solving concurrent safety and reachability games. In: SODA, pp. 197–206 (2009)
13. Condon, A.: The complexity of stochastic games. Information and Computation 96, 203–224 (1992)
14. Condon, A.: An algorithm for simple stochastic games. In: Advances in Computational Complexity Theory. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13 (1993)
15. Dai, D., Ge, R.: Another sub-exponential algorithm for the simple stochastic game. Algorithmica 61, 1092–1104 (2011)
16. Eherenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. International Journal of Game Theory 8, 109–113 (1979)

17. Gillette, D.: Stochastic games with zero stop probabilities. In: Tucker, A.W., Dresher, M., Wolfe, P. (eds.) Contribution to the Theory of Games III. Annals of Mathematics Studies, vol. 39, pp. 179–187. Princeton University Press (1957)
18. Gimbert, H., Horn, F.: Simple stochastic games with few random vertices are easy to solve. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 5–19. Springer, Heidelberg (2008)
19. Gurvich, V., Karzanov, A., Khachiyan, L.: Cyclic games and an algorithm to find minimax cycle means in directed graphs. USSR Computational Mathematics and Mathematical Physics 28, 85–91 (1988)
20. Sandberg, S., Björklund, H., Vorobyov, S.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. DIMACS Technical Report 2004-05, DIMACS, Rutgers University (2004)
21. Halman, N.: Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. Algorithmica 49(1), 37–50 (2007)
22. Ibsen-Jensen, R., Miltersen, P.B.: Solving simple stochastic games with few coin toss positions. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 636–647. Springer, Heidelberg (2012)
23. Jurdziński, M.: Deciding the winner in parity games is in UP ∩ co-UP. Inf. Process. Lett. 68(3), 119–124 (1998)
24. Jurdziński, M.: Games for Verification: Algorithmic Issues. PhD thesis, Department of Computer Science, University of Aarhus, Denmark (2000)
25. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: SODA 2006, pp. 117–123. ACM, New York (2006)
26. Karp, R.M.: A characterization of the minimum cycle mean in a digraph. Discrete Math. 23, 309–311 (1978)
27. Karzanov, A.V., Lebedev, V.N.: Cyclical games with prohibition. Mathematical Programming 60, 277–293 (1993)
28. Liggett, T.M., Lippman, S.A.: Stochastic games with perfect information and time-average payoff. SIAM Review 4, 604–607 (1969)
29. Littman, M.L.: Algorithm for sequential decision making, CS-96-09. PhD thesis, Dept. of Computer Science, Brown Univ., USA (1996)
30. Mine, H., Osaki, S.: Markovian decision process. American Elsevier Publishing Co., New York (1970)
31. Moulin, H.: Extension of two person zero sum games. Journal of Mathematical Analysis and Application 5(2), 490–507 (1976)
32. Pisaruk, N.N.: Mean cost cyclical games. Mathematics of Operations Research 24(4), 817–828 (1999)
33. Raghavan, T.E.S., Filar, J.A.: Algorithms for stochastic games- a survey. Mathematical Methods of Operations Research 35(6), 437–472 (1991)
34. Vorobyov, S.: Cyclic games and linear programming. Discrete Applied Mathematics, Special volume in Memory of Leonid Khachiyan (1952 - 2005) 156(11), 2195–2231 (2008)
35. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theoretical Computer Science 158(1-2), 343–359 (1996)

Direct Product via Round-Preserving Compression

Mark Braverman[1,⋆] , Anup Rao[2,⋆⋆],
Omri Weinstein[1,⋆⋆⋆], and Amir Yehudayoff[3,†]

[1] Princeton University
[2] University of Washington
[3] Technion IIT

Abstract. We obtain a strong direct product theorem for two-party bounded round communication complexity. Let $\mathsf{suc}_r(\mu, f, C)$ denote the maximum success probability of an r-round communication protocol that uses at most C bits of communication in computing $f(x, y)$ when $(x, y) \sim \mu$. Jain et al. [12] have recently showed that if $\mathsf{suc}_r(\mu, f, C) \leq \frac{2}{3}$ and $T \ll (C - \Omega(r^2)) \cdot \frac{n}{r}$, then $\mathsf{suc}_r(\mu^n, f^n, T) \leq \exp(-\Omega(n/r^2))$. Here we prove that if $\mathsf{suc}_{7r}(\mu, f, C) \leq \frac{2}{3}$ and $T \ll (C - \Omega(r \log r)) \cdot n$ then $\mathsf{suc}_r(\mu^n, f^n, T) \leq \exp(-\Omega(n))$. Up to a $\log r$ factor, our result asymptotically matches the upper bound on $\mathsf{suc}_{7r}(\mu^n, f^n, T)$ given by the trivial solution which applies the per-copy optimal protocol independently to each coordinate. The proof relies on a compression scheme that improves the tradeoff between the number of rounds and the communication complexity over known compression schemes.

1 Introduction

We study the *direct sum* and the *direct product* problem for bounded-round randomized communication complexity. The direct sum problem studies the amount of resources needed to solve n independent copies of a task in terms of the cost of solving one copy. It is the case that if one copy costs C resources, then n copies can be solved using $C_n \leq n \cdot C$ resources. Can one do better? Direct sum theorems answer this question by giving lower bounds for C_n in terms of C and n — aiming to give a tight $\Omega(n \cdot C)$ bound whenever possible. If the task is solved in a

⋆ Department of Computer Science, Princeton University, Research supported in part by an Alfred P. Sloan Fellowship, an NSF CAREER award (CCF-1149888), and a Turing Centenary Fellowship.

⋆⋆ Computer Science and Engineering, University of Washington, Supported by the National Science Foundation under agreement CCF-1016565, an NSF Career award and by the Binational Science Foundation under agreement 2010089. Part of this work was done while the author was visiting Princeton University and the Technion.

⋆⋆⋆ Department of Computer Science, Princeton University,

† Department of Mathematics, Technion-IIT, Haifa, Israel, Horev Fellow – supported by the Taub Foundation. Supported by the Israel Science Foundation and by the Binational Science Foundation under agreement 2010089.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 232–243, 2013.

randomized model, with some error allowed, the performance of a solution for a single copy of the task is characterized by its cost C and its success probability ρ. Clearly, with $n \cdot C$ resources a success probability of at least ρ^n is attainable, but is it optimal? A direct product theorem is stronger than a direct sum theorem in that in addition to asserting that a certain amount of resources is necessary to compute the n copies, it also shows that using a smaller amount of resources will lead to a very low (possibly exponentially small) success probability.

Direct product theorems have a long history in complexity theory, and in communication complexity in particular [19,16,21,11,12,6]. See [12] for a discussion of the various direct product theorems. In the context of communication complexity, direct product results for specific lower-bound techniques were given by a number of papers: for discrepancy in the two party case by Shaltiel [21] and Lee, Shraibman and Spalek [17], by Sherstov for generalized discrepancy [22], and by Viola and Wigderson for the multiparty case [23]. More recently, a direct product theorem was given by Jain and Yao in terms of the smooth rectangle bound [13]. Direct product results for specific communication problems such as set disjointness include [15,2]. Famous examples for direct product theorems for other models of computation include Yao's XOR lemma and Raz's parallel repetition theorem [20]. For (unbounded-round) communication complexity, the current state-of-the-art results are given by [6], which shows that n copies of a function cost $\Omega(\sqrt{n})$ times the cost of one copy, and any computation using less communication will fail except with an exponentially small probability. [13], building on [14], obtains a strong direct product theorem in terms of the smooth rectangle bound – showing that a strong direct product theorem holds for the communication complexity of a large number of commonly studied functions.

In this paper we focus on the *bounded-round*, distributional, two party communication complexity model. Bounded-round communication complexity is used extensively in streaming and sketching lower bounds (see e.g. [9,18] and references therein). We prove a tight direct sum and direct product theorem for this model. The two players are given inputs according to a distribution $(x, y) \sim \mu$ and need to compute a function $f(x, y)$. The players perform the computation using a communication protocol π. In the bounded-round model, the players are allowed a total of at most r messages in their protocol π. The *communication cost* $\|\pi\|$ of a protocol π is the (worst-case) number of bits the players send when running π. If π has r rounds then $\|\pi\| \geq r$. The *success probability* of π, denoted $\mathsf{suc}(\mu, f, \pi)$, is the probability it outputs the correct value of f (for a formal definition see Section 3.3). The probability that *any* r-round protocol of communication cost C succeeds at computing f is denoted by

$$\mathsf{suc}_r(\mu, f, C) := \max_{\pi \text{ is } r\text{-round and } \|\pi\| \leq C} \mathsf{suc}(\mu, f, \pi).$$

The *unbounded round* success probability $\mathsf{suc}(\mu, f, C)$ is defined as $\mathsf{suc}_C(\mu, f, C)$ (the trivial bound of C does not limit interaction, as $r \leq C$ by definition).

The function $f^n((x_1, \ldots, x_n), (y_1, \ldots, y_n))$ is just the concatenation of n copies of f. In other words, it outputs $(f(x_1, y_1), \ldots, f(x_n, y_n))$. Assume that $\mathsf{suc}_r(\mu, f, C) < 2/3$. Both the direct sum and the direct product question ask

what can be said about the cost, and the success probability of solving f^n. A strong *direct sum* theorem for bounded-round computation would assert that $\mathsf{suc}_{\alpha r}(\mu^n, f^n, \alpha n \cdot C) < 3/4$, for some constant $\alpha > 0$. A *direct product* theorem would further assert that $\mathsf{suc}_{\alpha r}(\mu^n, f^n, \alpha n \cdot C) < (2/3)^{\alpha n}$. Clearly, the latter statement is the best one can hope for up to constants, since trivially $\mathsf{suc}_r(\mu^n, f^n, n \cdot C) \geq \mathsf{suc}_r(\mu, f, C)^n$.

Prior to the present work, several general direct sum and direct product results for bounded-round communication complexity were given. The work [10] by Harsha, Jain, McAllester and Radhakrishnan gives a strong direct sum result for bounded-round communication, but it only works for product distributions (i.e. when μ is of the form $\mu = \mu_x \times \mu_y$). The paper [5] by Braverman and Rao gives a direct sum result for bounded-round communication of the following form: if $\mathsf{suc}(\mu, f, C) < 2/3$, then $\mathsf{suc}_r(\mu^n, f^n, n \cdot C \cdot (1 - o(1))) < 3/4$, for n sufficiently large. This result gives a tight dependence on the communication complexity, but assumes a lower bound on the communication complexity of a single copy of f *without restriction on the number of rounds*. Therefore, strictly speaking, it is not a direct sum result for *bounded-round* communication complexity. The only general *direct product* result for bounded-round communication complexity was recently given by a Jain, Pereszlenyi, and Yao [12], who showed that if $\mathsf{suc}_r(\mu, f, C) \leq \frac{2}{3}$ and $T \ll (C - \Omega(r^2)) \cdot \frac{n}{r}$, then $\mathsf{suc}_r(\mu^n, f^n, T) \leq \exp(-\Omega(n/r^2))$. This result is indeed a proper direct product theorem for bounded-round communication. Its parameters are sub-optimal in two respects: (1) there is no reason for the direct product theorem to not hold all the way to $T = \Omega(C \cdot n)$, and (2) in a tight direct product theorem the success probability $\mathsf{suc}_r(\mu^n, f^n, T)$ would be $\exp(-\Omega(n)) \ll \exp(-\Omega(n/r^2))$.

Our Results. Our main result is an optimal (up to constants and a $\log r$ factor) direct product theorem for bounded-round communication complexity (see Theorem 2 below). The theorem improves over the parameters in [12], with the exception of the dependence on the number of rounds: we require a lower bound for protocols using $7r$ rounds of communication for one copy to get a lower bound for an r-round protocol for n copies. Using Yao's minimax principle [24], our result also applies to the randomized bounded-round communication complexity.

Our Techniques. Our general strategy is similar to other recent direct sum and direct product results [10,1,12,6]. The first main ingredient is the notion of *information cost* of protocols. The information cost of a two-party protocol π over a distribution μ of inputs $(x, y) \sim \mu$ is defined as the amount of information the parties learn about each other's inputs from the messages of the protocol. More formally, if we define X, Y to be the random variables representing the inputs, and M to be the random variable representing the messages or transcript, then the information cost of π with respect to μ is given by

$$IC(\pi, \mu) := I(X; M|Y) + I(Y; M|X),$$

where $I(A; B|C)$ is the mutual information between A and B conditioned on C.

In general, direct sum and direct product proofs proceed in two steps: As a first step, it is shown that if f^n can be solved using fewer than T resources, then one copy of f can be solved using a protocol π, that while having high communication complexity (T), has low information complexity: $IC(\pi, \mu) = O(T/n)$.[1] The second step is to convert the protocol π into a protocol π' that has low communication cost, such as $O(IC(\pi, \mu)))$. This is done through *protocol compression*: the process of converting a low-information interactive protocol into a low communication protocol. If successful, this step leads to a low-communication protocol for one copy of f, which contradicts the initial lower bound assumption on one copy of f.

The process of obtaining new direct sum results in communication complexity has been tightly linked to the process of obtaining new protocol compression results. In fact, the question of whether the general (unbounded-round) direct sum for communication complexity holds is equivalent to the question of whether all protocols can be compressed [5,4]. In the case of bounded-round protocols the problem of compressing protocols reduces to the problem of compressing individual messages in the protocol. The problem of message compression can be rephrased as follows: player 1 has a distribution P of the message $M \sim P$ he wants to send to player 2. Player 2 has some prior belief Q about the distribution of M. How much communication is needed to ensure that both players jointly sample $M \sim P$? The natural information-theoretic lower bound for this problem is the KL-divergence $\mathsf{D}\left(\frac{P}{Q}\right)$. More specifically, if the element being sampled is a, we should expect player 1 to communicate at least $\log(P(a)/Q(a))$ bits to player 2.

If we start off with the assumption that it is hard to solve one copy of f using a *bounded-round* protocol, then to obtain a contradiction our compression scheme should preserve (or at least not blow-up) the number of rounds in the protocol. This means, ideally, that compression of one round should take only a constant number of rounds. The round-compression scheme of [5], in fact, manages to attain near-optimal compression in terms of communication cost. The communication cost of the problem described above is reduced to $\mathsf{D}\left(\frac{P}{Q}\right) \cdot (1 + o(1)) + O(\log 1/\epsilon)$, where ϵ is an error parameter. There is a price to be paid for such communication performance: there is no good bound on the number of rounds such compression would take. Thus the resulting compressed protocol is no longer bounded-round. Therefore, [5] only obtains a lower bound on the bounded-round communication complexity of f^n in terms of *the unbounded-round communication complexity of* f.

The recent works [11,12] devise a different compression scheme that does not increase the number of rounds at all: each message in the original protocol is compressed into one message in the compressed protocol. As a result, these works obtain direct product theorems for bounded-round communication complexity.

[1] In the case of direct product, what is shown is that π is statistically close to being a low information protocol.

These compressions, however, end up paying a high price in the communication overhead. Specifically, due to an application of Markov inequality, sending a message a, on average, takes $r \cdot \log(P(a)/Q(a))$ bits – a multiplicative loss by an r factor, which leads to a factor-r loss in the ultimate result.

Our main technical contribution is a new family of compression protocols for compressing one round of communication. These protocols are parameterized by two parameters (d, ℓ). They give a tradeoff between the communication overhead and the resulting number of rounds. Specifically:

Theorem 1. *For any $a, \ell > 0$, let $\log_\ell^+(a) = \max\{0, \log_\ell(a)\}$. Suppose that player 1 is given a distribution P (unknown to Player 2), and player 2 is given a distribution Q, both over a universe $\mathcal{U}$. Then, for every $0 < \epsilon < 1/2$, $d \geq 1$ and integer $\ell \geq 2$, there is a protocol such that at the end of the protocol:*

- *player 1 outputs an element a distributed according to P.*
- *player 2 outputs an element b s.t for each $x \in \mathcal{U}$, $\Pr[b = a | a = x] > 1 - \epsilon$.*
- *the communication is at most $(2\ell+1) \cdot \log_2^+(P(a)/Q(a)) + 2\log(1/\epsilon) + 2d + 5$.*
- *the number of rounds is at most $2\log_\ell^+[(1/d)\log_2^+(P(a)/Q(a))] + 2$.*

The second condition implies in particular that player 2 outputs an element b such that $b = a$ with probability at least $1 - \epsilon$. The protocol requires no prior knowledge or assumptions on P, Q.

One can see that setting d and ℓ to be large in Theorem 1 will result in few rounds but long communication, and vice versa. The compression scheme in Theorem 1 may be of independent interest. It is possible to view both compression schemes from [5] and from [11,12] as special cases of Theorem 1. The scheme in [5] approximately corresponds to $(d, \ell) = (2, 1)$. The scheme in [11,12] corresponds to $d = \Theta(IC(\pi, \mu))$. By carefully choosing the parameters in Theorem 1, and analyzing the resulting number of rounds and communication cost over all rounds simultaneously, we obtain a compression scheme that at the same time increases the communication cost and the number of rounds of communication by only a constant. This scheme, together with direct product reductions from [6], allows us to complete the proof of Theorem 2.

Discussion and Open Problems. Our work essentially closes the direct product question in the regime where the number of rounds r is small compared to C, and $\mathsf{suc}_r(\mu, f, C)$ is constant in $(0, 1)$. The general direct product problem (and even the weaker direct sum problem) remains wide open. The key compression challenge one needs to overcome is the problem of compressing protocols when $r \gg I$, that is, when the amount of information π conveys in a typical round is $o(1)$. Further discussion on this problem can be found in [4,3].

An important area of tradeoff – both in terms of direct sum/product results and in terms of compression is the relationship between error, communication complexity, and the number of rounds. When performing compression to a bounded number of rounds r, we inevitably have to abort the protocol if the rounds "quota" is exceeded. What is the effect this has on error incurred? A very recent work by Brody, Chakrabarti, and Kondapally [8] suggests the general

tradeoff may take an interesting form. Understanding these tradeoffs is crucial for getting tight parameters for bounded-round direct sum and product in the regime where $\mathsf{suc}_r(\mu, f, C)$ is very close to 1.

2 Results

Let $\mathsf{suc}_r(\mu, f, C)$ denote the maximum success probability of an r-round communication protocol that uses at most C bits of communication to compute $f(x, y)$ when $x, y \sim \mu$. Denote by $f^n(x_1, \ldots, x_n, y_1, \ldots, y_n)$ the function that maps its inputs to the tuple $(f(x_1, y_1), f(x_2, y_2), \ldots, f(x_n, y_n))$ and μ^n denote the product distribution on n pairs of inputs, where each pair is sampled independently according to μ. We prove the following direct product result.

Theorem 2 (Main Theorem). *Let f be a 2-party Boolean function. There is a universal constant $\alpha > 0$ such that if $\gamma = 1 - \mathsf{suc}_{7r}(\mu, f, C)$, $T \geq 2$, and $T < \alpha n \gamma^2 \left(C - \frac{r \log(r/2\gamma)}{\alpha\gamma} - \frac{r}{\alpha\gamma^2} \right)$, then $\mathsf{suc}_r(\mu^n, f^n, T) \leq \exp\left(-\alpha\gamma^2 n\right)$.*

When $\mathsf{suc}_r(\mu, f, C) \leq \frac{2}{3}$ and $r \log r \ll C$, Theorem 2 ensures that the success probability of any protocol attempting to compute f^n under μ^n using $\ll Cn$ communication and $r/7$ rounds must be exponentially small in n.

Our main technical contribution is showing how to compress bounded-round protocols without introducing (too many) additional rounds.

The first step is the sampling protocol described in Theorem 1, which shows how to jointly and efficiently sample from a desired distribution in an oblivious manner. Suppose player 1 knows a distribution P, player 2 knows a distribution Q, and the players wish to jointly sample from P without knowing the distribution of the other player. It is an extension of a protocol from [5]. The protocol is interactive and the requires multiple rounds. The number of rounds required for the simulation in [5] is $\Theta(\sqrt{\Delta})$, where Δ is the KL divergence between the distributions P and Q. While this suffices for the particular objective in [5], this is more than we can afford here: the compression scheme implies that an r-round protocol which reveals I bits of information can be simulated by an $O(r\sqrt{I})$-round protocol that has $I + o(I)$ communication. The resulting compressed protocol is no longer bounded-round, requiring us to assume a stronger lower bound on the hardness of one copy of f to reach a contradiction. Our new compression protocol ensures that at most $7r$ rounds of communication are used with high probability, which means that assuming that f cannot be efficiently solved by a $7r$-round protocol suffices.

The second step in the proof is showing how to use the single-message sampling protocol from Theorem 1 to simulate communication protocols, with communication comparable to the amount of information they convey, while keeping the number of rounds comparable to the original number. In fact, to prove our main result, we actually need to analyze protocols that are merely close to having low information cost. As noticed in [6], such protocols need not have low information themselves. E.g., consider the protocol π in which player 1 sends her n-bit uniformly random input x with probability ϵ, and otherwise sends a random string.

Then π is ϵ-close to a 0-information protocol, but $IC(\pi) = \epsilon n$. Nevertheless, *truncation* of protocols (as in [6]) implies that compression is possible even in this more general setting. This is formalized by the next theorem.

Theorem 3 (Round preserving compression). *Suppose θ is an r-round protocol with inputs x, y and messages m, and q is another distribution on these variables such that $\theta(xym) \overset{\epsilon}{\approx} q(xym)$. Let $I = I_q(X; M|Y) + I_q(Y; M|X)$. Then there exists a $7r$-round protocol τ that 11ϵ-simulates θ such that*

$$\|\tau\| \leq 7\frac{I}{\epsilon^2} + 2\frac{r\log(r/\epsilon)}{\epsilon} + 30\frac{r}{\epsilon^2}.$$

The compression protocol in Theorem 3 is obtained by sequential applications of Theorem 1. However, in order to prevent a blowup in the number of simulating rounds, we cannot use the guarantees of Theorem 1 on a per-round basis. We analyze the protocol in a global manner, which yields the desirable tradeoff between the number of rounds and the communication complexity.

3 Preliminaries

3.1 Notation

Unless otherwise stated, logarithms in this text are computed in base two. Random variables are denoted by capital letters and values they attain are denoted by lower-case letters. For example, A may be a random variable and then a denotes a value A may attain and we may consider the event $A = a$. Given $a = a_1, a_2, \ldots, a_n$, we write $a_{\leq i}$ to denote $a_1, \ldots, a_i$. We define $a_{>i}$ and $a_{\leq i}$ similarly. For an event E, define $\mathbf{1}_E$ to be the indicator random variable of E.

We use the notation $p(a)$ to denote both the distribution on the variable a, and the number $\Pr_p[A = a]$. The meaning will typically be clear from context, but in cases where there may be confusion we shall be more explicit about which meaning is being used. We write $p(a|b)$ to denote either the distribution of A conditioned on the event $B = b$, or the number $\Pr[A = a|B = b]$. For an event W, we write $p(W)$ to denote the probability of W according to p. We let $\mathbb{E}_{p(a)}[g(a)]$ denote the expected value of $g(a)$ when a is distributed according to p.

For two distributions p, q, we write $|p(a) - q(a)|$ to denote the ℓ_1 distance between the distributions p and q. We write $p \overset{\epsilon}{\approx} q$ if $|p - q| \leq \epsilon$.

The *divergence* between p, q is defined to be

$$\mathsf{D}\left(\frac{p(a)}{q(a)}\right) = \sum_a p(a)\log\frac{p(a)}{q(a)}.$$

For three random variables A, B, C jointly distributed according to $p(a, b, c)$, the *mutual information* between A, B conditioned on C is defined as

$$I_p(A; B|C) = \mathop{\mathbb{E}}_{p(cb)}\left[\mathsf{D}\left(\frac{p(a|bc)}{p(a|c)}\right)\right] = \mathop{\mathbb{E}}_{p(ca)}\left[\mathsf{D}\left(\frac{p(b|ac)}{p(b|c)}\right)\right] = \sum_{a,b,c} p(abc)\log\frac{p(a|bc)}{p(a|c)}.$$

3.2 Properties of Divergence

Lemma 1 (Chain Rule). *If $a = a_1, \ldots, a_s$, then*

$$\mathsf{D}\left(\frac{p(a)}{q(a)}\right) = \sum_{i=1}^{s} \mathop{\mathbb{E}}_{p(a_{<i})}\left[\mathsf{D}\left(\frac{p(a_i|a_{<i})}{q(a_i|a_{<i})}\right)\right].$$

The following lemmas describe basic properties of divergence (for proofs see [6]).

Lemma 2. *Let $S = \{a : p(a) < q(a)\}$. Then, $\sum_{a\in S} p(a) \log \frac{p(a)}{q(a)} \geq -1/(e \ln 2)$.*

Lemma 3 (Truncation Lemma [6]). *Let $p(a,b,c) \overset{\epsilon}{\approx} q(a,b,c)$ where $a = a_1, \ldots, a_s$. For every a, b, c, define k to be the minimum number j in $[s]$ such that*

$$\log \frac{p(a_{\leq j}|bc)}{p(a_{\leq j}|c)} > \beta.$$

If no such index exists, set $k = s + 1$. Then,

$$p(k < s+1) < \frac{I_q(A;B|C) + \log(s+1) + 1/(e\ln 2)}{\beta - 2} + 9\epsilon/2.$$

3.3 Communication Complexity

Given a protocol π that operates on inputs X, Y drawn from a distribution μ and (possibly) using public randomness S and messages M, we write $\pi(xyms)$ to denote the joint distribution of these variables. We write $\|\pi\|$ to denote the *communication complexity* of π, namely the maximum number of bits that may be exchanged by the protocol.

A central measure in this paper is the information complexity of a communication protocol (see [1,4] and references within for a more detailed overview). The *internal information cost* of π is defined to be $IC(\pi) := I_\pi(X; M|YS) + I_\pi(Y; M|XS)$. It is well known (e.g, [4]) that for any protocol π, $IC(\pi) \leq \|\pi\|$.

Let $q(x, y, a)$ be an arbitrary distribution. We say that π *δ-simulates* q, if there is a function g and a function h such that $\pi(x, y, g(x, s, m), h(y, s, m)) \overset{\delta}{\approx} q(x, y, a, a)$, where $q(x, y, a, a)$ is the distribution on 4-tuples (x, y, a, a) where (x, y, a) are distributed according to q. Thus if π δ-simulates q, the protocol allows the parties to sample a according to $q(a|xy)$. If in addition $g(x, s, m)$ does not depend on x, we say that π *strongly* δ-simulates q. Thus if π strongly simulates q, then the outcome of the simulation is apparent even to an observer that does not know x or y.

If λ is a protocol with inputs x, y, public randomness s' and messages m', we say that π δ-simulates λ if π δ-simulates $\lambda(x, y, (s', m'))$. Similarly, we say that π strongly δ-simulates λ if π strongly δ-simulates $\lambda(x, y, (s', m'))$. We say that π computes$f$ with success probability $1-\delta$, if π strongly δ-simulates $\pi(x, y, f(x, y))$. We denote this by $\mathsf{suc}(\mu, f, \pi) = 1 - \delta$.

The next proposition is straightforward. A formal proof can be found in the full version of this paper [7].

Proposition 1. *Let $f : \mathcal{X} \times \mathcal{Y} \longrightarrow \mathcal{Z}$ and let π be such that $\mathsf{suc}(\mu, f, \pi) = 1 - \delta$. Then if λ is a protocol that ϵ-simulates π, there is a protocol τ such that $\mathsf{suc}(\mu, f, \tau) \geq 1 - (\delta + \epsilon)$ and $\|\tau\| = \|\lambda\| + \log|\mathcal{Z}|$. The number of rounds in τ is the same as in π.*

4 Proof of Theorem 1

Proof (of Theorem 1). Due to space constraints, here we only present the sampling protocol. A full analysis of the protocol together with the proof of Theorem 1 appears in the full version of this paper [7].

We start by describing the content of the shared random tape. Both parties interpret part of the shared random tape as a sequence of independent uniformly selected elements $\{e_i\}_{i=1}^{\infty} = \{(x_i, p_i)\}_{i=1}^{\infty}$ from the set $\mathcal{E} := \mathcal{U} \times [0, 1]$. There is also a part of the shared random tape that contains random independent hash functions $\{h_i\}_{i=1}^{\infty}$, that is, for every i, the function $h_i : \mathcal{U} \to \{0, 1\}$ is so that $\Pr[h_i(x) = h_i(y)] = 1/2$ for every $x \neq y$ in $\mathcal{U}$.

The players use the following definitions: Define

$$\mathcal{E}_P := \{(x, p) \in \mathcal{E} : P(x) > p\},$$

the set of points under the histogram of P. Similarly, define

$$\mathcal{E}_Q := \{(y, q) \in \mathcal{E} : Q(y) > q\}.$$

For a constant $C \geq 1$, define the C-multiple of $\mathcal{E}_Q$ as

$$C \cdot \mathcal{E}_Q := \{(y, q) \in \mathcal{E} : (y, q/C) \in \mathcal{E}_Q\}.$$

For a non-negative integer t, set

$$C_t := 2^{d\ell^t} \quad \text{and} \quad s_t := 2d\ell^t + \lceil \log(1/\epsilon) \rceil + 1.$$

The Protocol. The protocol runs as follows:

1. Player 1 selects the first index i such that $e_i = (x_i, p_i) \in \mathcal{E}_P$, and outputs x_i.
2. Player 1 uses $1 + \lceil \log\log(1/\epsilon) \rceil$ bits to send player 2 the binary encoding of
$$k := \lceil i/|\mathcal{U}| \rceil.$$
If $k > 2^{\log\log(1/\epsilon)}$, player 1 sends the all-zero string and the players abort.
3. Repeat, until player 2 produces an output, starting with $t = 0$:
   (a) Player 1 sends the values of all hash functions $h_j(x_i)$ for $1 \leq j \leq s_t$, that have not been previously sent.
   (b) If there is an $a_r = (y_r, q_r)$ with $r \in \{(k-1) \cdot |\mathcal{U}| + 1, \ldots, k \cdot |\mathcal{U}|\}$ in $C_t \cdot \mathcal{E}_Q$ such that $h_j(y_r) = h_j(x_i)$ for some $1 \leq j \leq s_t$, then player 2 says "success" and outputs y_r (if there is more than one such a_r, player 2 selects the first one).
   (c) Otherwise, player 2 responds "failure" and the parties increment t to $t + 1$ and repeat.

5 Round Preserving Compression – Proof of Theorem 3

Proof (of Theorem 3). Our simulating protocol for θ is the protocol σ described in Figure 1 (The final protocol τ will be defined as a truncation of σ). Once again, due to space constraints, here we only present the protocol. For a complete analysis and the rest of the proof of Theorem 3, we refer the reader to the full version of this paper [7].

Protocol σ for simulating θ

Player 1 repeatedly computes a message $m' = m'_1, \ldots, m'_r$ and player 2 repeatedly computes a message $m'' = m''_1, \ldots, m''_r$ as follows.

- For odd j, player 1 sets $P = \theta(m_j | m'_{<j} x)$ and player 2 sets $Q = \theta(m_j | m''_{<j} y)$.
- For even j, player 1 sets $Q = \theta(m_j | m'_{<j} x)$ and player 2 sets $P = \theta(m_j | m''_{<j} y)$.
- In each round j, the players run the protocol from Theorem 1 with error parameter ϵ/r, with $\ell = 2$,
  and with $d = \frac{\beta}{r\epsilon} + \frac{1}{\epsilon}$ where

$$\beta = \frac{I + 1/(e \ln 2) + \log(r+1)}{\epsilon} + 2.$$

  This leaves player 1 with m'_j and player 2 with m''_j.

Fig. 1. A round preserving compression of the protocol θ

6 Direct Product for Bounded Round Protocols

Let π be a (deterministic) r-round protocol for computing f^n with inputs $\overline{x} = x_1, \ldots, x_n$ and $\overline{y} = y_1, \ldots, y_n$ drawn from μ^n. To prove Theorem 2, we follow the approach of [6] which itself resembles the proof of the parallel repetition theorem [20]. Let W be the event that π correctly computes f^n. For $i \in [n]$, let W_i denote the event that the protocol π correctly computes the i'th copy $f(x_i, y_i)$. Let $\pi(W)$ denote the probability of W, and $\pi(W_i|W)$ denote the conditional probability of the event W_i given W (clearly, $\pi(W_i|W) = 1$). We shall prove that if $\pi(W)$ is not very small and $\|\pi\| \ll Cn$, then $(1/n)\sum_{i=1}^{n} \pi(W_i|W) < 1$, which is a contradiction. In fact, the proof holds for an arbitrary event W, as long as it occurs with large enough probability:

Lemma 4 (Main Lemma). *Let f be a 2-party Boolean function. There is a universal constant $\alpha > 0$ so that the following holds. For every $\gamma > 0$, and event W such that $\pi(W) \geq 2^{-\gamma^2 n}$, if $\|\pi\| \geq 2$, and $\|\pi\| < \alpha n \gamma^2 \left(C - \frac{r \log(r/2\gamma)}{\alpha\gamma} - \frac{r}{\alpha\gamma^2}\right)$, then $\frac{1}{n}\sum_{i\in[n]} \pi(W_i|W) \leq \mathsf{suc}_{7r}(\mu, f, C) + \gamma/\alpha$.*

First let us see how Lemma 4 implies Theorem 2. As outlined above, let W denote the event that π computes f correctly in all n coordinates. So, $(1/n)\sum_{i\in[n]}\pi(W_i|W) = 1$. Set $\gamma = \alpha(1 - \mathsf{suc}_{7r}(\mu, f, C))/2$ so that $\mathsf{suc}_{7r}(\mu, f, C) + \gamma/\alpha < 1$. Then by Lemma 4, either $\|\pi\| < 2$, $\|\pi\| \geq \alpha n\gamma^2\left(C - \frac{r\log(r/2\gamma)}{\alpha\gamma} - \frac{r}{\alpha\gamma^2}\right)$, or $\pi(W) < 2^{-\gamma^2 n}$. It therefore remains to prove Lemma 4.

The overall idea is to use π to produce a $7r$-round protocol with communication complexity $< C$ that computes f correctly with probability at least $(1/n)\sum_{i=1}^{n}\pi(W_i|W) - O(\gamma)$. This would imply that $(1/n)\sum_{i\in[n]}\pi(W_i|W) \leq \mathsf{suc}_{7r}(\mu, f, C) + O(\gamma)$, as desired. The first step is to show that there exists a good simulating protocol for a random coordinate of $\pi|W$, whose average information cost is low (roughly $\|\pi\|/n$) and still uses only r rounds. The existence of such protocol was proven in [6], except their protocol is not guaranteed to actually have low information cost, but to merely be statistically close to a low-information protocol. This will suffices for our purpose:

Lemma 5 (Claims 26 and 27 from [6], restated). *There is a protocol σ taking inputs $x, y \sim \mu$ so that the following holds:*

- *σ publicly chooses a uniform $i \in [n]$ independent of x, y, and S_i which is part of the input to π.*
- *$\mathbb{E}_{x,y,m,i,s_i}|\sigma(xys_im) - \pi(x_iy_is_im|W)| \leq 2\gamma$.*
- *$Rounds(\sigma) = Rounds(\pi)$.*
- *$\mathbb{E}_i\left[I_\pi(X_i; M|Y_iS_iiW) + I_\pi(Y_i; M|X_iS_iiW)\right] \leq 4\|\pi\|/n$.*

The second step of the proof of Lemma 4 is to compress the simulating protocol σ so that it actually has low communication, without introducing many additional rounds in the compression process. Since the second and fourth propositions of Lemma 5 imply that σ is 2γ-close to a low-information distribution $q = \pi(x_iy_is_im|W)$, this is precisely the setting of Theorem 3. A formal proof of Lemma 4 can be found in the full version of this paper [7].

References

1. Barak, B., Braverman, M., Chen, X., Rao, A.: How to compress interactive communication. In: Proceedings of the 2010 ACM International Symposium on Theory of Computing, pp. 67–76 (2010)
2. Ben-Aroya, A., Regev, O., de Wolf, R.: A hypercontractive inequality for matrix-valued functions with applications to quantum computing. In: FOCS, pp. 477–486 (2008)
3. Braverman, M.: Coding for interactive computation: progress and challenges. In: 50th Annual Allerton Conference on Communication, Control, and Computing (2012)
4. Braverman, M.: Interactive information complexity. In: Proceedings of the 44th Symposium on Theory of Computing, STOC 2012, pp. 505–524. ACM, New York (2012)

5. Braverman, M., Rao, A.: Information equals amortized communication. In: Ostrovsky, R. (ed.) FOCS, pp. 748–757. IEEE (2011)
6. Braverman, M., Rao, A., Weinstein, O., Yehudayoff, A.: Direct products in communication complexity. Electronic Colloquium on Computational Complexity (ECCC) 19, 143 (2012)
7. Braverman, M., Rao, A., Weinstein, O., Yehudayoff, A.: Direct product via round-preserving compression. Electronic Colloquium on Computational Complexity (ECCC) 20, 35 (2013)
8. Brody, J., Chakrabarti, A., Kondapally, R.: Certifying equality with limited interaction
9. Clarkson, K.L., Woodruff, D.P.: Numerical linear algebra in the streaming model. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 205–214. ACM (2009)
10. Harsha, P., Jain, R., McAllester, D.A., Radhakrishnan, J.: The communication complexity of correlation. In: IEEE Conference on Computational Complexity, pp. 10–23. IEEE Computer Society (2007)
11. Jain, R.: New strong direct product results in communication complexity
12. Jain, R., Pereszlenyi, A., Yao, P.: A direct product theorem for the two-party bounded-round public-coin communication complexity. In: 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science (FOCS), pp. 167–176. IEEE (2012)
13. Jain, R., Yao, P.: A strong direct product theorem in terms of the smooth rectangle bound. CoRR abs/1209.0263 (2012)
14. Kerenidis, I., Laplante, S., Lerays, V., Roland, J., Xiao, D.: Lower bounds on information complexity via zero-communication protocols and applications. Electronic Colloquium on Computational Complexity (ECCC) 19, 38 (2012)
15. Klauck, H.: A strong direct product theorem for disjointness. In: STOC, pp. 77–86 (2010)
16. Lee, T., Shraibman, A., Spalek, R.: A direct product theorem for discrepancy. In: CCC, pp. 71–80 (2008)
17. Lee, T., Shraibman, A., Spalek, R.: A direct product theorem for discrepancy. In: 23rd Annual IEEE Conference on Computational Complexity, CCC 2008, pp. 71–80. IEEE (2008)
18. Molinaro, M., Woodruff, D., Yaroslavtsev, G.: Beating the direct sum theorem in communication complexity with implications for sketching. In: SODA (to appear, 2013)
19. Parnafes, I., Raz, R., Wigderson, A.: Direct product results and the GCD problem, in old and new communication models. In: Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC 1997), pp. 363–372. Association for Computing Machinery, New York (1997)
20. Raz, R.: A parallel repetition theorem. SIAM Journal on Computing 27(3), 763–803 (1998); Prelim version in STOC 1995
21. Shaltiel, R.: Towards proving strong direct product theorems. Computational Complexity 12(1-2), 1–22 (2003); Prelim version CCC 2001
22. Sherstov, A.A.: Strong direct product theorems for quantum communication and query complexity. SIAM Journal on Computing 41(5), 1122–1165 (2012)
23. Viola, E., Wigderson, A.: Norms, xor lemmas, and lower bounds for polynomials and protocols. Theory of Computing 4(1), 137–168 (2008)
24. Yao, A.C.-C.: Probabilistic computations: Toward a unified measure of complexity. In: 18th Annual Symposium on Foundations of Computer Science, pp. 222–227. IEEE (1977)

How Hard Is Counting Triangles in the Streaming Model?*

Vladimir Braverman[1,**], Rafail Ostrovsky[2,***], and Dan Vilenchik[3]

[1] Department of Computer Science, Johns Hopkins University
vova@cs.jhu.edu
[2] Department of Computer Science, UCLA
rafail@cs.ucla.edu
[3] Faculty of Mathematics and Computer Science, The Weizmann Institute, Israel
dan.vilenchik@weizmann.ac.il

Abstract. The problem of (approximately) counting the number of triangles in a graph is one of the basic problems in graph theory. In this paper we study the problem in the streaming model. Specifically, the amount of memory required by a randomized algorithm to solve this problem. In case the algorithm is allowed one pass over the stream, we present a best possible lower bound of $\Omega(m)$ for graphs G with m edges. If a constant number of passes is allowed, we show a lower bound of $\Omega(m/T)$, T the number of triangles. We match, in some sense, this lower bound with a 2-pass $O(m/T^{1/3})$-memory algorithm that solves the problem of distinguishing graphs with no triangles from graphs with at least T triangles. We present a new graph parameter $\rho(G)$ – the triangle density, and conjecture that the space complexity of the triangles problem is $\Theta(m/\rho(G))$. We match this by a second algorithm that solves the distinguishing problem using $O(m/\rho(G))$-memory.

1 Introduction

Counting the number of triangles in a graph $G = (V, E)$ is one of the basic algorithmic questions in graph theory. From a theoretical point of view, the

* The full version of this paper is available at http://arxiv.org/abs/1304.1458

** This work was supported in part by DARPA grant N660001-1-2-4014. Its contents are solely the responsibility of the authors and do not represent the official view of DARPA or the Department of Defense.

*** 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.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 244–254, 2013.

key question is to determine the time and space complexity of the problem. The brute-force approach enumerates all possible triples of nodes (taking $O(n^3)$ time, n is the number of vertices in G). The algorithms with the lowest time complexity for counting triangles rely on fast matrix multiplication. The asymptotically fastest algorithm to date is $O(n^{2.372})$ [17]. An algorithm that runs in time $O(m^{1.41})$ with space complexity $\Theta(n^2)$ is given in [2] (m is the number of edges in G). In more practical applications, the number of triangles is a frequently used network statistic in the exponential random graph model [13,8], and naturally appears in models of real-world network evolution [12], and web applications [4,7]. In the context of social networks, triangles have a natural interpretation: friends of friends tend to be friends [16], and this can be used in link recommendation/friend suggestion [14].

The memory restrictions when dealing with huge input sizes leads to consider the streaming model: The edges of the graph come down the stream, and the algorithm processes each edge as it comes in an on-line fashion (once it moves down the stream, it cannot access it again). The algorithm is allowed ideally one pass (or a limited number of passes) over the stream. The parameter of interest is the amount of *memory* that the algorithm uses to solve the problem. Formally,

Definition 1. TRIANGLES(c) *is the problem of approximating the number of triangles in the input graph within a multiplicative factor of* $9/10$ *with probability at least* $2/3$*, using at most* c *passes over the data stream.*

The choice of constants $9/10$ and $2/3$ in the definition is for the sake of clear and simple presentation. In particular, one can take both the approximation rate and success probability to be parameters of the problem.

Currently, no non-trivial algorithms are known to solve TRIANGLES(c) when c is constant (by trivial we mean an algorithm that uses $\Theta(m)$ memory, m the number of edges, which is asymptotically the same as storing all graph edges). All existing approximation algorithms receive T_3 (the number of triangles in the input graph) as part of their input. Obviously, T_3 cannot be part of the input. One way to implement such an algorithm is by "guessing" the correct value of T_3 and verifying the guess. This translates into $\Theta(\log n)$ passes over the stream (in form of a binary search for example). In light of what we've just said, a fundamental questions remains open.

Question 1: Determine the space complexity of TRIANGLES(1) and TRIANGLES($O(1)$).

Throughout we assume that it takes constant space to represent a single vertex of the graph. This assumption is widely made, and many results are stated under this assumption. Bar-Yossef et. al. [3] showed that the space complexity required to solve TRIANGLES(1) is $\Omega(n^2)$. Specifically, they showed that every one-pass 0.5-approximation algorithm that succeeds with probability 0.99, is as good as the trivial algorithm that stores all edges and exhaustively computes the number of triangles. While the lower bound determines the space complexity in the worst

case, it is informative to study more refined notions of "worst case". For example, what is the space complexity of TRIANGLES(1) when the graph has exactly m edges, or at least T triangles. What is the space complexity of TRIANGLES(2), when two passes over the stream are allowed, rather than one. In those cases the lower bound in [3] is irrelevant.

1.1 Our Results

In this paper we show that the space complexity of any algorithm that solves TRIANGLES(1) (i.e. in *one* pass) is $\Omega(m)$. This lower bound is asymptotically *tight*, since the trivial algorithm that stores all edges of the graph uses that much memory. Furthermore, the lower bound is true even when assuming that the graph has $T_3 = O(n)$ triangles. Clearly one cannot expect this to be the case for every value of T_3, since when $T_3 = \Theta(n^3)$ for example, a straightforward sampling algorithms solves TRIANGLES(1) using $O(1)$-space. Formally,

Theorem 1. *$\exists c_1, c_2 > 0$ s.t. the space complexity of* TRIANGLES(1) *is $\Omega(m)$, when the input is an n-vertex graph with $m \in [c_1 n, c_2 n^2]$ edges. Furthermore, this lower bound is true even if the graph has as many as $0.99n$ triangles.*

Theorem 1 extends the aforementioned result in [3] in two aspects: the number of edges is asymptotically the entire range (compared with $\Theta(n^2)$ in [3]). The graph may contain as many as a linear number of triangles (compared with one triangle in [3]). In addition, our proof technique is conceptually and technically simpler.

Improving upon the currently best known lower bound of $\Omega(n/T_3)$ for TRIANGLES($O(1)$) [10], we show that:

Theorem 2. *The space complexity of* TRIANGLES($O(1)$) *for input graphs with m edges and T_3 triangles is $\Omega(m/\max\{T_3, 1\})$.*

Theorems 1 and 2 are proven, respectively, via reductions from the index problem in communication complexity, and a variant of the set disjointness problem. The proofs are rather standard and are omitted due to page limitation. They appear in the full version of this paper.

From the algorithmic perspective, TRIANGLES(1) has an asymptotically tight lower bound, and a non-trivial solution for TRIANGLES($O(1)$) seems beyond the reach of current techniques. As we already mentioned, all current state-of-the-art approximation algorithms require a super-constant number of passes (regardless of the space complexity). Hence, we start with a softer notion of approximation, in the spirit of property testing.

Definition 2. DIST(c) *is the following problem. Given two graph families: $\mathcal{G}_1$ consisting of triangle-free graphs, $\mathcal{G}_2$ consisting of graphs with at least T triangles, and an input graph $G \in \mathcal{G}_1 \cup \mathcal{G}_2$, decide whether $G \in \mathcal{G}_1$ or $G \in \mathcal{G}_2$ with probability at least $2/3$, using at most c passes over the input.*

The same lower bounds that we derive for TRIANGLES(c) are true for DIST(c) as well (therefore there is nothing interesting to say algorithmically about DIST(1)). Our main result is an algorithm that solves DIST(2) using $O(m/T^{1/3})$ space. We now turn to describe our algorithm in details, and formally state the relevant theorem. We assume that the parameter T is known to the algorithm (as it is part of the problem definition).

Algorithm A

Output: '1' iff a triangle was found.
Pass 1
(a) Set $m' = 6m/T^{1/3}$, and $p = m'/m$.
(b) Store every edge e with probability p. If more than $5m'$ edges are stored, output FAIL.
(c) Let H be the graph stored by the algorithm at the end of **(b)**. Search for a triangle in H, if found output 1.
Pass 2 For every edge e, check whether e completes a triangle in H. Output 1 iff such edge exists.

Fig. 1. A two-pass algorithm with space complexity $O(m/T^{1/3})$

Theorem 3. *For $T \geq 216$, Algorithm A solves* DIST(2) *using at most $30m/T^{1/3}$ space.*

When $T = \omega(1)$, our algorithm solves DIST(2) using *sub-linear* space. Also, our lower bound on the space complexity of DIST(1), together with Algorithm A for DIST(2), imply a space complexity *separation* result between one-pass and two-passes. For example, DIST(2) can be solved in space $O(m/n^{1/3}) = o(m)$ for graphs with $T_3 = n/2$ triangles and m edges, while DIST(1) requires $\Omega(m)$-space for such graphs.

Remark. Algorithm A assumes m is known. This assumption is done only for the sake of clear and simple presentation, and can be easily removed: The algorithm "guesses" an initial value for m, say $m_1 = 1$. This value is used to define p for the first m_1 edges. If the number of edges exceeds that guess, then the algorithm sets $m_2 = 2m_1$, and updates p accordingly for the next m_2 edges. Every time guess i is exceeded, the algorithm sets $m_{i+1} = 2m_i$. The last interval will consist of the last $m/2$ edges. Edges are still stored independently of each other, and in expectation twice as many edges are stored. Storing more edges may only help the algorithm (while not changing the asymptotic space complexity). Hence the same analysis that we have for A goes through with this additional procedure.

1.2 A New Graph Parameter

While the bound given in Theorem 1 for TRIANGLES(1) is asymptotically tight, we suspect that the bound in Theorem 2 for TRIANGLES($O(1)$) is not tight,

and conjecture a tight bound instead. Define the *triangle density* of a graph G, $\rho(G)$, to be the number of vertices that belong to some triangle in G. If G has T triangles, it is easy to see that $(6T)^{1/3} \leq \rho(G) \leq 3T$ (a clique or T disjoint triangles).

Conjecture: The space complexity of TRIANGLES($O(1)$) and DIST($O(1)$) is $\Theta(m/\rho(G))$.

The lower bound in Theorem 2 is consistent with the case $\rho(G) = \Theta(T)$, and Algorithm A is consistent with $\rho(G) = \Theta(T^{1/3})$. We describe a second algorithm that solves DIST(2) using $O(m/\rho(G))$ space, thus showing that one cannot hope for a tighter bound than the one stated in the conjecture. A formal description of the algorithm, a proof of correctness and analysis of its space complexity is given in Section 2.

1.3 Related Work

Let us mention several approaches for approximating the number of triangles in the streaming model. The first, and arguably most natural, is a sampling approach. For example, the 2-pass algorithm suggested in [5], samples in the first pass s random pairs (e, v) of an edge $e = (u, w)$ and a vertex v, and stores them. Then in the second pass checks for every pair (e, v) whether (u, v, w) is a triangle. The total number of triangles is estimated as a function of the number of pairs (e, v) that formed a triangle. The number of samples s is proportional to $(T_1 + T_2 + T_3)/T_3$, where T_i is the number of vertex triples in the graph spanning exactly i edges (one can verify that $T_1 + 2T_2 + 3T_3 = m(n-2)$, where $m = |E(G)|$ and $n = |V(G)|$). A more sophisticated version of that algorithm uses $(T_1 + T_2)/T_3$ samples.

A different approach reduces the problem of approximating the number of triangles to the problem of estimating the frequency moments of the data stream, using the Alon-Matias-Szegedy (AMS) algorithm [1]. This approach was presented in [3] for the first time. The algorithm in [3] uses T_1, T_2, T_3 to compute the appropriate parameters with which to run AMS. The space complexity of this algorithm is proportional to $((T_1 + T_2)/T_3)^3$. In another work [10], the algorithm uses m, C_4, C_6, T_3 to compute the appropriate parameters to AMS (C_i is the number of i-cycles in the graph). The space complexity of that algorithm is $(m^3 + mC_4 + C_6 + T_3^2)/T_3^2$.

One disadvantage that the aforementioned algorithms share is that in the worse case, all of them store $\Omega(m)$ edges, regardless of the actual number of triangles in the graph. One example for such "worse case" input is the graph with vertex set $V = A_0 \cup A_1 \cup A_2$, each A_i of size $n/3$. The edge set E is the complete bi-partite graph on A_0, A_1 and on A_1, A_2. We can vary the number of triangles T_3 by adding isolated triangles, or edges between A_0 and A_2. For a wide range of T_3 values, each of the aforementioned algorithm stores $\Omega(m)$ edges (as in particular $T_2 = \Omega(n^3)$). In light of this, another interesting question arises:

Question 2: Can one solve TRIANGLES(c) using space complexity depending only on the number of edges m and triangles T_3?

The space complexity of our Algorithm A depends only on m and T_3 (see Theorem 3), however it solves DIST(c) and not TRIANGLES(c).

Finally let us mention the *graph scarification* approach, which we also used as an ingredient in Algorithm A (see Figure 1.1). Given a stream of graph G's edges, one computes a sparse image of G by storing every edge independently with probability p, where p is a parameter of the algorithm. This approach lends itself to a 1-pass algorithm, by computing the sparsified image and extracting from it an approximation to the total number of triangles. This approach was analyzed both in [15] and [11], where sufficient conditions for its success were established (assuming some additional information on the triangle structure).

In full generality, without any assumptions on G, this 1-pass algorithm is too naive. Specifically, if the algorithm answers correctly with probability say $2/3$ then p must be constant (think of the case where all triangles are "stacked" on one common edge. This edge will not appear in the sparsified image w.p. at least $1-p$, and a wrong answer is bound to be returned with that probability. Hence $p \geq 1/3$). In other words, the algorithm stores $\Omega(m)$ edges. This is of course consistent with our lower bound in Theorem 1. Although our Algorithm A also uses sparsification, we were able to avoid this pathological case, by introducing a second pass. More details in the next section.

1.4 Theorem 3: Proof Outline

Recall the graph sparsification procedure. Given a graph G, store every edge, independently of the others, with probability p. Let H be the sparsified image of G. If G has at least T triangles, the expected number of triangles in H is p^3T. Taking $p = \Omega(T^{-1/3})$, the expected number of triangles in H is $\Omega(1)$, and the number of edges in H is $O(m/T^{1/3})$. The key question is how concentrated is the number of triangles in H? For example, think of two triangles sharing an edge. If this edge was not picked in H then both triangles will not show up in H. This phenomenon may translate into a large variance in the number of triangles in H. To solve this problem, we identify the graph structure responsible for large variance. More concretely, we call s triangles that share the same edge an *s-tower*. For a carefully chosen number $s^* = s^*(p)$, one can show the following fact: If G has no s^*-tower, then the variance is small and the number of triangles in H is close to the expectation. If there is an s^*-tower, it is tall enough so that at least one floor survives (a floor is two edges that belong to the same triangle). In that case, in the second pass of A, the base of that tower is caught, and a triangle is detected.

Paper Organization. We proceed with the description of our second algorithm mentioned in Section 1.2. The proof of Theorem 3 follows in Section 3.

Algorithm $A_2(G, T, \rho(G))$

Output: 1 if a triangle is detected, 0 otherwise.
Pass 1
(a) Sample a set S of $4n/\rho(G)$ vertices, uniformly at random.
(b) Store all edges in the stream that touch the set S.
Pass 2 Check for every edge e if it completes a triangle with any of the stored edges. Output 1 iff such edge exists.

2 The Second Algorithm

Theorem 4. *Algorithm A_2 solves* DIST(2) *using $O(m/\rho(G))$ space in expectation.*

Proof. Let $Z \subset V$ be the set of vertices in G that belong to some triangle. In our notation, the size of Z is $\rho(G)$. The algorithm never fails if there are no triangles in G. Therefore let us consider the case where there are triangles in G.

The algorithm A_2 fails only if $S \cap Z = \emptyset$. Otherwise, S contains a vertex v that belongs to some triangle $\{v, u, w\}$, and in the first pass the algorithm stores all neighbors of v (and in particular the edges (v, u) and (v, w)). In the second pass the edge (u, w) will be considered and A_2 will detect the triangle. Let us bound the probability of $S \cap Z = \emptyset$. Let A_i be the event that the i^{th} vertex chosen to be in S doesn't belong to Z. It is easy to see that the A_i's are negatively correlated (as there is no replacement). For every i, $Pr[A_i] = 1 - \rho(G)/n$. Therefore,

$$Pr[S \cap Z = \emptyset] = Pr[A_1 \wedge A_2 \wedge \ldots A_{|S|}] \leq (1 - \rho(G)/n)^{4n/\rho(G)} \leq e^{-4}.$$

Now let us compute the expected number of edges stored by A_2. For the i^{th} vertex in S, let D_i be a random variable counting the degree of that vertex in G. Since the i^{th} vertex is a uniformly random vertex, $\mathbb{E}[D_i] = 2m/n$ (the average degree in G). The expected number of edges touching S is at most (using linearity of expectation)

$$\mathbb{E}\left[\sum_{i=1}^{|S|} D_i\right] = \sum_{i=1}^{|S|} \mathbb{E}\left[D_i\right] = (4n/\rho(G))(2m/n) = 8m/\rho(G).$$

This completes the Proof of Theorem 4.

3 Proof of Theorem 3

We denote by $Bin(n, p)$ the binomial random variable with parameters n and p, and expectation $\mu = np$. We shall use the following variant of the Chernoff bound, whose proof can be found in [9, p. 21]. Let $\varphi(x) = (1 + x)\ln(1 + x) - x$.

Theorem 5. *If $X \sim Bin(n,p)$ and $t \geq 0$ is some number, then*

$$Pr\big(X \geq \mu + t\big) \leq e^{-\mu\varphi(t/\mu)}, \qquad Pr\big(X \leq \mu - t\big) \leq e^{-\mu\varphi(-t/\mu)}.$$

The algorithm A always answers correctly if the graph G has no triangles. Therefore, it suffices to bound the error probability when the graph G has at least $T \geq 1$ triangles.

Let H be the graph in which each edge of the stream is included with probability p. Define two events:

$\mathcal{B}_1$ = "more than $5m'$ edges were stored in the first pass (causing the algorithm to output FAIL)", and
$\mathcal{B}_2$ = "H has no triangles, and no edge of the stream completes a triangle in H."

Then, $Pr[A \text{ fails}] \leq Pr[\mathcal{B}_1] + Pr[\mathcal{B}_2|\mathcal{B}_1^c] \leq Pr[\mathcal{B}_1] + Pr[\mathcal{B}_2]/Pr[\mathcal{B}_1^c]$.

To bound $Pr[\mathcal{B}_1]$ note that the number of edges stored by A is a binomial random variable with expectation $mp = m'$. In our case, $m' \geq 4$: we can assume w.l.o.g that $m \geq n/2$ (isolated vertices are never visible to the algorithm), and T always satisfies $T \leq n^3/6$, therefore $m' = 6m/T^{1/3} \geq 4$. Using Theorem 5, the probability of storing more than $5m'$ edges is at most $1/50$, hence $Pr[\mathcal{B}_1] \leq 1/50$. In turn,

$$Pr[A \text{ fails}] \leq \frac{1}{50} + \frac{50}{49} Pr[\mathcal{B}_2].$$

It suffices to show that $Pr[\mathcal{B}_2] \leq 0.3$, and then derive $Pr[A \text{ fails}] \leq 1/3$, as required. In what remains we prove $Pr[\mathcal{B}_2] \leq 0.3$.

To this end, we call s triangles that share the same edge an *s-tower*. Each pair of edges that belong to the same triangle is called a *floor* in the tower. Let $T_3 \geq T$ be the number of triangles in G. For $p = m'/m = 6/T^{1/3}$, let $\mu = p^3 T_3 = 216 T_3/T$ be the expected number of triangles in H and σ^2 the variance (σ is the standard deviation).

Lemma 1. *If G contains no tower with more than $T_3^{2/3}$ floors, then $\sigma \leq 110(T_3/T)^{5/6}$.*

To prove Lemma 1, we need the following claim.

Lemma 2. *Let G be a graph with T_3 triangles, having no tower with more than h floors. Let $\pi(G)$ be the number of pairs of triangles that share an edge. Then $\pi(G) \leq 3T_3 h/2$.*

Proof. Observe that every pair of triangles that share an edge belongs to exactly one tower: If the pair belongs to two towers, then the two triangles share two edges, but then they are the same triangle. Every pair belongs to at least one tower, since every such pair is a tower of height two. Therefore we can count the number of pairs sharing an edge, by counting the number of pairs of triangles in every tower. Let a_i be the number of towers with i floors. Using this notation, $\pi(G) = \sum_{i=2}^{h} a_i \binom{i}{2}$. Next observe that $\sum_{i=2}^{h} a_i i \leq 3T_3$. The sum counts the

number of triangles that belong to some tower, when every such triangle is accounted for at most three times (as it belongs to at most three different towers). Finally, we have

$$\pi(G) = \sum_{i=2}^{h} a_i \binom{i}{2} \leq \frac{1}{2}\sum_{i=2}^{h}(a_i i)\cdot i \leq \frac{1}{2}\sum_{i=2}^{h}(a_i i)\cdot h = \frac{h}{2}\sum_{i=2}^{h} a_i i \leq 3T_3 h/2.$$

Proof. (Lemma 1) Index the triangles in G by $1, 2, \ldots, T_3$. Let $\mathbf{1}_j$ be the indicator random variable which takes the value 1 if all three edges of triangle j belong to H.

$$\mathbb{E}[\mathbf{1}_j] = p^3, \qquad \mathrm{Var}[\mathbf{1}_j] = p^3(1-p^3) \leq p^3.$$

In these notations, σ^2 (the variance of the number of triangles in H) is given by

$$\sigma^2 = \sum_{i=1}^{T_3}\mathrm{Var}(\mathbf{1}_i) + \sum_{i<j}\mathrm{Cov}(\mathbf{1}_i,\mathbf{1}_j), \qquad \sum_{i=1}^{T_3}\mathrm{Var}(\mathbf{1}_i) \leq T_3 p^3 = \frac{216T_3}{T}.$$

For two triangles that share no edge, $\mathrm{Cov}(\mathbf{1}_i,\mathbf{1}_j) = \mathbb{E}[\mathbf{1}_i\mathbf{1}_j] - \mathbb{E}[\mathbf{1}_i]\mathbb{E}[\mathbf{1}_j] = p^6 - p^6 = 0$. Therefore we only need to consider triangles that share an edge. For every such pair, $\mathrm{Cov}(\mathbf{1}_i,\mathbf{1}_j) = p^5 - p^6 \leq p^5$. By Lemma 2 with $h = T^{2/3}$, there are at most $1.5T_3^{5/3}$ pairs of triangle that share an edge. Hence,

$$\sum_{i<j}\mathrm{Cov}(\mathbf{1}_i,\mathbf{1}_j) \leq 1.5T_3^{5/3}p^5 = \frac{1.5\cdot 6^5\cdot T_3^{5/3}}{T^{5/3}} \leq \left(\frac{278T_3}{T}\right)^{5/3}.$$

To summarize,

$$\sigma^2 \leq \frac{216T_3}{T} + \left(\frac{278T_3}{T}\right)^{5/3} \leq \left(\frac{282T_3}{T}\right)^{5/3}.$$

Taking the square root, we get the desired bound on σ.

Proposition 1. *Conditioned on G* not *having a tower with more than $T_3^{2/3}$ floors, $Pr[\mathcal{B}_2] \leq 0.26$.*

Proof. For a random variable X, with expectation μ and standard deviation σ, Chebychev's inequality implies $Pr[X=0] \leq \left(\frac{\sigma}{\mu}\right)^2$. The expected number of triangles in H is $\mu = 216T_3/T$. The standard deviation $\sigma \leq 110(T_3/T)^{5/6}$ (by Lemma 1). Therefore

$$Pr[\text{no triangles in } H] \leq \left(\frac{110}{216}\cdot\left(\frac{T}{T_3}\right)^{1/6}\right)^2 \leq 0.26.$$

Next we turn to the case where G contains a tower with at least $T_3^{2/3}$ floors.

Proposition 2. *Conditioned on G having a tower with at least $T_3^{2/3}$ floors, $Pr[\mathcal{B}_2] \leq 0.001$.*

Proof. Fix a tower with at least $T_3^{2/3}$ floors. Every floor belongs to H independently of the others with probability p^2. Therefore the expected number of floors that belong to H from that tower is

$$p^2 T_3^{2/3} = \left(\frac{6}{T^{1/3}}\right)^2 T_3^{2/3} = 36\left(\frac{T_3}{T}\right)^{2/3} \geq 36.$$

Using Chernoff's bound (second inequality of Theorem 5) with $\mu = 36$ and $t = 35$, we get

$$Pr[\text{no floor from the tower belongs to } H] \leq e^{-36\varphi(-36/35)} \leq e^{-30} \leq 0.001.$$

Finally, combining Propositions 1 and 2 we get $Pr[\mathcal{B}_2] \leq 0.26 + 0.001 < 0.3$. To complete the proof of Theorem 3, observe that the space complexity of A never exceeds $5m'$ which is $30m/T^{1/3}$.

References

1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: Proceedings of the 28th STOC, pp. 20–29 (1996)
2. Alon, M., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica 17, 209–223 (1997)
3. Bar-Yossef, Z., Kumar, R., Sivakumar, D.: Reductions in streaming algorithms, with an application to counting triangles in graphs. In: Proceedings of the 13th SODA, pp. 623–632 (2002)
4. Becchetti, L., Boldi, P., Castillo, C., Gionis, A.: Efficient semi-streaming algorithms for local triangle counting in massive graphs. In: Proceeding of the 14th ACM International Conference on Knowledge Discovery and Data Mining, pp. 16–24 (2008)
5. Buriol, L., Frahling, G., Leonardi, S., Marchetti-Spaccamela, A., Sohler, C.: Counting triangles in data streams. In: Proceedings of the Twenty-Fifth ACM Symposium on Principles of database Systems, pp. 253–262 (2006)
6. Chakrabarti, A., Cormode, G., Ranganath, K., McGregor, A.: Information Cost Tradeoffs for Augmented Index and Streaming Language Recognition. In: Proceedings of the 51st FOCS, pp. 387–396 (2010)
7. Eckmann, J., Moses, E.: Curvature of co-links uncovers hidden thematic layers in the world wide web. PNAS 99, 5825–5829 (2002)
8. Frank, O., Strauss, D.: Markov graphs. Journal of the American Statistical Association 81, 832–842 (1986)
9. Janson, S., Luczak, T., and Ruciński, A.: Random Graphs. Wiley (2000)
10. Jowhari, H., Ghodsi, M.: New streaming algorithms for counting triangles in graphs. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 710–716. Springer, Heidelberg (2005)
11. Kolountzakis, M., Miller, G., Peng, R., Tsourakakis, C.: Efficient Triangle Counting in Large Graphs via Degree-Based Vertex Partitioning. Internet Mathematics 8, 161–185 (2012)

12. Leskovec, J., Backstrom, L., Kumar, R., Tomkins, A.: Microscopic evolution of social networks. In: Proceeding of the 14th ACM International Conference on Knowledge Discovery and Data Mining, pp. 462–470 (2008)
13. Rinaldo, A., Fienberg, S., Zhou, Y.: On the geometry of discrete exponential families with application to exponential random graph models. Electronic Journal of Statistics 3, 446–484 (2009)
14. Tsourakakis, C., Drineas, P., Michelakis, E., Koutis, I., Faloutsos, C.: Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation. Social Network Analysis and Mining 1, 75–81 (2011)
15. Tsourakakis, C., Kang, U., Miller, G., Faloutsos, C.: DOULION: counting triangles in massive graphs with a coin. In: Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining, pp. 837–846 (2009)
16. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press (2004)
17. Vassilevska-Williams, V.: Multiplying matrices faster than Coppersmith-Winograd. In: Proceedings of the 44th STOC, pp. 887–898 (2012)

Online Checkpointing with Improved Worst-Case Guarantees

Karl Bringmann*, Benjamin Doerr, Adrian Neumann, and Jakub Sliacan

Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract. In the online checkpointing problem, the task is to continuously maintain a set of k checkpoints that allow to rewind an ongoing computation faster than by a full restart. The only operation allowed is to replace an old checkpoint by the current state. Our aim are checkpoint placement strategies that minimize rewinding cost, i.e., such that at all times T when requested to rewind to some time $t \leq T$ the number of computation steps that need to be redone to get to t from a checkpoint before t is as small as possible. In particular, we want that the closest checkpoint earlier than t is not further away from t than q_k times the ideal distance $T/(k+1)$, where q_k is a small constant.

Improving over earlier work showing $1 + 1/k \leq q_k \leq 2$, we show that q_k can be chosen asymptotically less than 2. We present algorithms with asymptotic discrepancy $q_k \leq 1.59 + o(1)$ valid for all k and $q_k \leq \ln(4) + o(1) \leq 1.39 + o(1)$ valid for k being a power of two. Experiments indicate the uniform bound $p_k \leq 1.7$ for all k. For small k, we show how to use a linear programming approach to compute good checkpointing algorithms. This gives discrepancies of less than 1.55 for all $k < 60$.

We prove the first lower bound that is asymptotically more than one, namely $q_k \geq 1.30 - o(1)$. We also show that optimal algorithms (yielding the infimum discrepancy) exist for all k.

1 Introduction

Checkpointing means storing intermediate states of a long sequence of computations. This allows reverting the system to a previous state much faster, since only the computations from the preceding checkpoint have to be redone. Checkpointing is one of the fundamental techniques in computer science. Classic results date back to the seventies [4], more recent topics are checkpointing in distributed [5], sensor network [8], or cloud [11] architectures.

Checkpointing usually involves a trade-off between the speed-up of reversions to previous states and the costs incurred by setting checkpoints (time, memory). Much of the classic literature (see [6] and the references therein) studies checkpointing with the focus of fault tolerance against immediately detectable faults. Consequently, only reversions to the most recent checkpoint are needed. However, setting a checkpoint can be expensive, because the whole system state

* Karl Bringmann is a recipient of the *Google Europe Fellowship in Randomized Algorithms*, and this research is supported in part by this Google Fellowship.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 255–266, 2013.

has to be copied to secondary memory. In such scenarios, the central question is how often to set a checkpoint such that the expected time spent on setting checkpoints and redoing computations from the last checkpoint is minimized (under a stochastic failure model and further, possibly time-dependent [10], assumptions on the cost of setting a checkpoint).

In this work, we will regard a checkpointing problem of a different nature. If not fault-tolerance of the system is the aim of checkpointing, then often the checkpoints can be kept in main memory. Applications of this type arise in data compression [2] and numerics [7,9]. In such scenarios, the cost of setting a checkpoint is small compared to the cost of the regular computation. Consequently, the memory used by the stored checkpoints is the bottleneck.

The first to provide a framework independent of a particular application were Ahlroth, Pottonen and Schumacher [1]. They do not make assumptions on which reversion will be requested, but simply investigate how checkpoints can be set in an online fashion such that at all times their distribution is balanced over the computation history.

They assume that the system is able to store up to k checkpoints (plus a free checkpoint at time 0). At any point in time, a checkpoint may be discarded and replaced by the current state as new checkpoint. Costs incurred by such a change are ignored. However, as it turns out, good checkpointing algorithms do not set checkpoints very often. For all algorithms discussed in the remainder of this paper, each checkpoint is changed only $O(\log T)$ times up to time T.

Each set of checkpoints, together with the current state and the state at time 0, partitions the time from the process start to the current time T into $k+1$ disjoint intervals. Clearly, without further problem-specific information, an ideal set of checkpoints would lead to all these intervals having identical length. Of course, this is not possible due to the restriction that new checkpoints can only be set on the current time. As discrepancy measure for a checkpointing algorithm, Ahlroth et al. mainly regard the *maximum gap ratio*, that is, the maximum ratio of the longest interval vs. the shortest interval (ignoring the last interval), over all current times T. They show that there is a simple algorithm achieving a discrepancy of two: Start with all checkpoints placed evenly, e.g., at times $1, \dots, k$. At an even time T, remove one of the checkpoints at an odd time and place it at T. This will lead to all checkpoints being at the even times $2, 4, \dots, 2k$ when $T = 2k$ is reached. Since these checkpoints form a scaled copy of the initial ones, we can continue in this fashion forever. It is easy to see that at all times, the intervals formed by neighboring checkpoints have at most two different lengths, the larger being twice the smaller. This shows the discrepancy of two.

Not much improvement is possible for general k as shown by the lower bound of $2^{1-1/\lceil (k+1)/2 \rceil} = 2(1 - o(1))$. For small values of k, namely $k = 2, 3, 4$, and 5, better upper bounds of approximately 1.414, 1.618, 1.755, and 1.755, respectively, were shown.

In this work, we shall regard a different, and, as we find, more natural discrepancy measure. Recall that the cost of reverting to a particular state is basically

the cost of redoing the computation from the preceding checkpoint to the desired point in time. Adopting a worst-case view on the time to revert to, our aim is to keep the length of the longest interval small (at all times). Note that with time progressing, the interval lengths necessarily grow. Hence a fair point of comparison is the length $T/(k+1)$ of a longest interval in the (at time T) optimal partition into equal length intervals. For this reason, we say that a checkpointing algorithm (using k checkpoints) has *maximum distance discrepancy* (or simply discrepancy) q if it places the checkpoints in such a way that at all times T, the longest interval has length at most $qT/(k+1)$. We denote by $q^*(k)$ the infimum discrepancy among all checkpointing algorithms using k checkpoints.

This discrepancy measure was suggested in [1]. There it was remarked that an upper bound of β for the gap-ratio implies an upper bound of $\beta(1+\frac{1}{k})$ for the maximum distance discrepancy. Furthermore, for all k an upper bound of 2 and a lower bound of $1+\frac{1}{k}$ is shown for $q^*(k)$. For $k = 2, 3, 4$, and 5, stronger upper bounds of $1.785, 1.789, 1.624$, and 1.565, respectively, were shown.

Our Results. In this work, we show that the optimal discrepancy $q^*(k)$ is asymptotically bounded away from both one and two by a constant. We present algorithms that achieve a discrepancy of $1.59 + O(1/k)$ for all k (Theorem 2), and a discrepancy of $\ln(4) + o(1) \leq 1.39 + o(1)$ for k being any power of two (Theorem 3). For small values of k, and this might be an interesting case in applications with memory-consuming states, we show superior bounds by suggesting a class of checkpointing algorithms and optimizing their parameters via a combination of exhaustive search and linear programming (Table 1). Experiments suggest $q^*(k) \leq 1.7$ for all k (Sect. 6). We complement these constructive results by a lower bound for $q^*(k)$ of $2 - \ln(2) - O(1/k) \geq 1.3 - O(1/k)$ (Theorem 6). We round off this work with a natural, but seemingly nontrivial result: We show that for each k there is indeed a checkpointing algorithm having discrepancy $q^*(k)$ (Theorem 4). In other words, the infimum in the definition of $q^*(k)$ can be replaced by a minimum.

Due to space restrictions, some proofs are omitted. They can be found in the full version of this paper [3].

2 Notation and Preliminaries

In our setting, we consider a long running computation during which we can choose to save the state at the current time T in a checkpoint, or delete a previously placed one. We assume that our storage can hold at most k checkpoints simultaneously, and that there are implicit checkpoints at time $t = 0$ and the current time. We disregard any costs for placing or maintaining checkpoints. Consequently, we may assume that we only delete a previous checkpoint when a new one is placed.

An *algorithm* for checkpoint placement can be described by two infinite sequences. First, the time points where new checkpoints are placed, i.e., a non-decreasing infinite sequence of reals $t_1 \leq t_2 \leq \ldots$ such that $\lim_{i\to\infty} t_i = \infty$, and

second, a rule that describes which old checkpoints to delete when a new one is installed, that is, an injective function $d : [k+1..\infty) \to \mathbb{N}$ satisfying $d_i < i$ for all $i \geq k+1$.

The algorithm A described by (t, d) will start with $t_1, \ldots, t_k$ as initial checkpoints and then for each $i \geq k+1$, at time t_i remove the checkpoint at t_{d_i} and set a new checkpoint at the current time t_i. We call the act of removing a checkpoint and placing a new one a *step* of A. Note that there is little point in setting the first k checkpoints to zero, so to make the following discrepancy measure meaningful, we shall always require that $t_k > 0$.

We call the set of checkpoints that exist at time T *active*. These, together with the two implicit checkpoints at times 0 and T, define a sequence of $k+1$ interval lengths $\mathcal{L}_T = (\ell_0, \ldots, \ell_k)$. The *discrepancy* $q(A, T)$ of an algorithm A at time $T \geq t_k$ is calculated as $q(A, T) := (k+1)\bar{\ell}_T/T$, where $\bar{\ell}_T = ||\mathcal{L}_T||_\infty$ denotes the length of the longest interval.

The discrepancy $\mathrm{Discr}(A)$ of an algorithm A then is the supremum over the discrepancy over all times T, i.e.,

$$\mathrm{Discr}(A) := \sup_{T \geq t_k} q(A, T).$$

Hence the discrepancy of an algorithm would be 1, if it always kept its checkpoints evenly distributed. Denote the infimum discrepancy of a checkpointing algorithm using k checkpoints by

$$q^*(k) := \inf_A \mathrm{Discr}(A),$$

where A runs over all algorithms using k checkpoints. We will see in Sect. 7 that algorithms achieving this discrepancy actually exist.

Note that we allow checkpointing algorithms to set checkpoints at continuous time points. One can convert any such algorithm to an algorithm with integral checkpoints by rounding all checkpointing times t_i down. This does not increase the discrepancy since $\lfloor t_i \rfloor - \lfloor t_{i-1} \rfloor \leq t_i - t_{i-1} + 1$, but with discrete time there are at most $\lfloor t_i \rfloor - \lfloor t_{i-1} \rfloor - 1$ steps to recompute in this interval.

In the definition of the discrepancy, the supremum is never attained at some T with $t_i < T < t_{i+1}$ for any i, further, at any time t_i it suffices to consider the two newly created intervals by deleting and storing a checkpoint. This is made precise in the following two lemmas.

Lemma 1. *In the definition of the discrepancy it suffices to consider times $T = t_i$ for all $i \geq k$, i.e., we have*

$$\mathrm{Discr}(A) = \sup_{i \geq k} q(A, t_i).$$

Lemma 2. *Let $i > k$ and let ℓ_1, ℓ_2 be the lengths of the two newly created intervals at time t_i due to the removal and the insertion of a checkpoint. Then*

$$\max\{q(A, t_{i-1}), q(A, t_i)\} = \max\{q(A, t_{i-1}), (k+1)\ell_1/t_i, (k+1)\ell_2/t_i\}.$$

Intuitively, both lemmas rely on the fact that unchanged intervals improve in discrepancy as time progresses. In Lemma 2, all intervals improve in discrepancy, except the two that change. In Lemma 1, the interval $[t_n, T]$, defined by the last checkpoint t_n and the current time T, has maximal discrepancy when we place a new checkpoint.

Often, it will be useful to use a different notation for the checkpoint removed in step i. Instead of the global index d, one can also use the index $p : [k+1..\infty) \to [1..k]$ among the active checkpoints, i.e.,

$$p_i = d_i - |\{j \in [i-1] \mid d_j < d_i\}|.$$

We call an algorithm $A = (t, p)$ *cyclic*, if the p_i are periodic with some period n, i.e., $p_i = p_{i+n}$ for all i, and after n steps A has transformed the intervals to a scaled version of themselves, that is, $\mathcal{L}_{t_{k+jn}} = \gamma^j \mathcal{L}_{t_k}$ for some $\gamma > 1$ and all $j \in \mathbb{N}$. We call γ the *scaling factor*. For a cyclic algorithm A, it suffices to fix the *pattern* of removals $P = (p_{k+1}, \ldots, p_{k+n})$ and the checkpoint positions $t_1, \ldots, t_k, t_{k+1}, \ldots, t_{k+n}$. Since our discrepancy notion is invariant under scaling, we can assume without loss of generality that $t_k = 1$ (and hence $t_{k+n} = \gamma$).

Since cyclic algorithms transform the starting position to a scaled copy of itself, it is easy to see that their discrepancy is given by the maximum over the discrepancies during one period, i.e., for cyclic algorithms A with period n we have

$$\mathrm{Discr}(A) = \max_{k<i\le k+n} q(A, t_i).$$

This makes this class of algorithms easy to analyze.

3 Introductory Example—A Simple Bound for $k = 3$

For the case of $k = 3$ there is a very simple algorithm, Simple, with a discrepancy of $4/\phi^2 \approx 1.53$, where $\phi = (\sqrt{5}+1)/2$ is the golden ratio. We use it to familiarize ourselves with the notation from Sect. 2. The algorithm is cyclic with a pattern of length one. We prove the following theorem.

Theorem 1. *For $k = 3$ there is a cyclic algorithm* Simple *with period length one and*

$$\mathrm{Discr}(\textsc{Simple}) = 4/\phi^2.$$

Proof. We fix the pattern to be $P = (1)$, that is, algorithm Simple always removes the oldest checkpoint. For this simple pattern it is easy to calculate the discrepancy depending on the scaling factor γ. Since the intervals need to be a scaled copy of themselves after just one step and we can fix $t_3 = 1$, we know immediately that

$$t_1 = \gamma^{-2},\ t_2 = \gamma^{-1},\ t_3 = 1,\ t_4 = \gamma,$$

and hence the discrepancy is determined by

$$4 \cdot \max\{t_1/t_3, (t_2 - t_1)/t_3, (t_3 - t_2)/t_3\} = 4 \cdot \max\left\{\gamma^{-2}, (\gamma-1)/\gamma^2, (\gamma-1)/\gamma\right\}.$$

A simple calculation shows this maximum to be minimal at $\gamma = \phi$.

Hence for $k = 3$ the algorithm with pattern (1) and checkpoint positions $t_1 = 1/\phi^2$, $t_2 = 1/\phi$, $t_3 = 1$, and $t_4 = \phi$ has discrepancy $4/\phi^2 \approx 1.53$. □

4 An Upper Bound for Large k

In this section we present an algorithm, Linear, with a discrepancy of roughly 1.59 for large k. This improves upon the asymptotic bound of 2 from [1]. Moreover, Linear is easily implemented for all k.

Like the algorithm Simple of the previous section, the algorithm Linear is cyclic. It has a simple pattern of length k. The pattern is just $(1, \ldots, k)$, that is, at the i-th step of a period Linear deletes the i-th active checkpoint. Overall, during one period Linear removes all checkpoints at times t_i with odd index i, as shown in Fig. 1.

This removal pattern is identical to the one of Powers-Of-Two algorithm from [1]. However, that algorithm starts with a uniform checkpoint distribution where removing any checkpoint doubles the maximum interval. This leads to an asymptotic discrepancy of two. In contrast, Linear places checkpoints on a polynomial. For $i \in [1, 2k]$ we set $t_i = (i/k)^\alpha$, where α is a constant. In the analysis we optimize the choice of α and set $\alpha := 1.302$. We show the following theorem.

Theorem 2. *Algorithm* Linear *has a discrepancy of at most* $1.586 + O(k^{-1})$.

The proof of the above theorem is a straightforward calculation of the lengths of the newly created intervals at each step, similar to the previous section, together with some analytic estimations of the resulting discrepancies.

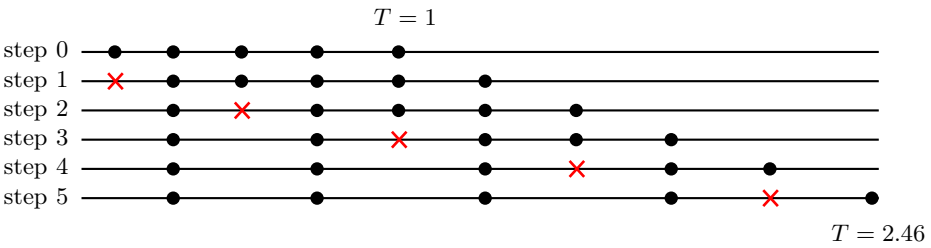


Fig. 1. One period of the algorithm Linear from Sect. 4 for $k = 5$. After one period all intervals are scaled by the same factor.

5 An Improved Upper Bound for Large k

In this section we present the algorithm Binary that yields a discrepancy of roughly $\ln(4) \approx 1.39$ for large k. Compared to the algorithm Linear from the last section, Binary has a considerably better discrepancy at the price of a more involved analysis, and it only works for k being a power of two.

Theorem 3. *For $k \geq 8$ being any power of 2, the algorithm* BINARY *has discrepancy*

$$\mathrm{Discr}(\textsc{Binary}) \leq \ln(4) + 0.05/\lg(k/4) + O\left(k^{-1}\right)$$

Here and in the remainder of this paper, let 'lg' denote the binary and 'ln' the natural logarithm. Note that the term $O(1/k)$ quickly tends to 0, whereas the $\Theta(1/\lg(k/4))$ term is small due to the constant 0.05. Hence, this discrepancy is close to $\ln(4)$ already for moderate k. Also note that $\ln(4)$ is by less than 0.1 larger than our lower bound from Sect. 8, leaving room for less than a 6% improvement over algorithm BINARY for large k.

5.1 The Algorithm BINARY

The initial checkpoints $t_1, \ldots, t_k$ satisfy the equation

$$t_i = \alpha t_{i/2} \tag{1}$$

for each even $1 \leq i \leq k$ and some $\alpha = \alpha(k) \geq 2$. Precisely, we set

$$\alpha := 2^{1+\frac{\lg(\sqrt{2}/\ln 4)}{\lg(k/4)}} \approx 2^{1+\frac{0.029}{\lg(k/4)}}.$$

However, the usefulness of this expression becomes clear only in the analysis.

During one period we delete all odd checkpoints $t_1, t_3, \ldots, t_{k-1}$ and insert

$$t_{k+i} := \alpha t_{k/2+i}, \tag{2}$$

for $1 \leq i \leq k/2$. Then after one period we end up with the checkpoints

$$\begin{aligned} &(t_2, t_4, \ldots, t_{k-2} \ , t_k \ , \ t_{k+1} \ , t_{k+2} \ , \ldots, \ t_{k+k/2}) \\ = \alpha \cdot &(t_1, t_2, \ldots, t_{k/2-1}, t_{k/2}, \ t_{k/2+1}, t_{k/2+2}, \ \ldots, \ t_{k/2+k/2}) = \alpha(t_1, t_2, \ldots, t_k), \end{aligned}$$

which proves cyclicity. Note that (1) and (2) allow us to compute all t_i from the values $t_{k/2+1}, \ldots, t_k$, however, we still have some freedom to choose the latter values. Without loss of generality we can set $t_k := 1$, then $t_{k/2} = \alpha^{-1}$. In between these two values, we interpolate $\lg t_i$ linearly, i.e., we set for $i \in (k/2, k]$

$$t_i := \alpha^{2i/k-2}, \tag{3}$$

completing the definition of the t_i. Note that(3) also works for $i = k$ and $i = k/2$.

In iteration $1 \leq i \leq k/2$ we insert the checkpoint t_{k+i} and remove the checkpoint $t_{d(i+k)}$, defined as follows. For $m \in \mathbb{N} = \mathbb{N}_{\geq 1}$ let $2^{e(m)}$ be the largest power of 2 that divides m. We define $S\colon \mathbb{N} \to \mathbb{N}, S(m) := m/2^{e(m)}$. Note that $S(m)$ is an odd integer. Using this definition, we set

$$d(k+i) := S(i + k/2) \tag{4}$$

finishing the definition of the algorithm BINARY. If we write this down as a pattern, then we have $p_i = 1 + k/(2^{1+e(i)})$ for $1 \leq i < k/2$ and $p_{k/2} = 1$. For intuition as to the behavior of this pattern, see the example in Fig. 2. It is not hard to see that with d as defined above, we indeed delete all odd checkpoints $t_1, t_3, \ldots, t_{k-1}$ during one period, in the full version of this paper we include a formal proof of this.

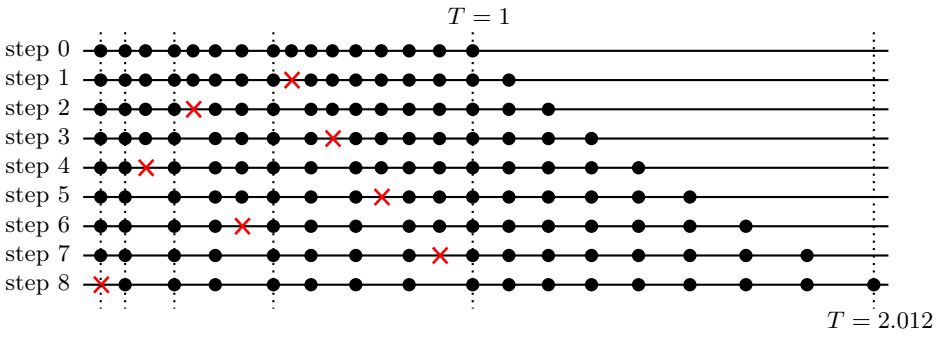


Fig. 2. One period of the algorithm BINARY for $k = 16$. Note that, recursively, checkpoints are removed twice as often from the right half of the initial setting as from the second quarter.

5.2 Discrepancy Analysis

We now bound the largest discrepancy encountered during one period, i.e.,

$$\mathrm{Discr}(\textsc{Binary}) = \max_{1\le i\le k/2} q(\textsc{Binary}, t_{i+k}) = (k+1) \max_{1\le i\le k/2} \overline{\ell}_{t_{i+k}}/t_{i+k}.$$

By Lemma 2, we only have to consider intervals newly created by insertion and deletion at any step. We do this exemplarily for the intervals from insertion.

Intervals from Insertion: We compute the discrepancy of the interval newly added at time t_{i+k}, $1 \le i \le k/2$. Its length is $t_{i+k} - t_{i+k-1}$, so its discrepancy is

$$\begin{aligned}(k+1)\frac{t_{i+k} - t_{i+k-1}}{t_{i+k}} &= (k+1)\left(1 - \frac{t_{i+k-1}}{t_{i+k}}\right) \\ &= (k+1)\left(1 - \frac{t_{i+k/2-1}}{t_{i+k/2}}\right) \overset{(3)}{=} (k+1)\left(1 - \alpha^{-2/k}\right),\end{aligned}$$

where the second equality holds because of (2) if $i > 1$ or (1) if $i = 1$.

Using $e^x \ge 1 + x$ for $x \in \mathbb{R}$ yields a bound on the discrepancy of

$$(k+1)\frac{t_{i+k} - t_{i+k-1}}{t_{i+k}} \le (k+1)\ln(\alpha)\frac{2}{k} = \ln(\alpha^2) + O(k^{-1}).$$

Since we choose $\alpha \approx 2$, we obtain a discrepancy of roughly $\ln(4)$, and the error term can easily be seen to be bounded by $0.05/\ln(k/4) + O(k^{-1})$.

6 Upper Bounds via Combinatorial Optimization

In this section we show how to find upper bounds on the optimal discrepancy $q^*(k)$ for fixed k. We do so by constructing cyclic algorithms using exhaustive enumeration of all short patterns in the case of very small k or randomized

local search on the patterns for larger k, combined with linear programming to optimize the checkpoint positions. This yields good algorithms as summarized in Table 1. In the following we describe our algorithmic approach.

First we describe how to find a nearly optimal cyclic algorithm given a pattern P and a scaling factor γ, i.e., how to optimize the checkpoint positions. To do so, we construct a linear program that is feasible if a cyclic algorithm with discrepancy λ and scaling factor γ exists. We use three kinds of constraints: We fix the ordering of the checkpoints, enforce that the i-th active checkpoint after one period is a factor γ larger than the i-th initial checkpoint, and upper bound the discrepancy of each interval during the period by λ. We then use binary search to optimize λ.

Lemma 3. *For a fixed pattern P of length n and scaling factor γ, let $q^* = \inf_A \mathrm{Discr}(A)$ be the optimal discrepancy among algorithms A using P and γ. Then finding an algorithm with discrepancy at most $q^* + \epsilon$ reduces to solving $O(\log \epsilon^{-1})$ linear feasibility problems with $O(nk)$ inequalities and $k+n$ variables.*

To find good algorithms without a fixed γ, we need the following lemma which is proved in the full paper.

Lemma 4. *A cyclic algorithm with k checkpoints, discrepancy $\lambda < k$, and a period length of n can have scaling factor at most $\gamma \leq (1 - \lambda/(k+1))^{-n}$.*

For any given pattern length n, Lemma 4 yields an upper bound on γ, while a trivial lower bound is given by $\gamma > 1$. Now, for any given pattern P we optimize over γ using a linear search with a small step size over the possible values for γ. For each tested γ, we optimize over the checkpoint positions using the linear programming approach described above.

Results: We ran experiments that exhaustively try all patterns up to length k for $k \in [3, 7]$. For $k = 8$ we stopped after examining all patterns of length 7. For larger k we used a randomized local search to find good patterns. The upper bounds we found are summarized in Table 1.

We cannot prove (near) optimality for these algorithms, because we do not know whether short patterns (or any finite patterns) are sufficient, and whether the discrepancy behaves smoothly with γ and a linear search can find a nearly optimal scaling factor. However, we tried all patterns of length $2k$ for $k \in [3, 4, 5]$ and found no better algorithm. Moreover, decreasing the step size in the linear search for γ only yielded small improvements, suggesting that λ is continuous in γ. This suggests that the bounds in Table 1 are indeed close to optimal.

Note that we can combine the results presented in Table 1 with experimental results for the algorithm Linear (Theorem 2) to read off a global upper bound of $q^*(k) \leq 1.7$ for the optimal discrepancy for *any* k.

For a fixed pattern the method is efficient enough to find good checkpoint positions for much larger k. For $k \leq 1000$ we experimentally compared the algorithm Linear of Sect. 4 with algorithms found for its pattern $(1, \ldots, k-1)$. The experiments show that for $k = 1000$ Linear is within 4.5% of the optimized

Table 1. Upper bounds for different k. For $k < 8$ all patterns up to length k were tried. For $k = 8$ all patterns up to length 7 were tried. For larger k, patterns were found via randomized local search.

k	3	4	5	6	7	8	9	10	15	20	30	50	100
Discr.	1.529	1.541	1.472	1.498	1.499	1.499	1.488	1.492	1.466	1.457	1.466	1.481	1.484

bounds. For the algorithm BINARY of Sect. 5, this comparison is even more favorable. For $k = 1024$ the algorithm places its checkpoints so well that the optimization procedure improves discrepancy only by 1.9%.

7 Existence of Optimal Algorithms

In this section, we prove that optimal algorithms for the checkpointing problem exist, i.e., that there is an algorithm having discrepancy equal to the infimum discrepancy $q^*(k) := \inf_A \mathrm{Discr}(A)$ among all algorithms for k checkpoints.

Theorem 4. *For each k there exists a checkpointing algorithm A for k checkpoints with $\mathrm{Discr}(A) = q^*(k)$, i.e., there is an optimal checkpointing algorithm.*

This a non-trivial statement. From the proof of this statement, we gain additional insight in the behavior of good algorithms. In particular, we show that we can assume without increasing discrepancy that for all i the i-th checkpoint is set by a factor of at least $(1 + 1/k)^{\Theta(i)}$ later than the first checkpoint.

Theorem 5. *Let $A = (t, d)$ be a checkpointing algorithm with $\mathrm{Discr}(A) < k - 1$. Then there is an algorithm $A' = (t', d')$ with the same starting position such that (i) $\mathrm{Discr}(A') \le \mathrm{Discr}(A)$ and (ii) $t'_{i+3} \ge (1 + 1/k) \cdot t'_i$ for all $i \ge k$.*

8 Lower Bound

In this section, we prove a lower bound on the discrepancy of all checkpointing algorithms. For large k we get a lower bound of roughly 1.3, so we have a lower bound that is asymptotically larger than the trivial bound of 1. Moreover, it shows that algorithm BINARY from Sect. 5 is nearly optimal, as for large k the presented lower bound is within 6% of the discrepancy of BINARY.

Theorem 6. *All checkpointing algorithms with k checkpoints have a discrepancy of at least $2 - \ln 2 - O(k^{-1}) \ge 1.306 - O(k^{-1})$.*

We present a sketch of the proof of the above theorem in the remainder of this section.

Let $A = (t, d)$ be an arbitrary checkpointing algorithm and let $q' := \mathrm{Discr}(A)$ be its discrepancy. For convenience, we define $q = kq'/(k + 1)$ and bound q.

Since $q < q'$ this suffices to show a lower bound for the discrepancy of A. For technical reasons we add a *gratis checkpoint* at time t_k that must not be removed by A. That is, even after the removal of the original checkpoint at t_k, there still is the gratis checkpoint active at t_k. Clearly, this can only improve the discrepancy. We analyze A from time t_k until it deleted $k/(2q)$ of the initial checkpoints[1]. More formally, we let t' be the minimal time at which the number of active checkpoints of A contained in $[0, t_k]$ is $k - k/(2q)$. Note that we might have $t' = \infty$, if the checkpointing algorithm A never deletes $k/(2q)$ points from $[0, t_k]$. However, in this case its discrepancy is lower bounded by 1.5, since, for any $i > k$, there are at most $k - k/(2q)$ checkpoints available in the interval $(t_k, t_i]$.

Lemma 5. *If* $t' = \infty$, *then* $\mathrm{Discr}(A) \geq 1.5$.

Hence, in the following we can assume that $t' < \infty$. We partition the intervals that exist at time t' into three types:

1. Intervals existing both at time t_k and t'. These are contained in $[0, t_k]$.
2. Intervals that are contained in $[0, t_k]$, but did not exist at time t_k. These were created by the removal of some checkpoint in $[0, t_k]$ after time t_k.
3. Intervals contained in $[t_k, t']$.

Note that we need the gratis checkpoint at t_k in order for these definitions to make sense, as otherwise there could be an interval overlapping t_k.

Let $\mathcal{L}_i$ denote the set of intervals of type i for $i \in \{1, 2, 3\}$, and set $k_i := |\mathcal{L}_i|$. Let $\mathcal{L}_2 = \{I_1, \ldots, I_{k_2}\}$, where the intervals are ordered by their creation times $\tau_1 \leq \ldots \leq \tau_{k_2}$. Since each interval in $\mathcal{L}_2$ contains at least one deleted point we have $k_2 \leq k/(2q)$, and we set $m := \frac{k}{2q} - k_2$. Then m counts the number of deleted checkpoints in $[0, t_k]$ that did not create an interval in $\mathcal{L}_2$, but some strict sub-interval of an interval in $\mathcal{L}_2$.

Since the intervals in $\mathcal{L}_1$ exist at time t_k, we can bound their length as follows.

Lemma 6. *The length of any interval in* $\mathcal{L}_1$ *is at most* qt_k/k.

The creation time τ_i of the i-th interval in $\mathcal{L}_2$ cannot be too late, as otherwise there would not be sufficiently many checkpoints in $[t_k, \tau_i]$ to guarantee discrepancy q. That allows to bound the length of these intervals.

Lemma 7. *The length of any interval* $I_i \in \mathcal{L}_2$ *is at most*

$$|I_i| \leq \frac{t_k}{k/q - m - i}.$$

Furthermore, we need a relation between k_1, k, m, and q.

[1] To be precise we should round $\frac{k}{2q}$ to one of its nearest integers. When doing so, all calculations in the remainder of this section go through as they are; this only slightly increases the hidden constant in the error term $O(k^{-1})$.

Lemma 8. *We have* $k_1 = k + m - k/q + 1$.

Now we use our bounds on the length of intervals from $\mathcal{L}_1$ and $\mathcal{L}_2$ to find a bound on q. Note that the intervals in $\mathcal{L}_1$ and $\mathcal{L}_2$ partition $[0, t_k]$, so that

$$t_k = \sum_{I \in \mathcal{L}_1} |I| + \sum_{I' \in \mathcal{L}_2} |I'|.$$

Using Lemmas 6 and 7, we obtain

$$t_k \leq k_1 \frac{q t_k}{k} + \sum_{i=1}^{k_2} \frac{t_k}{k/q - m - i}.$$

Plugging in Lemma 8 and simplifying yields

$$q \geq 2 - m\frac{p}{k} - O(k^{-1}) - H_{k/q-m-1} + H_{k/(2q)-1},$$

where H_n is the nth harmonic number. Now, observing $m\frac{q}{k} + H_{k/q-m-1} \leq H_{k/q-1}$ and using the asymptotics of H_n, we get the desired bound $q \geq 2 - \ln(2) - O(k^{-1})$.

References

1. Ahlroth, L., Pottonen, O., Schumacher, A.: Approximately uniform online checkpointing with bounded memory. Algorithmica (to appear, 2013)
2. Bern, M.W., Greene, D.H., Raghunathan, A., Sudan, M.: On-line algorithms for locating checkpoints. Algorithmica 11(1), 33–52 (1994)
3. Bringmann, K., Doerr, B., Neumann, A., Sliacan, J.: Online checkpointing with improved worst-case guarantees. CoRR, abs/1302.4216 (2013)
4. Chandy, K.M., Ramamoorthy, C.V.: Rollback and recovery strategies for computer programs. IEEE Transactions on Computers C-21, 546–556 (1972)
5. Elnozahy, E.N.M., Alvisi, L., Wang, Y.-M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys 34(3), 375–408 (2002)
6. Gelenbe, E.: On the optimum checkpoint interval. Journal of the ACM 26(2), 259–270 (1979)
7. Heuveline, V., Walther, A.: Online checkpointing for parallel adjoint computation in PDEs: Application to goal-oriented adaptivity and flow control. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 689–699. Springer, Heidelberg (2006)
8. Österlind, F., Dunkels, A., Voigt, T., Tsiftes, N., Eriksson, J., Finne, N.: Sensornet checkpointing: Enabling repeatability in testbeds and realism in simulations. In: Roedig, U., Sreenan, C.J. (eds.) EWSN 2009. LNCS, vol. 5432, pp. 343–357. Springer, Heidelberg (2009)
9. Stumm, P., Walther, A.: New algorithms for optimal online checkpointing. SIAM Journal on Scientific Computing 32(2), 836–854 (2010)
10. Toueg, S., Babaoglu, Ö.: On the optimum checkpoint selection problem. SIAM Journal on Computing 13(3), 630–649 (1984)
11. Yi, S., Kondo, D., Andrzejak, A.: Reducing costs of spot instances via checkpointing in the Amazon elastic compute cloud. In: IEEE 3rd International Conference on Cloud Computing (CLOUD 2010), pp. 236–243 (2010)

Exact and Efficient Generation of Geometric Random Variates and Random Graphs

Karl Bringmann[1,⋆] and Tobias Friedrich[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] Friedrich-Schiller-Universität Jena, Germany

Abstract. The standard algorithm for fast generation of Erdős-Rényi random graphs only works in the Real RAM model. The critical point is the generation of geometric random variates $\mathrm{Geo}(p)$, for which there is no algorithm that is both exact and efficient in any bounded precision machine model. For a RAM model with word size $w = \Omega(\log\log(1/p))$, we show that this is possible and present an exact algorithm for sampling $\mathrm{Geo}(p)$ in optimal expected time $\mathcal{O}(1 + \log(1/p)/w)$. We also give an exact algorithm for sampling $\min\{n, \mathrm{Geo}(p)\}$ in optimal expected time $\mathcal{O}(1 + \log(\min\{1/p, n\})/w)$. This yields a new exact algorithm for sampling Erdős-Rényi and Chung-Lu random graphs of n vertices and m (expected) edges in optimal expected runtime $\mathcal{O}(n + m)$ on a RAM with word size $w = \Theta(\log n)$.

1 Introduction

Random Graph Generation. A large fraction of empirical research on graph algorithms is performed on random graphs. Random graph generation is also commonly used for simulating networking protocols on the Internet topology and the spread of epidemics (or rumors) on social networks (e.g. [16]) It is also an important tool in real world applications such as detecting motifs in biological networks (e.g. [21]). We focus on homogenous and inhomogenous random graphs and consider Erdős-Rényi [9] and Chung-Lu graphs [5]. The key ingredient for generating such graphs with n vertices faster than the obvious $\Theta(n^2)$ algorithm is an efficient algorithm for exact sampling of geometric random variates.

Efficient Random Variate Generation. Non-uniform random variates are typically generated from random variates that are uniformly distributed on $[0, 1]$. With the introduction of Intel's Ivy Bridge microarchitecture with built-in hardware digital random number generation, we can assume that we have fast access to a stream of high quality random bits. However, most non-uniform random variate generation algorithms [8, 26] assume a Real RAM, which can manipulate real numbers. This assumption is highly problematic as real numbers are

⋆ Karl Bringmann is a recipient of the *Google Europe Fellowship in Randomized Algorithms*, and this research is supported in part by this Google Fellowship.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 267–278, 2013.

infinite objects and all physical computers can only handle finite portions of these objects. Typical implementations, with e.g. double floating point precision, are efficient, but *not exact* (e.g. in C++11); that is, some outcomes might not be reachable and others might become more likely than they should.

Exact Random Variate Generation. Knuth and Yao [18] initiated the study of exact nonuniform random number generation. They discuss the power of various restricted machine models. Several authors [10, 11, 18] provided additional results along these lines, but only presented efficient algorithms for single distributions. There also exist implementations of exact and efficient random number generators for exponential and normal distributions [15]. For parameterized families of distributions such as the geometric distribution, one can study the expected asymptotic runtime in the parameter. However, in this regard there are no exact and efficient algorithms known on any bounded precision machine model. For sampling $\mathrm{Geo}(p)$, the trivial algorithm of repeatedly sampling a coin with Bernoulli distribution $\mathrm{Ber}(p)$ until it falls heads up has expected runtime $\mathcal{O}(1/p)$, which is *not efficient* for p close to 0.

Exact and Efficient Random Variate Generation. Our aim is the design of exact and efficient algorithms for random variate generation. We show that this is possible in many cases and give a particularly fast algorithm for geometric random variates. This allows exact and efficient generation of Erdős-Rényi and Chung-Lu random graphs. It also allows exact and efficient generation of very large non-uniform random variates (e.g. for cryptographic applications [13]), which has been open so far.

Related Work on Random Graph Generation. There is a large body of work on generating random regular graphs (e.g. [17]), graphs with a prescribed degree distribution (e.g. [3]), and graphs with a prescribed joint degree distribution (e.g. [23]). All these algorithms converge to the desired distribution for $n \to \infty$. Note that this typically implies for finite n that only an approximation of the true distribution is reached.

The most studied random graph model is certainly the Erdős-Rényi [9] random graph $\mathcal{G}(n,p)$, where each edge of a graph of n vertices is present independently and uniformly with probability $p \in [0,1]$. Many experimental papers use algorithms with runtime $\Theta(n^2)$ to draw from $\mathcal{G}(n,p)$. The reason for this is probably that most graph algorithm software libraries such as JUNG, LEDA, BGL, and JDSL also do not contain efficient random graph generators. However, there are several algorithms which can sample from $\mathcal{G}(n,p)$ in expected time $\mathcal{O}(m+n)$ on a Real RAM, where $m = \Theta(pn^2)$ is the expected number of edges [1, 20]. This is done by using the fact that in an ordered list of all $\Theta(n^2)$ pairs of vertices the distance between two consecutive edges is geometrically distributed. The resulting distribution is *not exact* if the algorithm is run on a physical computer, which can only handle bounded precision, as it ignores the bias introduced by fixed-length number representations. The available implementation in the library NetworkX [14] therefore also does not return the desired distribution exactly. It is not obvious how to get an exact implementation even by using

algebraic real numbers [19] and/or some high accuracy floating-point representation. The problem of sampling $\mathcal{G}(n,p)$ on a bounded precision model has been studied by Blanca and Mihail [2]. They showed how to achieve an approximation of the desired distribution efficiently. Our aim is an *exact and efficient* generation on a bounded precision model instead.

Generating Random Geometric Distributions. As discussed above, all previous algorithms to sample a geometric random variate which are efficient on a Real RAM become inexact when implemented on a bounded precision machine. We assume the more realistic model of a Word RAM with word size w that can sample a random word in constant time; for a definition of the machine models see Section 2. We first observe the following lower bound for any exact sampling algorithm.

Theorem 1. *On a RAM with word size w, any algorithm sampling a geometric random variate* $\mathrm{Geo}(p)$ *with parameter $p \in (0,1)$ needs at least expected runtime $\Omega(1 + \log(1/p)/w)$.*

Theorem 1 follows from Lemma 1, which shows that the expected output size is $\Omega(\log(1/p))$ bits. To get a first upper bound we translate the well-known inversion method (typically used on a Real RAM [8]) to our bounded precision model by using multi-precision arithmetic, obtaining the following result.

Theorem 2. *On a RAM with word size $w = \Omega(\log\log(1/p))$, a geometric random variate* $\mathrm{Geo}(p)$ *with parameter $p \in (0,1)$ can be sampled in expected runtime $\mathcal{O}(1 + \log(1/p)\,\mathrm{poly}\log\log(1/p)/w)$.*

To the best of our knowledge, this observation is not discussed in the literature so far. However, as the following Theorem 3 is strictly stronger, we defer the presentation of the inversion method and Theorem 2 to the full version of this paper. It not only applies to geometric distributions, but to all distributions where the inverse of the cumulative distribution is efficiently computable on a Word RAM. The assumption on w is needed to handle pointers to an array as large as the expected output size in constant time. This result is independent of the rest of the paper and demonstrates that the classical inversion method *does not give an optimal runtime* matching Theorem 1, since this algorithm, as well as many other approaches, *does not avoid taking logarithms*. Note that it is a long-standing open problem in analytic number theory and computational complexity whether the logarithm can be computed in linear time.

Our aim is a Word RAM algorithm which returns the exact geometric distribution in optimal runtime. In Section 3 we give a simple algorithm for this and prove the following theorem. Note that our algorithm also works for *bitstreams* p, see Section 2.

Theorem 3. *On a RAM with word size $w = \Omega(\log\log(1/p))$, a geometric random variate* $\mathrm{Geo}(p)$ *with parameter $p \in (0,1)$ can be sampled in expected runtime $\mathcal{O}(1 + \log(1/p)/w)$, which is optimal.*

Observe that, as a sample of a geometric random variate can be arbitrarily large, the aforementioned sampling algorithm cannot work in bounded worst-case time

or space. Also note that on a parallel machine with P Word RAM processors the runtime decreases to $\mathcal{O}(1 + \log(1/p)/(wP))$.

Generating Bounded Random Geometric Distributions. In the full version of this paper we extend this to sampling bounded geometric random distributions $\mathrm{Geo}(p, n) = \min\{n, \mathrm{Geo}(p)\}$ and observe the following lower bound.

Theorem 4. *On a RAM with word size w, any algorithm sampling a bounded geometric random variate $\mathrm{Geo}(p, n) = \min\{n, \mathrm{Geo}(p)\}$ with parameters $n \in \mathbb{N}$ and $p \in (0, 1)$ needs at least expected runtime $\Omega(1 + \log(\min\{1/p, n\})/w)$.*

We present an algorithm which achieves this optimal runtime bound and prove the following theorem.

Theorem 5. *On a RAM with word size $w = \Omega(\log\log(1/p))$, a bounded geometric random variate $\mathrm{Geo}(p, n) = \min\{n, \mathrm{Geo}(p)\}$ with parameters $n \in \mathbb{N}$ and $p \in (0, 1)$ can be sampled in expected runtime $\mathcal{O}(1 + \log(\min\{1/p, n\})/w)$, which is optimal.*

If p is a rational number with numerator and denominator fitting in $\mathcal{O}(1)$ words, then this algorithm needs $\mathcal{O}(n)$ space in the worst case.

If p is a bitstream, we cannot bound the worst-case space usage of a sampling algorithm for $\mathrm{Geo}(n, p)$ in general. However, if p is a rational with numerator and denominator fitting in a constant number of words of the Word RAM, Theorem 5 shows that this is indeed possible.

Random Graph Generation. We believe our new exact and efficient sampling algorithms for bounded and unbounded geometric distributions are of independent interest, but also present in Section 4 one particular application, which is the generation of random graphs. For generating graphs with n vertices it is natural to assume $w = \Omega(\log n)$.

Theorem 6. *On a RAM with word size $w = \Omega(\log n)$, the random graph $\mathcal{G}(n, p)$ can be sampled in expected time $\Theta(n + m)$, where $m = \Theta(pn^2)$ is the expected number of edges. This is optimal if $w = \mathcal{O}(\log n)$. If p is a rational number with numerator and denominator fitting in $\mathcal{O}(1)$ words, then the worst-case space complexity of the algorithm is asymptotically equivalent to the size of the output graph, which is optimal.*

A similar algorithm achieves optimal runtime for the more general Chung-Lu random graphs $\mathcal{G}(n, W)$ [5], generating a random graph with a given sequence of expected degrees.

Theorem 7. *Let $W = (W_1, \ldots, W_n)$ be rationals with common denominator, where each numerator and the common denominator fit in $\mathcal{O}(1)$ words. Then on a RAM with word size $w = \Omega(\log n)$, the random graph $\mathcal{G}(n, W)$ can be sampled in expected time $\Theta(n + m)$, where $m = \Theta(\sum_{i=1}^{n} W_i)$ is the expected number of edges. This is optimal if $w = \mathcal{O}(\log n)$. The worst-case space complexity of the algorithm is asymptotically equivalent to the size of the output graph, which is optimal.*

Both theorems follow from plugging our algorithm for sampling geometric random variables into the best algorithm known for sampling the respective graph class.

Note that due to tight space constraints many proofs, in particular for Theorems 2, 4, 5, and 7, are defered to the full version of this paper.

2 Preliminaries

Machine Models. We discuss two variants of random access machines (RAMs). Both are abstract computational machine models with an arbitrary number of registers that can be indirectly addressed. In the classic RAM, each register can contain an arbitrarily large natural number $\mathbb{N}_0$ and all basic mathematical functions can be performed in constant time.

The two models relevant for this paper are the Real RAM, which allows computation even with real numbers, and the Word RAM, which only allows computation with bounded precision. In addition to the standard definitions, we assume that a uniform random number can be sampled in constant time. As we are dealing with randomized algorithms and a sample of a geometric random variate can be arbitrarily large, we also allow potentially unbounded space usage[1].

The *Real RAM* is the main model of computability in computational geometry and is also used in numerical analysis. Here, each register can contain a real number in the mathematical sense. All basic mathematical functions including the logarithm of a real number can be computed in constant time. The disadvantage of the model is that real numbers are infinite objects and all physical computers can only handle finite portions of these objects.

The *Word RAM* is a more realistic model of computation. It is parameterized by a parameter w which determines the word length. The registers are called words and contain integers in the range $\{0, \ldots, 2^w - 1\}$. The execution of basic arithmetic instructions on words takes constant time; our algorithms only need constant time addition, subtraction and comparison, as well as constant time generation of random words. Long integers are represented by a string of words. Floating point numbers are represented by an exponent (a string of words of some length k) and a mantissa (a string of words of some length ℓ). Addition and multiplication can then be done in time $\mathcal{O}(\text{poly}(\ell, k))$ and with error $2^{-w\ell}$. Note that the Word RAM offers an intrinsic parallelism where, in constant time, an operation on w bits can be performed in parallel.

Random Graph Models. In the *Erdős-Rényi* [9] random graph model $G(n, p)$, each edge of an n vertex graph is independently present with probability p. This yields a binomial degree distribution and approaches a Poisson distribution in the

[1] We assume that accessing the i-th memory cell costs $\mathcal{O}(1)$ although it might make more sense to assume cost proportional to the length of a pointer to i (which is $\Theta(1 + \log(i)/w)$) or larger. However, our results remain valid as long as this cost is $\mathcal{O}((2-\varepsilon)^{i/2^w})$ for some $\varepsilon > 0$.

limit. As many real-world networks have power-law degree distributions, we also study inhomogenous random graphs. We consider *Chung-Lu* [5] graphs $G(n, W)$ with n vertices and weights $W = (W_1, W_2, \dots, W_n) \in \mathbb{R}^n_{\geqslant 0}$. In this model, an edge between two vertices i and j is independently present with probability $p_{i,j} := \min\{W_i W_j / \sum_k W_k, 1\}$. For sufficiently large graphs, the expected degree of i converges to W_i. The related definitions of generalized random graph [25] with $p_{ij} = W_i W_j / (\sum_k W_k + W_i W_j)$ and Norros-Reittu random graphs [22] with $p_{i,j} = 1 - \exp(-W_i W_j / \sum_k W_k)$ can be handled in a similar way. However, we will focus on Chung-Lu random graphs.

Probability Distributions. Let $p \in (0, 1)$. The *Bernoulli* distribution $\mathrm{Ber}(p)$ takes values in $\{0, 1\}$ such that $\Pr[\mathrm{Ber}(p) = 1] = 1 - \Pr[\mathrm{Ber}(p) = 0] = p$. The *geometric* distribution $\mathrm{Geo}(p)$ takes values in $\mathbb{N}_0$ such that for any $i \in \mathbb{N}_0$, we have $\Pr[\mathrm{Geo}(p) = i] = p(1-p)^i$. For $n \in \mathbb{N}_0$, we define the *bounded geometric* distribution $\mathrm{Geo}(p, n)$ to be $\min\{n, \mathrm{Geo}(p)\}$. This means that $\mathrm{Geo}(p, n)$ takes values in $\{0, \dots, n\}$ such that for any $i \in \mathbb{N}_0$, $i < n$, we have $\Pr[\mathrm{Geo}(p, n) = i] = p(1-p)^i$, and $\Pr[\mathrm{Geo}(p, n) = n] = (1-p)^n$. The *uniform* distribution $\mathrm{Uni}[0, 1]$ takes values in $[0, 1]$ with uniform probability. For $n \in \mathbb{N}$, we define the uniform distribution $\mathrm{Uni}(n)$ to be the uniform distribution over $\{0, \dots, n-1\}$.

Input Model. We assume the input p to be given in the following form: We are given a number $k \in \mathbb{N}_0$ such that $2^{-k} \geqslant p > c2^{-k}$ for some fixed constant[2] $c > 0$. Moreover, for any $i \in \mathbb{N}$ we are able to compute a number $p_i \leqslant 1$ such that $|p_i - 2^k p| \leqslant 2^{-i}$. We can assume that p_i has at most $i + 1$ bits (otherwise take the first $i + 1$ bits of p_{i+1}, which are a 2^{-i}-approximation of $2^k p$). Since we assumed $w = \Omega(\log \log(1/p))$, k fits into $\mathcal{O}(1)$ words; this resembles the usual assumption that we can compute with numbers as large as the input/output size in constant time. Furthermore, we want to assume that p_i can be computed in time $\mathrm{poly}(i)$. This means that p can be approximated efficiently. However, it is sufficient even if the runtime is $\mathcal{O}((2-\varepsilon)^i)$ for some constant $\varepsilon > 0$. All numbers other than the input parameter p will be encoded as simple strings of words or floating point numbers, as discussed in the paragraph "machine models".

Notations. The base of all logarithms is 2. For integer division we use $a \operatorname{div} b := \lfloor a/b \rfloor$ for $a, b \in \mathbb{Z}$. We typically use x_i to denote the i-th bit (approximation) of x. We denote the set $\{1, \dots, n\}$ by $[n]$.

3 Sampling Geometric Random Variates

In this section we show a Word RAM algorithm for generating a geometric random variate $\mathrm{Geo}(p)$ in optimal expected runtime. We assume that the parameter p is given by a bitstream, i.e., we are given $k \in \mathbb{N}_0$ and can approximate p_* such that $p = 2^{-k} p_*$ and $c < p_* \leqslant 1$ for some $c > 0$. Approximating p_* with precision i needs time $\mathcal{O}((2-\varepsilon)^i)$ for some $\varepsilon > 0$. We first prove that the expected

[2] One could set $c = 1/2$, but our results hold more generally, in the case where we cannot compute such a good approximation of p.

output size is $\Theta(\log(1/p))$, which gives a lower bound of $\Omega(1 + \log(1/p)/w)$ for the expected runtime of any algorithm sampling $\mathrm{Geo}(p)$ on the Word RAM as (at most) w bits can be processed in parallel.

Lemma 1. *For any $p \in (0,1)$, we have $\mathbb{E}[\log(1+\mathrm{Geo}(p))] = \Theta(\log(1/p))$, where the lower bound holds for $1/p$ large enough.*

We now present an algorithm achieving this optimal expected runtime. The main trick is that we split up $\mathrm{Geo}(p)$ into $\mathrm{Geo}(p) \operatorname{div} 2^k$ and $\mathrm{Geo}(p) \bmod 2^k$. It is easy to see that both parts are independent random variables. Now, $\mathrm{Geo}(p) \operatorname{div} 2^k$ has constant expected value, so we can iteratively check whether it equals $0, 1, 2, \ldots$. On the other hand, $\mathrm{Geo}(p) \bmod 2^k$ is sufficiently well approximated by the uniform distribution over $\{0, \ldots, 2^k - 1\}$; the rejection method suffices for fast sampling. These ideas are brought together in Algorithm 1.

Algorithm 1. GenGeo(p) samples $\mathrm{Geo}(p)$ given a bitstream $p = 2^{-k} p_*$.

```
D ← 0
while Ber((1 − p)^{2^k}) do
    D ← D + 1
repeat
    M ←(lazy)— Uni(2^k)
until Ber((1 − p)^M)
fill up M with random bits
return 2^k D + M
```

Here, D represents $\mathrm{Geo}(p) \operatorname{div} 2^k$, initialized to 0. It is increased by 1 as long as a Bernoulli random variate $\mathrm{Ber}((1-p)^{2^k})$ turns out to be 1. Then M, corresponding to $\mathrm{Geo}(p) \bmod 2^k$, is chosen uniformly from the interval $\{0, \ldots, 2^k - 1\}$, but rejected with probability $(1-p)^M$. We sample M lazily, i.e., a bit of M is sampled only if needed by the test $\mathrm{Ber}((1-p)^M)$. After we leave the loop, M is filled up with random bits, so that we return the same value as if we had sampled M completely inside of the second loop. The result is, naturally, $2^k D + M$.

We will next discuss correctness of this algorithm, describe the details of how to implement it efficiently, and analyze its runtime. We postpone the issue of how to sample $\mathrm{Ber}((1-p)^n)$ to the end of this section. For the moment we will just assume that this can be done in expected constant time, looking at the first expected constant many bits of p and n.

Correctness. Let $n \geqslant 0$. The probability of outputting $n = 2^k D + M$ should be $p(1-p)^n$, i.e., it should be proportional to $(1-p)^n$. Following the algorithm step by step we see that the probability is

$$\underbrace{\left((1-p)^{2^k}\right)^D \cdot \left(1 - (1-p)^{2^k}\right)}_{\text{first loop}} \cdot \underbrace{\sum_{t \geqslant 0} \left(1 - \sum_{i=0}^{2^k-1} 2^{-k}(1-p)^i\right)^t 2^{-k}(1-p)^M}_{\text{second loop}},$$

where t is the number of iterations of the second loop; note that $2^{-k}(1-p)^i$ is the probability of outputting i in the first iteration of the second loop, so that $\sum_{i=0}^{2^k-1} 2^{-k}(1-p)^i$ is the probability of leaving the second loop after the first iteration. Collecting the factors dependent on D and M we see that this probability is proportional to $(1-p)^{2^k D+M} = (1-p)^n$, showing correctness of the algorithm.

Runtime. We show that the expected runtime of Algorithm 1 is $\mathcal{O}(1 + \log(1/p)/w)$. Again, assume that we can sample $\mathrm{Ber}((1-p)^n)$ in expected constant time. By the last section, incrementing the counter D can be done in amortized constant time, and we only need an expected constant number of bits of M during the second loop, after which we fill up M with random bits in time $\mathcal{O}(1 + \log(1/p)/w)$. Hence, if we show that the two loops run in expected constant time, then Algorithm 1 runs in expected time $\mathcal{O}(1 + \log(1/p)/w)$.

We consider the probabilities of dropping out of the two loops. Since $2^{-k} \geqslant p > c2^{-k}$, for the first loop this is

$$1 - (1-p)^{2^k} \geqslant 1 - (1-p)^{c/p} \geqslant 1 - e^{-c}, \tag{1}$$

so we have constant probability to drop out of this loop in every iteration. Moreover, the second loop terminates immediately if $k = 0$; otherwise we have

$$(1-p)^M \geqslant (1-p)^{2^k} \geqslant (1-2^{-k})^{2^k} \geqslant (1-1/2)^2 = 1/4, \tag{2}$$

so for the second loop we also have constant probability of dropping out.

To show that each loop runs in expected constant time, let T be a random variable denoting the number of iterations of the loop; note that $\mathbb{E}[T] = \mathcal{O}(1)$, since the probability of dropping out of each loop is $\Omega(1)$. Furthermore, let X_i be the runtime of the i-th iteration of the loop; note that by assumption we can sample $\mathrm{Ber}((1-p)^n)$ in expected constant time, so that $\mathbb{E}[X_i \mid T \geqslant i] = \mathcal{O}(1)$. The total runtime of the loop is $X_1 + \ldots + X_T$. Thus, the following lemma shows that the expected runtime of the loop is $\mathcal{O}(1)$. This finishes the proof of Theorem 3, aside from sampling $\mathrm{Ber}((1-p)^n)$.

Lemma 2. *Let T be a random variable with values in $\mathbb{N}_0$ and X_i, $i \in \mathbb{N}$, be random variables with values in $\mathbb{R}$; we assume no independence. Let $\alpha \in \mathbb{R}$ with $\mathbb{E}[X_i \mid T \geqslant i] \leqslant \alpha$ for all $i \in \mathbb{N}$. Then we have $\mathbb{E}[X_1 + \ldots + X_T] \leqslant \alpha \cdot \mathbb{E}[T]$.*

We remark that the above lemma is an easy special case of Wald's equation.

Note that the only points where this algorithm is using the Word RAM parallelism are when we fill up M and when we compute with exponents. The generation of $\mathrm{Ber}((1-p)^n)$, discussed in the remainder of this section, will use Word RAM parallelism only for working with exponents. The filling of M can be done in time $\mathcal{O}(1 + \log(1/p)/w)$ as we assumed that we can generate random words in unit time. Also note that given P processors, each one capable of performing Word RAM operations, we can trivially further parallelize this algorithm to run in expected time $\mathcal{O}\big(1 + \frac{\log(1/p)}{wP}\big)$.

Sampling Ber$((1-p)^n)$. It is left to show how to sample a Bernoulli random variable with parameter $(1-p)^n$. We can use the fact that we know k with $2^{-k} \geqslant p > c2^{-k}$ and can approximate $2^k p$ by p_i, and that $n \in \mathbb{N}$, $n \leqslant 2^k$. Note that we can easily get an approximation n_i of n of the form $|2^{-k}n - n_i| \leqslant 2^{-i}$ in the situation of Algorithm 1: In the first loop we have $n = 2^k$, then simply pick $n_i = 1$; in the second loop $n = M$ is uniform in $\{0, \ldots, 2^k - 1\}$, so that we get n_i by determining (i.e. flipping) the highest i bits of n. In this situation we can show the following lemma.

Lemma 3. *Given bitstream p with $2^{-k} \geqslant p = \Omega(2^{-k})$, for $n = 2^k$ or for uniformly random n in $\{0, \ldots, 2^k - 1\}$ we can sample* Ber$((1-p)^n)$ *in expected constant time.*

In the full version of this paper we discuss a method to sample Ber(q) in expected constant time which is closely related to Flajolet and Saheb [10].

Lemma 4. *A Bernoulli random variate* Ber(q) *with parameter $q \in (0,1)$ (given as a bitstream) can be sampled in constant expected runtime on a Word RAM.*

The only thing we need to efficiently sample Ber(q) is to be able to compute an approximation q_i of q with $|q - q_i| \leqslant 2^{-i}$ in time $\mathcal{O}((2-\varepsilon)^i)$. To get such an approximation for $(1-p)^n$, we make use of the binomial theorem $(1-p)^n = \sum_{j=0}^{n} \binom{n}{j}(-p)^j$. Noting that $\binom{n}{j} \leqslant \frac{n^j}{j!}$ and $n \leqslant 1/p$, we see that the j-th summand is absolutely bounded by $1/j!$. Moreover, the absolute value of the summands is monotonically decreasing in j, and their sign is $(-1)^j$, implying $\left|\sum_{j=i+2}^{n} \binom{n}{j}(-p)^j\right| \leqslant 1/(i+2)! \leqslant 2^{-i-1}$. Thus, by summing up only the first $i+2$ summands we get a good approximation of $(1-p)^n$.

Moreover, we have

$$\sum_{j=0}^{i+1} \binom{n}{j}(-p)^j = \frac{1}{(i+1)!} \sum_{j=0}^{i+1} (-p)^j \left(\prod_{h=j+1}^{i+1} h \right) \prod_{h=0}^{j-1} (n-h). \tag{3}$$

We will compute the right-hand side of this with working precision r. This means that we work with floating point numbers, with an exact exponent encoded by a string of words, and a mantissa which is a string of $\lceil r/w \rceil$ words. We get p and n up to working precision r by plugging in $2^{-k}p_r$ and $2^k n_r$. Then we calculate the numerator and denominator of the right-hand side independently with working precision r. Note that adding or multiplying the floating point numbers takes time $\mathcal{O}(\text{poly}(r))$ for adding/multiplying the mantissas (even using the school method for multiplication is fine for this), and $\mathcal{O}(1 + \log(i))$ for subtracting/adding the exponents, as all exponents in equation (3) are absolutely bounded by $\mathcal{O}(\text{poly}(i) \cdot k)$ and k fits in $\mathcal{O}(1)$ words.

Regarding runtime, noting that there are $\mathcal{O}(\text{poly}(i))$ operations to carry out in computing the right-hand side of equation (3), we see that we can compute the latter with working precision r in time $\mathcal{O}(\text{poly}(r, i))$. If we choose r large enough so that this yields an approximation of equation (3) with absolute error

at most 2^{-i-1}, then combined with the error analysis from using only the first $i+2$ terms, we get a runtime of $\mathcal{O}(\text{poly}(r,i))$ to compute an approximation of $(1-p)^n$ with absolute error 2^{-i}. Now, as long as we can choose $r = \text{poly}(i)$, this runtime is small enough to use Lemma 4, since we only needed an approximation of $(1-p)^n$ with absolute error 2^{-i} in time $\mathcal{O}((2-\varepsilon)^i)$ for some $\varepsilon > 0$. Under this assumption on r, we are done proving Lemma 3. The following lemma shows that $r = \text{poly}(i)$ is indeed sufficient.

Lemma 5. *The absolute error of computing equation* (3) *with working precision* $r = i + \alpha(1 + \log(i))$ *is at most* 2^{-i-1}*, for a large enough constant* α*.*

4 Generating Random Graphs

In this section we show that Erdős-Rényi and Chung-Lu random graphs can be efficiently generated. For this we simply take the efficient generation on Real RAMs from [1, 20] and replace the generation of bounded geometric variables by our algorithm from the last section. In the following we discuss why this is sufficient and leads to the runtimes claimed in Theorems 6 and 7.

Consider the original efficient generation algorithm of Erdős-Rényi random graphs described in [1], which is essentially the following. For each vertex $u \in [n]$ we want to sample its neighbors $v \in [u-1]$ in decreasing order. Defining $v_0 := u$, the first neighbor v_1 of u is distributed as $v_1 \sim v_0 - 1 - \text{Geo}(p, v_0 - 1)$, where the event $v_1 = 0$ represents that u has no neighbor. Then the next neighbor is distributed as $v_2 \sim v_1 - 1 - \text{Geo}(p, v_1 - 1)$ and so on. Sampling the graph in this way, we use $m + n$ bounded geometric variables, where m is the number of edges in the final graph (which is a random variable).

In this algorithm we have to cope with indices of vertices, thus, it is natural to assume $w = \Omega(\log n)$. Under this assumption, all single operations of the original algorithm can be performed in worst-case constant time on a Word RAM, except for the generation of bounded geometric variables $\text{Geo}(p,k)$, with $k \leqslant n$. The latter, however, can be done in expected time $\mathcal{O}(1+\log(\min\{n, 1/p\})/w) = \mathcal{O}(1)$ using our algorithm from Theorem 5. Hence, the expected runtime of the modified algorithm (with replaced sampling of bounded geometric variables) should be the same as that of the original algorithm. To prove this, consider the runtime the modified algorithm spends on sampling bounded geometric variables. This random variable can be written as $X_1 + \ldots + X_T$, where T is a random variable denoting the number of bounded geometric variables sampled by the algorithm, and X_i is the time spent on sampling the i-th such variable. Note that $\mathbb{E}[X_i \mid T \geqslant i] = \mathcal{O}(1)$. Thus, by Lemma 2 we can bound $\mathbb{E}[X_1 + \ldots + X_T]$ by $\mathcal{O}(\mathbb{E}[T])$. Since the original algorithm spends time $\Omega(T)$, the total expected runtime of the modified algorithm is asymptotically the same as the expected runtime of the original algorithm, namely $\mathcal{O}(n + pn^2)$. This runtime is optimal, as writing down a graph takes time $\Omega(n+m)$ (each index needs $\Theta(1)$ words; this depends, however, on the representation of the graph). Noting that the space requirements of the algorithm are met by Theorem 5, this proves Theorem 6.

A similar result applies to the more general Chung-Lu random graphs $\mathcal{G}(n, W)$. Again we assume $w = \Omega(\log n)$. Let us further assume, for simplicity, that all given weights W_u, $u \in V$ are rational numbers with the same denominator, with each numerator and the common denominator fitting in $\mathcal{O}(1)$ words. In this case, the sum $S = \sum_{u \in V} W_u$ has the same denominator as all W_u and numerator bounded by n times the numerator of the largest W_u. Since $w = \Omega(\log n)$, the numerator of S fits in $\mathcal{O}(1)$ more words than used for the largest W_u. Hence, numerator and denominator of S fit in $\mathcal{O}(1)$ words and can be computed in $\mathcal{O}(n)$ time. Moreover, the edge probabilities $p_{u,v} = \min\{W_u W_v / S, 1\}$ are also rationals with numerator and denominator fitting in $\mathcal{O}(1)$ words that can be computed in constant time if S is available.

Carefully examining the efficient sampling algorithm for Chung-Lu random graphs, Algorithm 2 of Miller and Hagberg [20], we see that now every step can be performed in the same deterministic time bound as on a Real RAM, except for the generation of bounded geometric variables and Bernoulli variables. Note that for any $p \in (0, 1)$ we have $\mathrm{Ber}(p) \sim \mathrm{Geo}(1 - p, 1)$, so Theorem 5 shows that the bounded geometric as well as the Bernoulli random variables can be sampled in expected constant time and bounded space (for $w = \Omega(\log n)$). Thus, we can bound the expected runtime of the modified generation for Chung-Lu graphs analogously to the Erdős-Rényi case, proving Theorem 7.

5 Conclusions and Future Work

We have presented new exact algorithms which can sample $\mathrm{Geo}(p)$ and $\min\{n, \mathrm{Geo}(p)\}$ in optimal time and space on a Word RAM. It remains open to find similar algorithms for other non-uniform random variates besides exponential and normal distributions. Moreover, it would be interesting to see whether our theoretically optimal algorithms are also practical, e.g., for generating very large geometric random variates for cryptographic applications [13].

Regarding our new exact algorithm for sampling Erdős-Rényi and Chung-Lu random graphs in optimal time and space on a Word RAM, we believe that similar results can be proven for the more general case where the weights W_u are given as bitstreams.

References

[1] Batagelj, V., Brandes, U.: Efficient generation of large random networks. Phys. Rev. E 71, 036113 (2005)

[2] Blanca, A., Mihail, M.: Efficient generation ϵ-close to G(n,p) and generalizations (2012), `arxiv.org/abs/1204.5834`

[3] Blitzstein, J.K., Diaconis, P.: A sequential importance sampling algorithm for generating random graphs with prescribed degrees. Internet Mathematics 6, 489–522 (2011)

[4] Brent, R., Zimmermann, P.: Modern Computer Arithmetic. Cambridge University Press (2010)

[5] Chung, F., Lu, L.: Connected components in random graphs with given expected degree sequences. Annals of Combinatorics 6, 125–145 (2002)
[6] Cook, S.A., Reckhow, R.A.: Time bounded random access machines. J. Comput. Syst. Sci. 7, 354–375 (1973)
[7] De, A., Kurur, P.P., Saha, C., Saptharishi, R.: Fast integer multiplication using modular arithmetic. In: 40th Symp. Theory of Computing (STOC), pp. 499–506 (2008)
[8] Devroye, L.: Non-uniform random variate generation. Springer (1986)
[9] Erdős, P., Rényi, A.: On random graphs. Publ. Math. Debrecen 6, 290–297 (1959)
[10] Flajolet, P., Saheb, N.: The complexity of generating an exponentially distributed variate. J. Algorithms 7, 463–488 (1986)
[11] Flajolet, P., Pelletier, M., Soria, M.: On Buffon machines and numbers. In: 22nd Symp. Discrete Algorithms (SODA), pp. 172–183 (2011)
[12] Fürer, M.: Faster integer multiplication. SIAM J. Comput. 39, 979–1005 (2009)
[13] Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. In: 41st Symp. Theory of Computing (STOC), pp. 351–360 (2009)
[14] Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using NetworkX. In: 7th Conf. Python in Science, pp. 11–15 (2008)
[15] Karney, C.: Random number library, version 1.5 (2012), http://sourceforge.net/projects/randomlib
[16] Keeling, M.J., Eames, K.T.: Networks and epidemic models. Journal of The Royal Society Interface 2, 295–307 (2005)
[17] Kim, J.H., Vu, V.H.: Generating random regular graphs. In: 35th Symp. Theory of Computing (STOC), pp. 213–222. ACM (2003)
[18] Knuth, D.E., Yao, A.C.-C.: The complexity of nonuniform random number generation. In: Traub, J.F. (ed.) Symposium on Algorithms and Complexity: New Directions and Recent Results, pp. 357–428 (1976)
[19] Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press (1999)
[20] Miller, J.C., Hagberg, A.: Efficient generation of networks with given expected degrees. In: Frieze, A., Horn, P., Prałat, P. (eds.) WAW 2011. LNCS, vol. 6732, pp. 115–126. Springer, Heidelberg (2011)
[21] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: Simple building blocks of complex networks. Science 298, 824–827 (2002)
[22] Norros, I., Reittu, H.: On a conditionally Poissonian graph process. Advances in Applied Probability 38, 59–75 (2006)
[23] Ray, J., Pinar, A., Seshadhri, C.: Are we there yet? When to stop a markov chain while generating random graphs. In: Bonato, A., Janssen, J. (eds.) WAW 2012. LNCS, vol. 7323, pp. 153–164. Springer, Heidelberg (2012)
[24] Solovay, R., Strassen, V.: A fast Monte-Carlo test for primality. SIAM J. Comput. 6, 84–85 (1977)
[25] van der Hofstad, R.: Random graphs and complex networks (2009), http://www.win.tue.nl/~rhofstad/NotesRGCN.pdf
[26] von Neumann, J.: Various techniques used in connection with random digits. In: Householder, A.S., et al. (eds.) The Monte Carlo Method. National Bureau of Standards, Applied Mathematics Series, vol. 12, pp. 36–38 (1951)

Finding Short Paths on Polytopes by the Shadow Vertex Algorithm*

Tobias Brunsch and Heiko Röglin

Department of Computer Science
University of Bonn, Germany
{brunsch,roeglin}@cs.uni-bonn.de

Abstract. We show that the shadow vertex algorithm can be used to compute a short path between a given pair of vertices of a polytope $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ along the edges of P, where $A \in \mathbb{R}^{m \times n}$. Both, the length of the path and the running time of the algorithm, are polynomial in m, n, and a parameter $1/\delta$ that is a measure for the flatness of the vertices of P. For integer matrices $A \in \mathbb{Z}^{m \times n}$ we show a connection between δ and the largest absolute value Δ of any sub-determinant of A, yielding a bound of $O(\Delta^4 mn^4)$ for the length of the computed path. This bound is expressed in the same parameter Δ as the recent non-constructive bound of $O(\Delta^2 n^4 \log(n\Delta))$ by Bonifas et al. [1].

For the special case of totally unimodular matrices, the length of the computed path simplifies to $O(mn^4)$, which significantly improves the previously best known constructive bound of $O(m^{16} n^3 \log^3(mn))$ by Dyer and Frieze [7].

1 Introduction

We consider the following problem: Given a matrix $A = [a_1, \ldots, a_m]^{\mathrm{T}} \in \mathbb{R}^{m \times n}$, a vector $b \in \mathbb{R}^m$, and two vertices x_1 and x_2 of the polytope $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, find a short path from x_1 to x_2 along the edges of P efficiently. In this context *efficient* means that the running time of the algorithm is polynomially bounded in m, n, and the length of the path it computes. Note, that the polytope P does not have to be bounded.

The *diameter* $d(P)$ of the polytope P is the smallest integer d that bounds the length of the shortest path between any two vertices of P from above. The *polynomial Hirsch conjecture* states that the diameter of P is polynomially bounded in m and n for any matrix A and any vector b. As long as this conjecture remains unresolved, it is unclear whether there always exists a path of polynomial length between the given vertices x_1 and x_2. Moreover, even if such a path exists, it is open whether there is an efficient algorithm to find it.

Related Work. The diameter of polytopes has been studied extensively in the last decades. In 1957 Hirsch conjectured that the diameter of P is bounded by

* This research was supported by ERC Starting Grant 306465 (BeyondWorstCase).

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 279–290, 2013.

$m-n$ for any matrix A and any vector b (see Dantzig's seminal book about linear programming [6]). This conjecture has been disproven by Klee and Walkup [9] who gave an unbounded counterexample. However, it remained open for quite a long time whether the conjecture holds for bounded polytopes. More than fourty years later Santos [12] gave the first counterexample to this refined conjecture showing that there are bounded polytopes P for which $d(P) \geq (1+\varepsilon)\cdot m$ for some $\varepsilon > 0$. This is the best known lower bound today. On the other hand, the best known upper bound of $O(m^{1+\log n})$ due to Kalai and Kleitman [8] is only quasi-polynomial. It is still an open question whether $d(P)$ is always polynomially bounded in m and n. This has only been shown for special classes of polytopes like 0/1 polytopes, flow-polytopes, and the transportation polytope. For these classes of polytopes bounds of $m - n$ (Naddef [10]), $O(mn \log n)$ (Orlin [11]), and $O(m)$ (Brightwell et al. [3]) have been shown, respectively. On the other hand, there are bounds on the diameter of far more general classes of polytopes that depend polynomially on m, n, and on additional parameters. Recently, Bonifas et al. [1] showed that the diameter of polytopes P defined by integer matrices A is bounded by a polynomial in n and a parameter that depends on the matrix A. They showed that $d(P) = O(\Delta^2 n^4 \log(n\Delta))$, where Δ is the largest absolute value of any sub-determinant of A. Although the parameter Δ can be very large in general, this approach allows to obtain bounds for classes of polytopes for which Δ is known to be small. For example, if the matrix A is totally unimodular, i.e., if all sub-determinants of A are from $\{-1, 0, 1\}$, then their bound simplifies to $O(n^4 \log n)$, improving the previously best known bound of $O(m^{16} n^3 \log^3(mn))$ by Dyer and Frieze [7].

We are not only interested in the existence of a short path between two vertices of a polytope but we want to compute such a path efficiently. It is clear that lower bounds for the diameter of polytopes have direct (negative) consequences for this algorithmic problem. However, upper bounds for the diameter do not necessarily have algorithmic consequences as they might be non-constructive. The aforementioned bounds of Orlin, Brightwell et al., and Dyer and Frieze are constructive, whereas the bound of Bonifas et al. is not.

Our Contribution. We give a constructive upper bound for the diameter of the polytope $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ for arbitrary matrices $A \in \mathbb{R}^{m \times n}$ and arbitrary vectors $b \in \mathbb{R}^m$.[1] This bound is polynomial in m, n, and a parameter $1/\delta$, which depends only on the matrix A and is a measure for the angle between edges of the polytope P and their neighboring facets. We say that a facet F of the polytope P is neighboring an edge e if exactly one of the endpoints of e belongs to F. The parameter δ denotes the smallest sine of any angle between an edge and a neighboring facet in P. If, for example, every edge is orthogonal to its neighboring facets, then $\delta = 1$. On the other hand, if there exists an edge that is almost parallel to a neighboring facet, then $\delta \approx 0$. The formal definition of δ is deferred to Section 4.

A well-known pivot rule for the simplex algorithm is the shadow vertex rule, which gained attention in recent years because it has been shown to have poly-

[1] Note that we do not require the polytope to be bounded.

nomial running time in the framework of smoothed analysis [13]. We will present a randomized variant of this pivot rule that computes a path between two given vertices of the polytope P. We will introduce this variant in Section 2 and we call it *shadow vertex algorithm* in the following.

Theorem 1. *Given vertices x_1 and x_2 of P, the shadow vertex algorithm efficiently computes an x_1-x_2-path on the polytope P with expected length $O\left(\frac{mn^2}{\delta^2}\right)$.*

Let us emphasize that the algorithm is very simple and its running time depends only polynomially on m, n and the length of the path it computes.

Theorem 1 does not resolve the polynomial Hirsch conjecture as δ can be exponentially small. Furthermore, it does not imply a good running time of the shadow vertex method for optimizing linear programs because for the variant considered in this paper both vertices have to be known. Contrary to this, in the optimization problem the objective is to determine the optimal vertex. To compare our results with the result by Bonifas et al. [1], we show that, if A is an integer matrix, then $\frac{1}{\delta} \leq n\Delta^2$, which yields the following corollary.

Corollary 2. *Let $A \in \mathbb{Z}^{m\times n}$ be an integer matrix and let $b \in \mathbb{R}^m$ be a real-valued vector. Given vertices x_1 and x_2 of P, the shadow vertex algorithm efficiently computes an x_1-x_2-path on the polytope P with expected length $O(\Delta^4 mn^4)$.*

This bound is worse than the bound of Bonifas et al., but it is constructive. Furthermore, if A is a totally unimodular matrix, then $\Delta = 1$. Hence, we obtain the following corollary.

Corollary 3. *Let $A \in \mathbb{Z}^{m\times n}$ be a totally unimodular matrix and let $b \in \mathbb{R}^m$ be a vector. Given vertices x_1 and x_2 of P, the shadow vertex algorithm efficiently computes an x_1-x_2-path on the polytope P with expected length $O(mn^4)$.*

This is a significant improvement upon the previously best constructive bound of $O(m^{16}n^3 \log^3(mn))$ due to Dyer and Frieze because we can assume $m \geq n$. Otherwise, P does not have vertices and the problem is ill-posed.

Organization of the Paper. In Section 2 we describe the shadow vertex algorithm. In Section 3 we give an outline of our analysis and present the main ideas. After that, in Section 4, we introduce the parameter δ and discuss some of its properties. Section 5 is devoted to the proof of Theorem 1. We omitted some proofs due to space limitations.

2 The Shadow Vertex Algorithm

Let us first introduce some notation. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. Let $A \in \mathbb{R}^{m\times n}$ be an $m \times n$-matrix and let $i \in [m]$ and $j \in [n]$ be indices. With $A_{i,j}$ we refer to the $(m-1) \times (n-1)$-submatrix obtained from A by removing the i^{th} row and the j^{th} column. We call the determinant of any $k \times k$-submatrix of A a *sub-determinant of A of size k*. By $\mathbb{I}_n$ we denote the

$n \times n$-identity matrix $\mathrm{diag}(1, \ldots, 1)$ and by $\mathbb{O}_{m \times n}$ the $m \times n$-zero matrix. If $n \in \mathbb{N}$ is clear from the context, then we define vector e_i to be the i^{th} column of $\mathbb{I}_n$. For a vector $x \in \mathbb{R}^n$ we denote by $\|x\| = \|x\|_2$ the Euclidean norm of x and by $\mathcal{N}(x) = \frac{1}{\|x\|} \cdot x$ for $x \neq 0$ the normalization of vector x.

2.1 Shadow Vertex Pivot Rule

Our algorithm is inspired by the shadow vertex pivot rule for the simplex algorithm. Before describing our algorithm, we will briefly explain the geometric intuition behind this pivot rule. For a complete and more formal description, we refer the reader to [2] or [13]. Let us consider the linear program $\min c^{\mathrm{T}} x$ subject to $x \in P$ for some vector $c \in \mathbb{R}^n$ and assume that an initial vertex x_1 of the polytope P is known. For the sake of simplicity, we assume that there is a unique optimal vertex $x^\star$ of P that minimizes the objective function $c^{\mathrm{T}} x$. The shadow vertex pivot rule first computes a vector $w \in \mathbb{R}^n$ such that the vertex x_1 minimizes the objective function $w^{\mathrm{T}} x$ subject to $x \in P$. Again for the sake of simplicity, let us assume that the vectors c and w are linearly independent.

In the second step, the polytope P is projected onto the plane spanned by the vectors c and w. The resulting projection is a polygon P' and one can show that the projections of both the initial vertex x_1 and the optimal vertex $x^\star$ are vertices of this polygon. Additionally every edge between two vertices x and y of P' corresponds to an edge of P between two vertices that are projected onto x and y, respectively. Due to these properties a path from the projection of x_1 to the projection of $x^\star$ along the edges of P' corresponds to a path from x_1 to $x^\star$ along the edges of P.

This way, the problem of finding a path from x_1 to $x^\star$ on the polytope P is reduced to finding a path between two vertices of a polygon. There are at most two such paths and the shadow vertex pivot rule chooses the one along which the objective $c^{\mathrm{T}} x$ improves.

2.2 Our Algorithm

As described in the introduction we consider the following problem: We are given a matrix $A = [a_1, \ldots, a_m]^{\mathrm{T}} \in \mathbb{R}^{m \times n}$, a vector $b \in \mathbb{R}^m$, and two vertices x_1, x_2 of the polytope $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. Our objective is to find a short path from x_1 to x_2 along the edges of P.

We propose the following variant of the shadow vertex pivot rule to solve this problem: First choose two vectors $w_1, w_2 \in \mathbb{R}^n$ such that x_1 uniquely minimizes $w_1^{\mathrm{T}} x$ subject to $x \in P$ and x_2 uniquely maximizes $w_2^{\mathrm{T}} x$ subject to $x \in P$. Then project the polytope onto the plane spanned by w_1 and w_2 in order to obtain a polygon P'. Let us call the projection π. By the same arguments as for the shadow vertex pivot rule, it follows that $\pi(x_1)$ and $\pi(x_2)$ are vertices of P' and that a path from $\pi(x_1)$ to $\pi(x_2)$ along the edges of P' can be translated into a path from x_1 to x_2 along the edges of P. Hence, it suffices to compute such a path to solve the problem. Again computing such a path is easy because P' is a two-dimensional polygon.

The vectors w_1 and w_2 are not uniquely determined, but they can be chosen from cones that are determined by the vertices x_1 and x_2 and the polytope P. We choose w_1 and w_2 randomly from these cones. A more precise description of this algorithm is given as Algorithm 1.

Algorithm 1. Shadow Vertex Algorithm

1: Determine n linearly independent rows u_k^{T} of A for which $u_k^{\mathrm{T}} x_1 = b_k$.
2: Determine n linearly independent rows v_k^{T} of A for which $v_k^{\mathrm{T}} x_2 = b_k$.
3: Draw vectors $\lambda, \mu \in (0,1]^n$ independently and uniformly at random.
4: Set $w_1 = -[\mathcal{N}(u_1), \ldots, \mathcal{N}(u_n)] \cdot \lambda$ and $w_2 = [\mathcal{N}(v_1), \ldots, \mathcal{N}(v_n)] \cdot \mu$.
5: Use the function $\pi : x \mapsto \left(w_1^{\mathrm{T}} x, w_2^{\mathrm{T}} x\right)$ to project P onto the Euclidean plane and obtain the shadow vertex polygon $P' = \pi(P)$.
6: Walk from $\pi(x_1)$ along the edges of P' in increasing direction of the second coordinate until $\pi(x_2)$ is found.
7: Output the corresponding path of P.

Let us give some remarks about the algorithm above. The vectors $u_1, \ldots, u_n$ in Line 1 and the vectors $v_1, \ldots, v_n$ in Line 2 must exist because x_1 and x_2 are vertices of P. The only point where our algorithm makes use of randomness is in Line 3. By the choice of w_1 and w_2 in Line 4, x_1 is the unique optimum of the linear program $\min w_1^{\mathrm{T}} x$ s.t. $x \in P$ and x_2 is the unique optimum of the linear program $\max w_2^{\mathrm{T}} x$ s.t. $x \in P$. The former follows because for any $y \in P$ with $y \neq x_1$ there must be an index $k \in [n]$ with $u_k^{\mathrm{T}} x_1 < b_k$. The latter follows analogously. Note, that $\|w_1\| \leq \sum_{k=1}^{n} \lambda_k \cdot \|\mathcal{N}(u_k)\| \leq \sum_{k=1}^{n} \lambda_k \leq n$ and, similarly, $\|w_2\| \leq n$. The shadow vertex polygon P' in Line 5 has several important properties: The projections of x_1 and x_2 are vertices of P' and all edges of P' correspond to projected edges of P. Hence, any path on the edges of P' is the projection of a path on the edges of P. Though we call P' a polygon, it does not have to be bounded. This is the case if P is unbounded in the directions w_1 or $-w_2$. Nevertheless, there is always a path from x_1 to x_2 which will be found in Line 6. For more details about the shadow vertex pivot rule and formal proofs of these properties, we refer to the book of Borgwardt [2].

Let us consider the projection $P' = \pi(P)$ of P to the Euclidean plane. We denote the first coordinate by ξ and the second coordinate by η. Since w_1 and w_2 are chosen such that x_1 and x_2 are, among the points of P, optimal for the function $x \mapsto w_1^{\mathrm{T}} x$ and $x \mapsto w_2^{\mathrm{T}} x$, respectively, the projections $\pi(x_1)$ and $\pi(x_2)$ of x_1 and x_2 must be the leftmost vertex and the topmost vertex of P', respectively. As P' is a (not necessarily bounded) polygon, this implies that if we start in vertex $\pi(x_1)$ and follow the edges of P' in direction of increasing values of η, then we will end up in $\pi(x_2)$ after a finite number of steps. This is not only true if P' is bounded but also if P is unbounded. Moreover, note that the slopes of the edges of the path from $\pi(x_1)$ to $\pi(x_2)$ are positive and monotonically decreasing.

3 Outline of the Analysis

In the remainder of this paper we assume that the polytope P is non-degenerate, i.e., for each vertex x of P there are exactly n indices i for which $a_i^{\mathrm{T}} x = b_i$. This implies that for any edge between two vertices x and y of P there are exactly $n-1$ indices i for which $a_i^{\mathrm{T}} x = a_i^{\mathrm{T}} y = b_i$.

From the description of the shadow vertex algorithm it is clear that the main step in proving Theorem 1 is to bound the expected number of edges on the path from $\pi(x_1)$ to $\pi(x_2)$ on the polygon P'. In order to do this, we look at the slopes of the edges on this path. As we discussed above, the sequence of slopes is monotonically decreasing. We will show that due to the randomness in the objective functions w_1 and w_2, it is even strictly decreasing with probability one. Furthermore all slopes on this path are bounded from below by 0.

Instead of counting the edges on the path from $\pi(x_1)$ to $\pi(x_2)$ directly, we will count the number of different slopes in the interval $[0, 1]$ and we observe that the expected number of slopes from the interval $[0, \infty)$ is twice the expected number of slopes from the interval $[0, 1]$. In order to count the number of slopes in $[0, 1]$, we partition the interval $[0, 1]$ into several small subintervals and we bound for each of these subintervals I the expected number of slopes in I. Then we use linearity of expectation to obtain an upper bound on the expected number of different slopes in $[0, 1]$, which directly translates into an upper bound on the expected number of edges on the path from $\pi(x_1)$ to $\pi(x_2)$.

We choose the subintervals so small that, with high probability, none of them contains more than one slope. Then, the expected number of slopes in a subinterval $I = (t, t+\varepsilon]$ is approximately equal to the probability that there is a slope in the interval I. In order to bound this probability, we use a technique reminiscent of the principle of deferred decisions that we have already used in [5]. The main idea is to split the random draw of the vectors w_1 and w_2 in the shadow vertex algorithm into two steps. The first step reveals enough information about the realizations of these vectors to determine the last edge $e = (\hat{p}, p^\star)$ on the path from $\pi(x_1)$ to $\pi(x_2)$ whose slope is bigger than t. Even though e is determined in the first step, its slope is not. We argue that there is still enough randomness left in the second step to bound the probability that the slope of e lies in the interval $(t, t + \varepsilon]$ from above, yielding Theorem 1.

We will now give some more details on how the random draw of the vectors w_1 and w_2 is partitioned. Let $\hat{x}$ and $x^\star$ be the vertices of the polytope P that are projected onto $\hat{p}$ and $p^\star$, respectively. Due to the non-degeneracy of the polytope P, there are exactly $n-1$ constraints that are tight for both $\hat{x}$ and $x^\star$ and there is a unique constraint $a_i^{\mathrm{T}} x \leq b_i$ that is tight for $x^\star$ but not for $\hat{x}$. In the first step the vector w_1 is completely revealed while instead of w_2 only an element $\tilde{w}_2$ from the ray $\{w_2 + \gamma \cdot a_i : \gamma \geq 0\}$ is revealed. We then argue that knowing w_1 and $\tilde{w}_2$ suffices to identify the edge e. The only randomness left in the second step is the exact position of the vector w_2 on the ray $\{\tilde{w}_2 - \gamma \cdot a_i : \gamma \geq 0\}$, which suffices to bound the probability that the slope of e lies in the interval $(t, t + \varepsilon]$.

Let us remark that the proof of Theorem 1 is inspired by the recent smoothed analysis of the successive shortest path algorithm for the minimum-cost flow problem [4]. Even though the general structure bears some similarity, the details of our analysis are much more involved.

4 The Parameter δ

In this section we define the parameter δ that describes the flatness of the vertices of the polytope and state some relevant properties.

Definition 4. *1. Let $z_1, \ldots, z_n \in \mathbb{R}^n$ be linearly independent vectors and let $\varphi \in (0, \frac{\pi}{2}]$ be the angle between z_n and the hyperplane $\operatorname{span}\{z_1, \ldots, z_{n-1}\}$. By $\hat{\delta}(\{z_1, \ldots, z_{n-1}\}, z_n) = \sin\varphi$ we denote the sine of angle φ. Moreover, we set $\delta(z_1, \ldots, z_n) = \min_{k\in[n]} \hat{\delta}(\{z_i : i \in [n] \setminus \{k\}\}, z_k)$.*

2. Given a matrix $A = [a_1, \ldots, a_m]^{\mathrm{T}} \in \mathbb{R}^{m\times n}$, we set

$$\delta(A) = \min\{\delta(a_{i_1}, \ldots, a_{i_n}) : a_{i_1}, \ldots, a_{i_n} \text{ linearly independent}\}.$$

The value $\hat{\delta}(\{z_1, \ldots, z_{n-1}\}, z_n)$ describes how orthogonal z_n is to the span of $z_1, \ldots, z_{n-1}$. If $\varphi \approx 0$, i.e., z_n is close to the span of $z_1, \ldots, z_{n-1}$, then $\hat{\delta}(\{z_1, \ldots, z_{n-1}\}, z_n) \approx 0$. On the other hand, if z_n is orthogonal to $z_1, \ldots, z_{n-1}$, then $\varphi = \frac{\pi}{2}$ and $\hat{\delta}(\{z_1, \ldots, z_{n-1}\}, z_n) = 1$. The value $\hat{\delta}(\{z_1, \ldots, z_{n-1}\}, z_n)$ equals the distance between both faces of the parallelotope Q, given by $Q = \{\sum_{i=1}^n \alpha_i \cdot \mathcal{N}(z_i) : \alpha_i \in [0,1]\}$, that are parallel to $\operatorname{span}\{z_1, \ldots, z_{n-1}\}$ and is scale invariant.

The value $\delta(z_1, \ldots, z_n)$ equals twice the inner radius r_n of the parallelotope Q and, thus, is a measure of the flatness of Q: A value $\delta(z_1, \ldots, z_n) \approx 0$ implies that Q is nearly $(n-1)$-dimensional. On the other hand, if $\delta(z_1, \ldots, z_n) = 1$, then the vectors $z_1, \ldots, z_n$ are pairwise orthogonal, that is, Q is an n-dimensional unit cube.

The next lemma lists some useful statements concerning the parameter $\delta = \delta(A)$ including a connection to the parameters Δ_1, Δ_{n-1}, and Δ introduced in the paper of Bonifas et al. [1].

Lemma 5. *Let $z_1, \ldots, z_n \in \mathbb{R}^n$ be linearly independent vectors, let $A \in \mathbb{R}^{m\times n}$ be a matrix, let $b \in \mathbb{R}^m$ be a vector, and let $\delta = \delta(A)$. Then, the following claims hold true:*

1. *If M is the inverse of $[\mathcal{N}(z_1), \ldots, \mathcal{N}(z_n)]^{\mathrm{T}}$, then*
$$\delta(z_1, \ldots, z_n) = \frac{1}{\max_{k\in[n]} \|m_k\|} \le \frac{\sqrt{n}}{\max_{k\in[n]} \|M_k\|},$$
where $[m_1, \ldots, m_n] = M$ and $[M_1, \ldots, M_n] = M^{\mathrm{T}}$.
2. *If $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix, then $\delta(Qz_1, \ldots, Qz_n) = \delta(z_1, \ldots, z_n)$.*
3. *Let y_1 and y_2 be two neighboring vertices of $P = \{x \in \mathbb{R}^n : Ax \le b\}$ and let a_i^{T} be a row of A. If $a_i^{\mathrm{T}} \cdot (y_2 - y_1) \neq 0$, then $|a_i^{\mathrm{T}} \cdot (y_2 - y_1)| \ge \delta \cdot \|y_2 - y_1\|$.*
4. *If A is an integral matrix, then $\frac{1}{\delta} \le n\Delta_1\Delta_{n-1} \le n\Delta^2$, where Δ, Δ_1, and Δ_{n-1} are the largest absolute values of any sub-determinant of A of arbitrary size, of size 1, and of size $n-1$, respectively.*

5 Analysis

For the proof of Theorem 1 we assume that $\|a_i\| = 1$ for all $i \in [m]$. This entails no loss of generality since normalizing the rows of matrix A (and scaling the right-hand side b appropriately) does neither change the behavior of our algorithm nor does it change the parameter $\delta = \delta(A)$.

For given linear functions L_1 and L_2, we denote by $\pi = \pi_{L_1,L_2}$ the function $\pi\colon \mathbb{R}^n \to \mathbb{R}^2$, given by $\pi(x) = (L_1(x), L_2(x))$. Note, that n-dimensional vectors can be treated as linear functions. By $P' = P'_{L_1,L_2}$ we denote the projection $\pi(P)$ of polytope P onto the Euclidean plane, and by $R = R_{L_1,L_2}$ we denote the path from $\pi(x_1)$ to $\pi(x_2)$ along the edges of polygon P'.

Our goal is to bound the expected number of edges of the path $R = R_{w_1,w_2}$ which is random since w_1 and w_2 depend on the realizations of the random vectors λ and μ. Each edge of R corresponds to a slope in $(0, \infty)$. These slopes are pairwise distinct with probability one (see Lemma 8). Hence, the number of edges of R equals the number of distinct slopes of R. In order to bound the expected number of distinct slopes we first restrict our attention to slopes in the interval $(0, 1]$.

Definition 6. *For a real $\varepsilon > 0$ let $\mathbb{F}_\varepsilon$ denote the event that there are three pairwise distinct vertices z_1, z_2, z_3 of P such that z_1 and z_3 are neighbors of z_2 and such that*

$$\left| \frac{w_2^{\mathrm{T}} \cdot (z_2 - z_1)}{w_1^{\mathrm{T}} \cdot (z_2 - z_1)} - \frac{w_2^{\mathrm{T}} \cdot (z_3 - z_2)}{w_1^{\mathrm{T}} \cdot (z_3 - z_2)} \right| \leq \varepsilon\,.$$

Note that if event $\mathbb{F}_\varepsilon$ does not occur, then all slopes of R differ by more than ε. Particularly, all slopes are pairwise distinct. First of all we show that event $\mathbb{F}_\varepsilon$ is very unlikely to occur if ε is chosen sufficiently small.

Lemma 7. *The probability that there are two neighboring vertices z_1, z_2 of P such that $|w_1^{\mathrm{T}} \cdot (z_2 - z_1)| \leq \varepsilon \cdot \|z_2 - z_1\|$ is bounded from above by $\frac{2m^n\varepsilon}{\delta}$.*

Lemma 8. *The probability of event $\mathbb{F}_\varepsilon$ tends to 0 for $\varepsilon \to 0$.*

Let $p \neq \pi(x_2)$ be a vertex of R. We call the slope s of the edge incident to p to the right of p *the slope of* p. As a convention, we set the slope of $\pi(x_2)$ to 0 which is smaller than the slope of any other vertex p of R.

Let $t \geq 0$ be an arbitrary real, let $\hat{p}$ be the right-most vertex of R whose slope is larger than t, and let $p^\star$ be the right neighbor of $\hat{p}$. Let $\hat{x}$ and $x^\star$ be the neighboring vertices of P with $\pi(\hat{x}) = \hat{p}$ and $\pi(x^\star) = p^\star$. Now let $i = i(x^\star, \hat{x}) \in [m]$ be the index for which $a_i^{\mathrm{T}} x^\star = b_i$ and for which $\hat{x}$ is the (unique) neighbor x of $x^\star$ for which $a_i^{\mathrm{T}} x < b_i$. This index is unique due to the non-degeneracy of the polytope P. For an arbitrary real $\gamma \geq 0$ we consider the vector $\tilde{w}_2 = w_2 + \gamma \cdot a_i$.

Lemma 9. *Let $\tilde{\pi} = \pi_{w_1,\tilde{w}_2}$ and let $\tilde{R} = R_{w_1,\tilde{w}_2}$ be the path from $\tilde{\pi}(x_1)$ to $\tilde{\pi}(x_2)$ in the projection $\tilde{P}' = P'_{w_1,\tilde{w}_2}$ of polytope P. Furthermore, let $\tilde{p}^\star$ be the left-most vertex of $\tilde{R}$ whose slope does not exceed t. Then, $\tilde{p}^\star = \tilde{\pi}(x^\star)$.*

Let us reformulate the statement of Lemma 9 as follows: The vertex $\tilde{p}^\star$ is defined for the path $\tilde{R}$ of polygon $\tilde{P}'$ with the same rules as used to define the vertex $p^\star$ of the original path R of polygon P'. Even though R and $\tilde{R}$ can be very different in shape, both vertices, $p^\star$ and $\tilde{p}^\star$, correspond to the same solution $x^\star$ in the polytope P, that is, $p^\star = \pi(x^\star)$ and $\tilde{p}^\star = \tilde{\pi}(x^\star)$. Let us remark that Lemma 9 is a significant generalization of Lemma 4.3 of [4].

Proof. We consider a linear auxiliary function $\bar{w}_2 \colon \mathbb{R}^n \to \mathbb{R}$, given by $\bar{w}_2(x) = \tilde{w}_2^{\mathrm{T}} x - \gamma \cdot b_i$. The paths $\bar{R} = R_{w_1,\bar{w}_2}$ and $\tilde{R}$ are identical except for a shift by $-\gamma \cdot b_i$ in the second coordinate because for $\bar{\pi} = \pi_{w_1,\bar{w}_2}$ we obtain

$$\bar{\pi}(x) = (w_1^{\mathrm{T}} x, \tilde{w}_2^{\mathrm{T}} x - \gamma \cdot b_i) = (w_1^{\mathrm{T}} x, \tilde{w}_2^{\mathrm{T}} x) - (0, \gamma \cdot b_i) = \tilde{\pi}(x) - (0, \gamma \cdot b_i)$$

for all $x \in \mathbb{R}^n$. Consequently, the slopes of $\bar{R}$ and $\tilde{R}$ are exactly the same.

Let $x \in P$ be an arbitrary point from the polytope P. Then, $\tilde{w}_2^{\mathrm{T}} x = w_2^{\mathrm{T}} x + \gamma \cdot a_i^{\mathrm{T}} x \leq w_2^{\mathrm{T}} x + \gamma \cdot b_i$. The inequality is due to $\gamma \geq 0$ and $a_i^{\mathrm{T}} x \leq b_i$ for all $x \in P$. Equality holds, among others, for $x = x^\star$ due to the choice of a_i. Hence, for all points $x \in P$ the two-dimensional points $\pi(x)$ and $\bar{\pi}(x)$ agree in the first coordinate while the second coordinate of $\pi(x)$ is at least the second coordinate of $\bar{\pi}(x)$ as $\bar{w}_2(x) = \tilde{w}_2^{\mathrm{T}} x - \gamma \cdot b_i \leq w_2^{\mathrm{T}} x$. Additionally, we have $\pi(x^\star) = \bar{\pi}(x^\star)$. Thus, path $\bar{R}$ is below path R but they meet at point $p^\star = \pi(x^\star)$. Hence, the slope of $\bar{R}$ to the left (right) of $p^\star$ is at least (at most) the slope of R to the left (right) of $p^\star$ which is greater than (at most) t. Consequently, $p^\star$ is the left-most vertex of $\bar{R}$ whose slope does not exceed t. Since $\bar{R}$ and $\tilde{R}$ are identical up to a shift of $-(0, \gamma \cdot b_i)$, $\tilde{\pi}(x^\star)$ is the left-most vertex of $\tilde{R}$ whose slope does not exceed t, i.e., $\tilde{\pi}(x^\star) = \tilde{p}^\star$. □

Lemma 9 holds for any vector $\tilde{w}_2$ on the ray $\vec{r} = \{w_2 + \gamma \cdot a_i : \gamma \geq 0\}$. As $\|w_2\| \leq n$ (see Section 2.2), we have $w_2 \in [-n, n]^n$. Hence, ray $\vec{r}$ intersects the boundary of $[-n, n]^n$ in a unique point z. We choose $\tilde{w}_2 = \tilde{w}_2(w_2, i) := z$ and obtain the following result.

Corollary 10. *Let $\tilde{\pi} = \pi_{w_1,\tilde{w}_2(w_2,i)}$ and let $\tilde{p}^\star$ be the left-most vertex of path $\tilde{R} = R_{w_1,\tilde{w}_2(w_2,i)}$ whose slope does not exceed t. Then, $\tilde{p}^\star = \tilde{\pi}(x^\star)$.*

Note, that Corollary 10 only holds for the right choice of index $i = i(x^\star, \hat{x})$. The vector $\tilde{w}_2(w_2, i)$ is defined for any vector $w_2 \in [-n, n]^n$ and any index $i \in [m]$. In the remainder, index i is an arbitrary index from $[m]$.

We can now define the following event that is parameterized in i, t, and a real $\varepsilon > 0$ and that depends on w_1 and w_2.

Definition 11. *For an index $i \in [m]$ and a real $t \geq 0$ let $\tilde{p}^\star$ be the left-most vertex of $\tilde{R} = R_{w_1,\tilde{w}_2(w_2,i)}$ whose slope does not exceed t and let $y^\star$ be the corresponding vertex of P. For a real $\varepsilon > 0$ we denote by $\mathbb{E}_{i,t,\varepsilon}$ the event that the conditions (1) $a_i^{\mathrm{T}} y^\star = b_i$ and (2) $\frac{w_2^{\mathrm{T}}(\hat{y} - y^\star)}{w_1^{\mathrm{T}}(\hat{y} - y^\star)} \in (t, t + \varepsilon]$, where $\hat{y}$ is the neighbor y of $y^\star$ for which $a_i^{\mathrm{T}} y < b_i$, are met. Note, that the vertex $\hat{y}$ always exists and that it is unique since the polytope P is non-degenerate.*

Let us remark that the vertices $y^\star$ and $\hat{y}$, which depend on the index i, equal $x^\star$ and $\hat{x}$ if we choose $i = i(x^\star, \hat{x})$. In general, this is not the case.

Observe that all possible realizations of w_2 from $L := \{w_2 + x \cdot a_i : x \in \mathbb{R}\}$ are mapped to the same vector $\tilde{w}_2(w_2, i)$. Consequently, if w_1 is fixed and if we only consider realizations of μ for which $w_2 \in L$, then vertex $\tilde{p}^\star$ and, hence, vertex $y^\star$ from Definition 11 are already determined. However, since w_2 is not completely specified, we have some randomness left for event $\mathbb{E}_{i,t,\varepsilon}$ to occur. This allows us to bound the probability of event $\mathbb{E}_{i,t,\varepsilon}$ from above (see proof of Lemma 13). The next lemma shows why this probability matters.

Lemma 12. *For reals $t \geq 0$ and $\varepsilon > 0$ let $\mathbb{A}_{t,\varepsilon}$ denote the event that the path $R = R_{w_1,w_2}$ has a slope in $(t, t+\varepsilon]$. Then, $\mathbb{A}_{t,\varepsilon} \subseteq \bigcup_{i=1}^{m} \mathbb{E}_{i,t,\varepsilon}$.*

Lemma 13. *For reals $t \geq 0$ and $\varepsilon > 0$ the probability of event $\mathbb{A}_{t,\varepsilon}$ is bounded by $\mathbf{Pr}[\mathbb{A}_{t,\varepsilon}] \leq \frac{4mn^2\varepsilon}{\delta^2}$.*

Proof. Due to Lemma 12 it suffices to show $\mathbf{Pr}[\mathbb{E}_{i,t,\varepsilon}] \leq \frac{1}{m} \cdot \frac{4mn^2\varepsilon}{\delta^2} = \frac{4n^2\varepsilon}{\delta^2}$. We apply the principle of deferred decisions and assume that vector $\lambda \in (0,1]^n$ is not random anymore, but arbitrarily fixed. Thus, vector w_1 is already fixed. Now we extend the normalized vector a_i to an orthonormal basis $\{q_1, \ldots, q_{n-1}, a_i\}$ of $\mathbb{R}^n$ and consider the random vector $(Y_1, \ldots, Y_{n-1}, Z)^{\mathrm{T}} = Q^{\mathrm{T}} w_2$ given by the matrix vector product of the transpose of the orthogonal matrix $Q = [q_1, \ldots, q_{n-1}, a_i]$ and the vector $w_2 = [v_1, \ldots, v_n] \cdot \mu$. For fixed values $y_1, \ldots, y_{n-1}$ let us consider all realizations of μ such that $(Y_1, \ldots, Y_{n-1}) = (y_1, \ldots, y_{n-1})$. Then, w_2 is fixed up to the ray $w_2(Z) = Q \cdot (y_1, \ldots, y_{n-1}, Z)^{\mathrm{T}} = \sum_{j=1}^{n-1} y_j \cdot q_j + Z \cdot a_i = w + Z \cdot a_i$ for $w = \sum_{j=1}^{n-1} y_j \cdot q_j$. All realizations of $w_2(Z)$ that are under consideration are mapped to the same value $\tilde{w}_2$ by the function $w_2 \mapsto \tilde{w}_2(w_2, i)$, i.e., $\tilde{w}_2(w_2(Z), i) = \tilde{w}_2$ for any possible realization of Z. In other words, if $w_2 = w_2(Z)$ is specified up to this ray, then the path $R_{w_1, \tilde{w}_2(w_2,i)}$ and, hence, the vectors $y^\star$ and $\hat{y}$ used for the definition of event $\mathbb{E}_{i,t,\varepsilon}$, are already determined.

Let us only consider the case that the first condition of event $\mathbb{E}_{i,t,\varepsilon}$ is fulfilled. Otherwise, event $\mathbb{E}_{i,t,\varepsilon}$ cannot occur. Thus, event $\mathbb{E}_{i,t,\varepsilon}$ occurs iff

$$(t, t+\varepsilon] \ni \frac{w_2^{\mathrm{T}} \cdot (\hat{y} - y^\star)}{w_1^{\mathrm{T}} \cdot (\hat{y} - y^\star)} = \underbrace{\frac{w^{\mathrm{T}} \cdot (\hat{y} - y^\star)}{w_1^{\mathrm{T}} \cdot (\hat{y} - y^\star)}}_{=:\alpha} + Z \cdot \underbrace{\frac{a_i^{\mathrm{T}} \cdot (\hat{y} - y^\star)}{w_1^{\mathrm{T}} \cdot (\hat{y} - y^\star)}}_{=:\beta} .$$

The next step in this proof will be to show that the inequality $|\beta| \geq \frac{\delta}{n}$ is necessary for event $\mathbb{E}_{i,t,\varepsilon}$ to happen. For the sake of simplicity let us assume that $\|\hat{y} - y^\star\| = 1$ since β is invariant under scaling. If event $\mathbb{E}_{i,t,\varepsilon}$ occurs, then $a_i^{\mathrm{T}} y^\star = b_i$, $\hat{y}$ is a neighbor of $y^\star$, and $a_i^{\mathrm{T}} \hat{y} \neq b_i$. That is, by Lemma 5, Claim 3 we obtain $|a_i^{\mathrm{T}} \cdot (\hat{y} - y^\star)| \geq \delta \cdot \|\hat{y} - y^\star\| = \delta$ and, hence,

$$|\beta| = \left| \frac{a_i^{\mathrm{T}} \cdot (\hat{y} - y^\star)}{w_1^{\mathrm{T}} \cdot (\hat{y} - y^\star)} \right| \geq \frac{\delta}{|w_1^{\mathrm{T}} \cdot (\hat{y} - y^\star)|} \geq \frac{\delta}{\|w_1\| \cdot \|\hat{y} - y^\star)\|} \geq \frac{\delta}{n \cdot 1} .$$

Summarizing the previous observations we can state that if event $\mathbb{E}_{i,t,\varepsilon}$ occurs, then $|\beta| \geq \frac{\delta}{n}$ and $\alpha + Z \cdot \beta \in (t, t+\varepsilon] \subseteq [t-\varepsilon, t+\varepsilon]$. Hence,

$$Z \in \left[\frac{t-\alpha}{\beta} - \frac{\varepsilon}{|\beta|}, \frac{t-\alpha}{\beta} + \frac{\varepsilon}{|\beta|}\right] \subseteq \left[\frac{t-\alpha}{\beta} - \frac{\varepsilon}{\frac{\delta}{n}}, \frac{t-\alpha}{\beta} + \frac{\varepsilon}{\frac{\delta}{n}}\right] =: I(y_1, \ldots, y_{n-1}) .$$

Let $\mathbb{B}_{i,t,\varepsilon}$ denote the event that Z falls into the interval $I(Y_1, \ldots, Y_{n-1})$ of length $\frac{2n\varepsilon}{\delta}$. We showed that $\mathbb{E}_{i,t,\varepsilon} \subseteq \mathbb{B}_{i,t,\varepsilon}$. Consequently,

$$\mathbf{Pr}\left[\mathbb{E}_{i,t,\varepsilon}\right] \leq \mathbf{Pr}\left[\mathbb{B}_{i,t,\varepsilon}\right] \leq \frac{2n \cdot \frac{2n\varepsilon}{\delta}}{\delta(Q^{\mathrm{T}}v_1, \ldots, Q^{\mathrm{T}}v_n)} \leq \frac{4n^2\varepsilon}{\delta^2} ,$$

where the second inequality is due to first claim of Theorem 15: By definition, we have $(Y_1, \ldots, Y_{n-1}, Z)^{\mathrm{T}} = Q^{\mathrm{T}}w_2 = Q^{\mathrm{T}} \cdot [v_1, \ldots, v_n] \cdot \mu = [Q^{\mathrm{T}}v_1, \ldots, Q^{\mathrm{T}}v_n] \cdot \mu$. The third inequality stems from $\delta(Q^{\mathrm{T}}v_1, \ldots, Q^{\mathrm{T}}v_n) = \delta(v_1, \ldots, v_n) \geq \delta$, where the equality is due to the orthogonality of Q (Claim 2 of Lemma 5). □

Lemma 14. *Let Y be the number of slopes of $R = R_{w_1,w_2}$ that lie in the interval $(0, 1]$. Then,* $\mathbf{E}[Y] \leq \frac{4mn^2}{\delta^2}$.

Proof (Theorem 1). Lemma 14 bounds only the expected number of edges on the path R that have a slope in the interval $(0, 1]$. However, the lemma can also be used to bound the expected number of edges whose slope is larger than 1. For this, one only needs to exchange the order of the objective functions $w_1^{\mathrm{T}}x$ and $w_2^{\mathrm{T}}x$ in the projection π. Then any edge with a slope of $s > 0$ becomes an edge with slope $\frac{1}{s}$. Due to the symmetry in the choice of w_1 and w_2, Lemma 14 can also be applied to bound the expected number of edges whose slope lies in $(0, 1]$ for this modified projection, which are exactly the edges whose original slope lies in $[1, \infty)$.

Formally we can argue as follows. Consider the vertices $x_1' = x_2$ and $x_2' = x_1$, the directions $w_1' = -w_2$ and $w_2' = -w_1$, and the projection $\pi' = \pi_{w_1',w_2'}$, yielding a path R' from $\pi'(x_1')$ to $\pi'(x_2')$. Let X be the number of slopes of R and let Y and Y' be the number of slopes of R and of R', respectively, that lie in the interval $(0, 1]$. The paths R and R' are identical except for the linear transformation $(x, y) \mapsto (-y, -x)$. Consequently, s is a slope of R if and only if $\frac{1}{s}$ is a slope of R' and, hence, $X \leq Y + Y'$. One might expect equality here but in the unlikely case that R contains an edge with slope equal to 1 we have $X = Y + Y' - 1$. The expectation of Y is given by Lemma 14. Since this result holds for any two vertices x_1 and x_2 it also holds for x_1' and x_2'. Note, that w_1' and w_2' have exactly the same distribution as the directions the shadow vertex algorithm computes for x_1' and x_2'. Therefore, Lemma 14 can also be applied to bound $\mathbf{E}[Y']$ and we obtain $\mathbf{E}[X] \leq \mathbf{E}[Y] + \mathbf{E}[Y'] = \frac{8mn^2}{\delta^2}$. □

The proof of Corollary 2 follows from Theorem 1 and Claim 4 of Lemma 5.

6 Some Probability Theory

The following theorem is a variant of Theorem 35 from [5]. The two differences are as follows: In [5] arbitrary densities are considered. We only consider uniform distributions. On the other hand, instead of considering matrices with entries from $\{-1, 0, 1\}$ we consider real-valued square matrices. This is why the results from [5] cannot be applied directly.

Theorem 15. *Let $X_1, \dots, X_n$ be independent random variables uniformly distributed on $(0, 1]$, let $A = [a_1, \dots, a_n] \in \mathbb{R}^{n \times n}$ be an invertible matrix, let $(Y_1, \dots, Y_{n-1}, Z)^{\mathrm{T}} = A \cdot (X_1, \dots, X_n)^{\mathrm{T}}$ be the linear combinations of $X_1, \dots, X_n$ given by A, and let $I \colon \mathbb{R}^{n-1} \to \{[x, x + \varepsilon] : x \in \mathbb{R}\}$ be a function mapping a tuple $(y_1, \dots, y_{n-1})$ to an interval $I(y_1, \dots, y_{n-1})$ of length ε. Then the probability that Z lies in the interval $I(Y_1, \dots, Y_{n-1})$ can be bounded by*

$$\mathbf{Pr}\left[Z \in I(Y_1, \dots, Y_{n-1})\right] \leq \frac{2n\varepsilon}{\delta(a_1, \dots, a_n) \cdot \min_{k \in [n]} \|a_k\|}.$$

References

1. Bonifas, N., Di Summa, M., Eisenbrand, F., Hähnle, N., Niemeier, M.: On subdeterminants and the diameter of polyhedra. In: Proc. of the 28th ACM Symposium on Computational Geometry (SoCG), pp. 357–362 (2012)
2. Borgwardt, K.H.: A probabilistic analysis of the simplex method. Springer-Verlag New York, Inc., New York (1986)
3. Brightwell, G., van den Heuvel, J., Stougie, L.: A linear bound on the diameter of the transportation polytope. Combinatorica 26(2), 133–139 (2006)
4. Brunsch, T., Cornelissen, K., Manthey, B., Röglin, H.: Smoothed analysis of the successive shortest path algorithm. In: Proc. of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1180–1189 (2013)
5. Brunsch, T., Röglin, H.: Improved smoothed analysis of multiobjective optimization. In: Proc. of the 44th Annual ACM Symposium on Theory of Computing (STOC), pp. 407–426 (2012)
6. Dantzig, G.B.: Linear programming and extensions. Rand Corporation Research Study. Princeton University Press (1963)
7. Dyer, M.E., Frieze, A.M.: Random walks, totally unimodular matrices, and a randomised dual simplex algorithm. Mathematical Programming 64, 1–16 (1994)
8. Kalai, G., Kleitman, D.J.: A quasi-polynomial bound for the diameter of graphs of polyhedra. Bulletin of the AMS 26(2), 315–316 (1992)
9. Klee, V., Walkup, D.W.: The d-step conjecture for polyhedra of dimension $d < 6$. Acta Mathematica 117, 53–78 (1967)
10. Naddef, D.: The hirsch conjecture is true for (0, 1)-polytopes. Mathematical Programming 45, 109–110 (1989)
11. Orlin, J.B.: A polynomial time primal network simplex algorithm for minimum cost flows. Mathematical Programming 78(2), 109–129 (1997)
12. Santos, F.: A counterexample to the hirsch conjecture. CoRR, abs/1006.2814 (2010)
13. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. Journal of the ACM 51(3), 385–463 (2004)

On Randomized Online Labeling with Polynomially Many Labels*

Jan Bulánek[1,2,**], Michal Koucký[2,***], and Michael Saks[3,†]

[1] Department of Theoretical Computer Science and Mathematical Logic, Charles University, Prague
[2] Institute of Mathematics, Academy of Sciences CR, Prague
[3] Department of Mathematics, Rutgers University

Abstract. We prove an optimal lower bound on the complexity of *randomized* algorithms for the online labeling problem with polynomially many labels. All previous work on this problem (both upper and lower bounds) only applied to deterministic algorithms, so this is the first paper addressing the (im)possibility of faster randomized algorithms. Our lower bound $\Omega(n \log(n))$ matches the complexity of a known deterministic algorithm for this setting of parameters so it is asymptotically optimal.

In the online labeling problem with parameters n and m we are presented with a sequence of n *items* from a totally ordered universe U and must assign each arriving item a label from the label set $\{1, 2, \ldots, m\}$ so that the order of labels (strictly) respects the ordering on U. As new items arrive it may be necessary to change the labels of some items; such changes may be done at any time at unit cost for each change. The goal is to minimize the total cost. An alternative formulation of this problem is the *file maintenance problem*, in which the items, instead of being labeled, are maintained in sorted order in an array of length m, and we pay unit cost for moving an item.

1 Introduction

In the online labeling problem with parameters n, m, r, we are presented with a sequence of n *items* from a totally ordered universe U of size r and must assign each arriving item a label from the label set $\{1, 2, \ldots, m\}$ so that the order of labels (strictly) respects the ordering on U. As new items arrive it may be necessary to change the labels of some items; such changes may be done at any time at unit cost for each change. The goal is to minimize the total cost. An alternative formulation of this problem is the *file maintenance problem*, in which the items, instead of being labeled, are maintained in sorted order in an array of length m, and we pay unit cost for moving an item.

* Full version of the paper is available from the second author's homepage.

** Partially supported by student project GAUK 344711 and RVO: 67985840.

*** Supported in part by grant IAA100190902 of GA AV ČR, Center of Excellence CE-ITI (P202/12/G061 of GA ČR) and RVO: 67985840.

† The work of this author was done while on sabbatical at Princeton University and was also supported in part by NSF under grant CCF-0832787 and CCF-1218711.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 291–302, 2013.

The problem, which was introduced by Itai, Konheim and Rodeh [13], is natural and intuitively appealing, and has had applications to the design of data structures (see for example the discussion in [10], and the more recent work on cache-oblivious data structures [4,8,5]). A connection between this problem and distributed resource allocation was recently shown by Emek and Korman [12].

The parameter m, the size of the *label space* must be at least the number of items n or else no valid labeling is possible. There are two natural ranges of parameters which have received the most attention. In the case of *linearly many labels* we have $m = cn$ for some $c > 1$, and in the case of *polynomially many labels* we have $m = \theta(n^C)$ for some constant $C > 1$. The size r of the universe U is also a parameter which is not discussed explicitly in most of the literature on the paper. If $r \leq m$, since then we can simply fix an order preserving bijection from U to $\{1, \ldots, m\}$ in advance. In this paper we assume $U = \{1, \ldots, 2^n\}$.

Itai et al. [13] gave an algorithm for the case of linearly many labels having worst case total cost $O(n \log(n)^2)$. Improvements and simplifications were given by Willard [15] and Bender et al. [3]. In the special case that $m = n$, algorithms with cost $O(\log(n)^3)$ per item are known [16,6]. It is also well known that the algorithm of Itai et al. can be adapted to give total cost $O(n \log(n))$ in the case of polynomially many labels. All of these algorithms are deterministic.

In a previous paper [9], we proved a $\Omega(n \log(n)^2)$ lower bound in the case of linearly many labels, and $\Omega(n \log(n)^3)$ lower bound for the case $m = n$. In subsequent work with Babka and Čunát [2] we proved a lower bound $\Omega(n \log(n)/(\log \log(m) - \log \log(n)))$ when $n^{1+\varepsilon} \leq m \leq 2^{n^\varepsilon}$, In particular, this gives a $\Omega(n \log(n))$ bound for the case of m being polynomial in n. These lower bounds match the known upper bounds to within a constant factor. Both of these papers built heavily on previous partial results of Dietz, Seiferas and Zhang ([16,11,10]). These lower bounds apply only to *deterministic* algorithms, leaving open the possibility of better randomized algorithms.

In this paper we use a model in which the cost of a randomized labeling algorithm is the worst case over all input sequences of a given length n of the expected number of moves made by the algorithm. This corresponds to running the algorithm against an *oblivious adversary* (see [7]) who selects the input sequence having full knowledge of the algorithm, but not of the random bits flipped in the execution of the algorithm.

There are many online problems where randomized algorithms perform provably better than deterministic ones. For example, the best deterministic algorithm for the paging problem with k pages has competitive ratio k but there are randomized algorithms having competitive ratio $\Theta(\log(k))$ [7].

Our Results. In this paper we establish the first lower bound for *randomized* online labeling algorithms by showing that in the case of polynomially many labels any randomized online labeling algorithm will have expected cost $\Omega(n \log(n))$ (for the worst case input). This matches the known deterministic upper bounds up to constant factors, and thus randomization provides no more than a constant factor advantage over determinism. Our bound also implies an $\Omega(n \log(n))$

lower bound on the message complexity of randomized protocols for Distributed Controller Problem [12,1].

Unlike many other lower bounds for non-uniform computation models, our proof does not use Yao's principle. Yao's principle says (roughly) that to prove a lower bound on the expected cost of an arbitrary randomized algorithm it suffices to fix a distribution over inputs, and prove a lower bound on the expected cost of a deterministic algorithm against the chosen distribution. Rather than use Yao's principle, our proof takes an arbitrary randomized algorithm and selects a (deterministic) sequence that is hard for that algorithm.

The construction and analysis of the hard sequence follow the same overall strategy of the previous lower bound for deterministic algorithms in the case of polynomially many labels [10,2] which involves relating online labeling to a family of one-player games called *bucketing games* (introduced in [10]) which involve the sequential placement of items into an ordered sequence of bins subject to certain rules and costs. We define a map (an *adversary*) which associates to a labeling algorithm **A** a hard sequence of items. We then show that the behavior of the algorithm on this hard sequence can be associated to a strategy for playing a particular bucketing game, such that the cost incurred by the algorithm on the hard sequence is bounded below by the cost of the associated bucketing game strategy. Finally we prove a lower bound on the cost of any strategy for the bucketing game, which therefore gives a lower bound on the cost of the algorithm on the hard input sequence.

In extending this argument from the case of deterministic algorithms to the randomized case, each part of the proof requires significant changes. The adversary which associates an algorithm to a hard sequence requires various careful modifications. The argument that relates the cost of **A** on the hard sequence to the cost of an associated bucketing strategy does not work for the original version of the bucketing game, and we can only establish the connection to a new variant of the bucketing game called tail-bucketing. Finally the lower bound proof on the cost of any strategy for tail-bucketing is quite different from the previous lower bound for the original version of bucketing.

Mapping a Randomized Algorithm to a Hard Input Sequence. We now give an overview of the adversary which maps an algorithm to a hard input sequence $y_1, \ldots, y_n$. The adversary is deterministic. Its behavior will be determined by the expected behavior of the randomized algorithm. Even though we are choosing the sequence obliviously, without seeing the actual responses of the algorithm, we view the selection of the sequence in an online manner. We design the sequence item by item. Having selected the first $t-1$ items, we use the known randomized algorithm to determine a probability distribution over the sequence of labelings determined by the algorithm after each step. We then use this probability distribution to determine the next item, which we select so as to ensure that the expected cost incurred by the algorithm is large.

The adversary will maintain a hierarchy consisting of a nested sequence subsets of the set of items inserted so far. The hierarchy at step t is used to determine the item to be inserted at step p. This hierarchy is denoted

$$S_t(1) \supset T_t(2) \supset S_t(2) \supset T_t(3) \supset \cdots \supset T_t(d) \supset S_t(d).$$

The set $S_t(1)$ consists of all items inserted through step $t-1$. Each of the other sets is an interval relative to the items inserted so far, i.e., it consists of all inserted items in a given interval. The final subset $S_t(d)$ has between 2 and 6 elements. (Note that this differs from the previous work for deterministic algorithms where the hierarchy was a nested sequence of intervals of label values rather than items; this modification seems necessary to handle randomized algorithms). The next item to be inserted is selected to be an item that is between two items in the final set $S_t(d)$.

The hierarchy at step t is constructed as follows. The hierarchy for $t = 1$ has $d_1 = 1$ and $S_1(1) = \{0, 2^n\}$. The hierarchy at step $t > 1$ is constructed based on the hierarchy at the previous step $t-1$ and the expected behavior of the algorithm on $y_1, \ldots, y_{t-1}$.

We build the sets for the hierarchy at step t in order of increasing level (i.e., decreasing size). Intervals are either *preserved* (carried over from the previous hierarchy, with the addition of y_{t-1}) or *rebuilt*. To specify which intervals are preserved, we specify a *critical level for step* t, q_t which is at most the depth d_{t-1} of the previous hierarchy. We'll explain the choice of q_t below. At step t, the intervals $T_t(i)$ and $S_t(i)$ for $i \leq q_t$ are *preserved*, which means that it is obtained from the corresponding interval at step $t-1$ by simply adding y_{t-1}. The intervals $T_t(i)$ and $S_t(i)$ for $i \geq q_t$ are *rebuilt*. The rule for rebuilding the hierarchy for $i > q_t$ is defined by induction on i as follows: Given $S_t(i-1)$, $T_t(i)$ is defined to be either the first or second half of $S_t(i-1)$, depending on which of these sets is more likely to have a smaller range of labels (based on the distribution over labels determined by the given algorithm). More precisely, we look at the median item of $S_t(i)$ and check whether (based on the randomized labeling) it is more likely that its label is closer to the label of the minimum or to the maximum element of $S_t(i)$. If the median is more likely to have label close to the minimum we pick the first half as $T_t(i)$ otherwise the second half. Having chosen $T_t(i)$, we take $S_t(i)$ to be the middle third of items in $T_t(i)$. This process terminates when $|S_t(i)| < 7$ and the depth d_t of the hierarchy is set to this final i. The adversary selects the next requested item y_t to be between two items in $S_t(d)$.

This construction of the hierarchy is similar to that used in [2] in the deterministic case. An important difference comes in the definition of the critical level q_t. In the deterministic case the critical level is the smallest index i such that neither endpoint of $T_{t-1}(i)$ was moved by the algorithm when inserting y_{t-1}. In the randomized case we need a probabilistic version of this: the critical level is the smallest index i such that the probability that either endpoint of $T(i)$ was moved since the last time it was rebuilt is less than $1/4$.

One of the crucial requirements in designing the adversary is that the hierarchy never grows too deep. Note that when we rebuild $T_t(i)$ it's size is at most $|S_t(i-1)|/2$ and when we rebuild $S_t(i)$ its size is at most $|T_t(i)|/3$. This suggests that as we proceed through the hierarchy each set is at most $1/2$ the size of the previous

and so the depth is at most $\log(n)$. This is not true since during a sequence of steps in which a set in the hierarchy is not rebuilt its size grows by 1 at each step and so the condition that the set is at most half the size of its predecessor may not be preserved. Nevertheless we can show that the depth never grows to more than $4\log(m+1)$ levels.

A Lower Bound on the Expected Cost of the Algorithm on the Hard Sequence. In [9] it is noted that we can assume without loss of generality that the algorithm is *lazy*, in the sense that for each step the set of relabeled items is a sub-interval of the inserted items that contains the most recently inserted item. (Intuitively, a non-lazy algorithm can be modified so that any relabeling that violates laziness is deferred until later). This observation extends to randomized algorithms.

In the deterministic case, this assumption and the definition of the critical level q_{t+1} can be used to show that when the algorithm responds to the item y_t it moved at least a constant fraction of the items belonging to $S_t(q_{t+1}+1)$ must have moved at step $t-1$ and so the total cost of the algorithm is at least $\Omega(\sum_t |S_t(q_{t+1}+1)|)$. In the randomized case we get a related bound that the expected total number of items is $\Omega(\sum_t |S_t(q_{t+1}) - S_t(q_{t+1}+1)|)$.

Bucketing Games. The next step in the analysis is to define bucketing games, and to show that the lower bound on the cost of the algorithm given in the previous paragraph is an upper bound on the cost of an appropriate bucketing game.

The prefix bucketing game with n items and k buckets is a one player game. The game starts with k empty buckets indexed $1,\ldots,k$. At each step the player places an item in some bucket p. All the items from buckets $1,\ldots,p-1$ are then moved into bucket p as well, and the cost is the number of items in buckets $1,\ldots,p$ before the merge, which is the number of items in bucket p after the merge. The goal is to select the sequence of indices so as to minimize the total cost. The total cost is the sum of the costs of each step. The goal is to select the sequence of indexes p so that we would minimize the total cost. In [2] (following [10]) it is shown that any deterministic labeling algorithm could be associated to a bucketing strategy such that the cost of the labeling algorithm against our adversary is at least a constant times the cost of the bucketing strategy. This result is deduced using the lower bound of $\Omega(\sum_t |S_t(q_{t+1}+1)|$ for the cost of the algorithm mentioned earlier. It was also shown in [10] (see also [2]) that the minimal cost of any bucketing strategy (for more than $2\log(n)$ buckets) is $\Omega(n\log(n)/(\log(k)-\log\log(n))$. These results together gave the lower bound on deterministic labeling.

We use the same basic idea for the randomized case, but require several significant changes to the game. The first difficulty is that the lower bound $\Omega(\sum_t |S_t(q_{t+1}) - S_t(q_{t+1}+1)|)$ on the cost of a randomized algorithm in terms of a sum given earlier is not the same as the lower bound we had for deterministic algorithms. This lower bound is no longer bounded below by the cost of prefix bucketing. To relate this lower bound to bucketing, we must replace the

cost function in bucketing by a smaller cost function, which is the number of items in the bucket p *before* the merge, not after. In general, this cost function is less expensive (often much less expensive) than the original cost function and we call it the *cheap* cost function. The argument relating the cost of a randomized algorithm to a bucketing strategy requires that the number of buckets be at least $4\log(m)$ buckets. If we could prove a lower bound on the cost of bucketing under the cheap function similar to the bound mentioned above for the original function this would be enough to deduce the desired lower bound on randomized labeling. However with this cheap cost function this lower bound fails: if the number of buckets is at least $1 + \log(n)$, there is a bucketing strategy that costs 0 with the cheap cost function! So this will not give any lower bound on cost of a randomized labeling algorithm

We overcome this problem by observing that we may make a further modification of the rules for bucketing and still preserve the connection between the cost of a randomized algorithm against our adversary and the cheap cost of a bucketing. This modification is called *tail bucketing*. In a tail bucketing, after merging all the items into the bucket p, we redistribute these items back among buckets $1, \ldots, p$, so that bucket p keeps $1 - \beta$ fraction of the items and passes the rest to the bucket $p - 1$, bucket $p - 1$ does the same, and the process continues down until bucket 1 which keeps the remaining items. It turns our that our adversary can be related to tail bucketing for $\beta = 1/6$. We can prove that the minimal *cheap* cost of tail bucketing is $\Omega(n \log(n))$ when $k = O(\log n)$. This lower bound is asymptotically optimal and yields a similar bound for randomized online labeling.

The lower bound proof for the cheap cost of tail bucketing has some interesting twists. The proof consists of several reductions between different versions of bucketing. The reductions show that we can lower bound the cheap cost of tail bucketing with $C\log(n)$ buckets (for any C) by the cheap cost of ordinary prefix bucketing with $k = \frac{1}{4}\log n$ buckets. Even though the cheap cost of ordinary bucketing dropped to 0 once $k = \log(n) + 1$, we are able to show that for $k = \frac{1}{4}\log(n)$ there is a $\theta(n\log(n))$ bound for ordinary bucketing with the cheap cost.

Due to space limitations large portion of the proofs are omitted and they will appear in the full version. Unless otherwise specified, logarithms in this paper are to base 2.

2 The Online Labeling Problem

We first define the deterministic version of online labeling. We have parameters $n \leq m < r$, and are given a sequence of n numbers from the set $U = [1, r]$ and must assign to each of them a label in the range $[1, m]$. (Here, and throughout the paper, interval notation is used for consecutive sets of integers). A *deterministic* online labeling algorithm A with parameters (n, m, r) is an algorithm that on input sequence $(y_1, y_2, \ldots, y_t)$ with $t \leq n$ of distinct elements from U outputs a *labeling* $f_A : \{y_1, y_2, \ldots, y_t\} \to [m]$ that respects the natural ordering

of $y_1, \ldots, y_t$, that is for any $x, y \in \{y_1, y_2, \ldots, y_t\}$, $f_A(x) < f_A(y)$ if and only if $x < y$. We refer to $y_1, y_2, \ldots, y_t$ as *items*.

Fix an algorithm A. Any item sequence $y_1, \ldots, y_n$ determines a sequence $f_{A,0}, f_{A,1}, \ldots, f_{A,n}$ of labelings where f_t is the labeling of $(y_1, \ldots, y_t)$ determined by A. When the algorithm A is fixed we omit the subscript A. We say that A *relabels* $y \in \{y_1, y_2, \ldots, y_t\}$ *at time* t if $f_{t-1}(y) \neq f_t(y)$. In particular, y_t is relabeled at time t. $Rel_t = Rel_{A,t}$ denotes the set of items relabeled at step t. The cost of A on $y_1, y_2, \ldots, y_n$ is $\chi_A(y_1, \ldots, y_n) = \sum_{t=1}^{n} |Rel_t|$.

A *randomized* online labeling algorithm $\mathbf{A}$ is a probability distribution on deterministic online labeling algorithms. Given a item sequence $y_1, \ldots, y_n$, the algorithm $\mathbf{A}$ determines a probability distribution over sequences of labelings $f_0, \ldots, f_n$. The set Rel_t is a random variable whose value is a subset of $y_1, \ldots, y_t$. The cost of $\mathbf{A}$ on $y_1, y_2, \ldots, y_n \in U$ is the expected cost $\chi_{\mathbf{A}}(y_1, \ldots, y_n) = \mathbf{E}[\chi_A(y_1, \ldots, y_n)]$. The maximum cost $\chi_{\mathbf{A}}(y_1, \ldots, y_n)$ over all sequences $y_1, \ldots, y_n$ is denoted $\chi_{\mathbf{A}}(n)$. We write $\chi_m(n)$ for the smallest cost $\chi_{\mathbf{A}}(n)$ that can be achieved by any algorithm $\mathbf{A}$ with range m.

We state our main theorem.

Theorem 1. *For any constant C_0, there are positive constants C_1 and C_2 so that the following holds. Let $\mathbf{A}$ be a randomized algorithm with parameters (n, m, r), where $n \geq C_1$, $r \geq 2^n$ and $m \leq n^{C_0}$. Then $\chi_{\mathbf{A}}(n) \geq C_2 n \log(n)$.*

To prove the theorem we will need some additional definitions. Let $S \subseteq Y \subseteq U$. We write $\min(S)$ and $\max(S)$ for the least and greatest elements, respectively. We say that S is a *Y-interval* if $S = Y \cap [\min(S), \max(S)]$. We write $\mathrm{med}(S)$ for the *median* of S which we take to be the $\lceil |S|/2 \rceil$-th largest element of S. We define **left-half**$(S) = \{y \in S | y \leq \mathrm{med}(S)\}$ and **right-half**$(S) = \{y \in S | y \geq \mathrm{med}(S)\}$ (note that $\mathrm{med}(S)$ is contained in both). Also define **left-third**(S) to be the smallest $\lfloor |S|/3 \rfloor$ elements, **right-third**(S) to be the largest $\lfloor |S|/3 \rfloor$ elements and **middle-third**$(S) = S -$ **left-third**$(S) -$ **right-third**(S).

Given a labeling f of Y and a Y-interval S, we say that the Y-interval S is *left-leaning* with respect to f if $\mathrm{med}(S)$ has a label that is closer to the label of $\min(S)$ than it is to the label of $\max(S)$, i.e. $(f(\mathrm{med}(S)) - f(\min(S))) \leq (f(\max(S)) - f(\mathrm{med}(S))$. It is *right-leaning* otherwise.

A deterministic labeling algorithm is *lazy* if at each step t, the set of relabeled items is a Y_t-interval (which necessarily contains y_t), and a randomized algorithm is lazy if it is a distribution over lazy deterministic algorithms. In [9], it was shown that there is an optimal deterministic algorithm that is lazy, and the same proof works to show that there is an optimal lazy randomized algorithm. (Intuitively this is the case because if the relabeled items at step t do not form a Y_t-interval and W is the largest Y_t-interval of relabeled items containing y_t then we can defer relabeling the items in $Y_t \setminus W$ until later.)

2.1 The Adversary

We now specify an adversary **Adversary**$(\mathbf{A}, n, m)$ which given an online labeling algorithm $\mathbf{A}$, a length n, and label space size m, constructs a item sequence

$y_1, y_2, \ldots, y_n$ from the universe $U = \{1, \ldots, 2^n - 1\}$. Our adversary and notation borrow from past work in the deterministic case ([10,9]).

We think of the adversary as selecting $y_1, \ldots, y_n$ online, but after each step the adversary only knows a probability distribution over the configurations of the algorithm.

To avoid dealing with special cases in the description of the adversary, we augment the set of items by items 0 and 2^n which are given (permanent) labels 0 and $m+1$ respectively. We write Y_t for the set $\{y_1, \ldots, y_t\} \cup \{0, 2^n\}$ of labeled items after step t. At the beginning of step t, having chosen the set Y_{t-1}, the adversary will select a Y_{t-1}-interval, denoted $S_t(*)$, of size at least 2 and select y_t to be $\min(S_t(*)) + 2^{n-t}$. An easy induction on t shows that the items chosen through step t are multiples of 2^{n-t}, and it follows that y_t is strictly between the smallest and second smallest elements of $S_t(*)$. Therefore all of the chosen items are distinct.

To choose $S_t(*)$, the adversary constructs a nested sequence of sets:

$$Y_{t-1} = S_t(1) \supset T_t(2) \supset S_t(2) \supset T_t(3) \supset \cdots \supset T_t(d_t) \supset S_t(d_t)$$

called the *hierarchy at step t*, and chooses $S_t(*) = S_t(d_t)$.

Note that the subscript for the hierarchy is one larger than the subscript of the containing set Y_{t-1} because the hierarchy is built in step t in order to determine y_t. Each set $S_t(i)$ and $T_t(i)$ is a Y_{t-1}-interval of size at least 2. The depth d_t of the hierarchy may vary with t. The sets $S_t(i)$ and $T_t(i)$ are said to be at *level i* in the hierarchy.

The hierarchy for $t = 1$ has $d_1 = 1$ and $S_1(1) = \{0, 2^n\}$. The hierarchy at step $t > 1$ is constructed based on the hierarchy at the previous step $t - 1$ and the expected behavior of the algorithm on $y_1, \ldots, y_{t-1}$ as reflected by the joint probability distribution over the sequence of functions $f_1, \ldots, f_{t-1}$.

The pseudo-code for the adversary is given in Figure 1, and follows the informal description of the adversary sketched in the introduction.

Here is a more detailed explanation of how the critical level q_t is selected. When constructing each set $S_t(i)$ of the hierarchy for $i \geq 2$, the adversary defines a parameter $\mathbf{birth}_t(i)$ which is set equal to t if $S_t(i)$ is rebuilt, and is otherwise set to $\mathbf{birth}_{t-1}(i)$. It is easy to see (by induction on t), that $\mathbf{birth}_t(i)$ is equal to the largest time $u \leq t$ such that $S_u(i)$ was rebuilt. It follows that for each $u \in [\mathbf{birth}_t(i), t]$, $\min(T_u(i)) = \min(T_t(i))$ and $\max(T_u(i)) = \max(T_t(i))$.

Say that item y has *stable label* during interval $[a, b]$ if for each step u in $[a, b]$, $f_u(y)$ has the same value, and has unstable label on $[a, b]$ otherwise. We define the event $\mathbf{stable}_t(i)$ to be the event (depending on $\mathbf{A}$) that $\min(T_t(i))$ and $\max(T_t(i))$ have stable labels during interval $[\mathbf{birth}_t(i-1), t]$.

We are finally ready to define q_t. If there is at least one level $i \geq 2$ for which $\mathbf{Pr}[\mathbf{stable}_{t-1}(i)] \leq 3/4$, let $i_{\min}$ be the least such level, and choose $q_t = i_{\min} - 1$. Otherwise set $q_t = d_{t-1}$.

The following lemma immediately implies Theorem 1.

Lemma 1. *Let $n \leq m$ be integers. Let $\mathbf{A}$ be a lazy randomized online labeling algorithm with the range m. Let $y_1, y_2, \ldots, y_n$ be the output of* **Adversary**$(\mathbf{A}, n, m)$. *Then the cost satisfies:*

$$\chi_{\mathbf{A}}(y_1, y_2, \ldots, y_n) \geq \frac{5}{96}\left(\frac{1}{6}\right)^{2^8 c \log(2c)} (n+1)\log(n+1) - \frac{n}{4},$$

where $c = \log(m+1)/\log(n+1)$.

The proof of this lemma has two main steps. The first step is to bound the cost $\chi_{\mathbf{A}}(y_1, \ldots, y_n)$ from below by the minimum cost of a variant of the prefix-bucketing game. The prefix-bucketing game was introduced and studied before to get lower bounds for deterministic online labeling. The variant we consider is called *tail-bucketing*. The second step is to give a lower bound on the cost of tail-bucketing.

Adversary$(\mathbf{A}, n, m)$

$t = 1$: $S_1(1) \longleftarrow \{0, 2^n\}$ and $\mathbf{birth}_1(1) = 1$, $y_1 = 2^{n-1}$.

For $t = 2, \ldots, n$ do

- $S_t(1) \longleftarrow S_{t-1}(1) \cup \{y_{t-1}\}$;
- (Choose critical level) Consider the sequence of (dependent) random functions $f_1, \ldots, f_{t-1}$ produced by $\mathbf{A}$ in response to $y_1, \ldots, y_{t-1}$. If there is an index $i \geq 2$ for which $\mathbf{Pr}[\mathbf{stable}_{t-1}(i)] \leq 3/4$, let $i_{\min}$ be the least such index and let $q_t = i_{\min} - 1$. Otherwise set $q_t = d_{t-1}$.
- $i \longleftarrow 1$.
- (Preserve intervals up to the critical level) While $i < q_t$ do:
  - $i \longleftarrow i + 1$.
  - $T_t(i) \longleftarrow T_{t-1}(i) \cup \{y_{t-1}\}$
  - $S_t(i) \longleftarrow S_{t-1}(i) \cup \{y_{t-1}\}$
  - $\mathbf{birth}_t(i) \longleftarrow \mathbf{birth}_{t-1}(i)$
- (Build intervals after the critical level) While $|S_t(i)| \geq 7$ do:
  - $i \longleftarrow i + 1$
  - If $S_t(i-1)$ is left-leaning with respect to f_{t-1} with probability at least $1/2$ then $T_t(i) \longleftarrow$ **left-half**$(S_t(i-1))$ otherwise $T_t(i) \longleftarrow$ **right-half**$(S_t(i-1))$
  - $S_t(i) \longleftarrow$ **middle-third**$(T_t(i))$.
  - $\mathbf{birth}_t(i) \longleftarrow t$. [Record that $S_t(i)$ and $T_t(i)$ were rebuilt]
- $d_t \longleftarrow i$.
- $y_t \longleftarrow \min(S_t(d_t)) + 2^{n-t}$.

Output: $y_1, y_2, \ldots, y_n$.

Fig. 1. Pseudocode for the adversary

To prove the first step we will need two properties of **Adversary**$(\mathbf{A}, n, m)$. In what follows, we fix $\mathbf{A}, n, m$. **Adversary**$(\mathbf{A}, n, m)$ determines $y_1, \ldots, y_n$ and the critical levels $q_1, \ldots, q_n$. Note that the definition of q_j only depends on the expected behavior of the algorithm through the construction of f_{j-1}. For purposes of analysis, we also define the critical level q_{n+1} based on the expected behavior of $f_1, \ldots, f_n$ in exactly the same way.

Lemma 2. *For any $t \in [1, n]$, $d_t \leq 4 \log(m + 1)$.*

Lemma 3. *The cost of* **A** *on $y_1, \ldots, y_n$ satisfies:*

$$\chi_{\mathbf{A}}(y_1, y_2, \ldots, y_n) \geq \frac{1}{40} \sum_t |S_t(q_{t+1}) \setminus S_t(1 + q_{t+1})|,$$

where the sum ranges over time steps $t \in [1, n]$ for which $q_{t+1} < d_t$.

3 Prefix Bucketing and Tail Bucketing

We will need several different variants of prefix bucketing game introduced by Dietz, Seiferas and Zhang [10]. We have k buckets numbered $1, \ldots, k$ in which items are placed. A *bucket configuration* is an arrangement of items in the buckets; formally it is a mapping $C : \{1, \ldots, k\}$ to the nonnegative integers, where $C(i)$ is the number of items in bucket i. It will sometimes be convenient to allow the range of the function C to be the nonnegative real numbers, which corresponds to allowing a bucket to contain a fraction of an item.

A *bucketing game* is a one player game in which the player is given a sequence of groups of items of sizes $n_1, \ldots, n_\ell$ and must sequentially place each group of items into a bucket. The case that the sequence $n_1 = \cdots = n_\ell = 1$ is called *simple bucketing*. The placement is done in a sequence of ℓ steps, and the player selects a sequence $p_1, \ldots, p_\ell \in [1, k]^\ell$, called an (ℓ, k)-*placement sequence* which specifies the bucket into which each group is placed. Bucketing games vary depending on two ingredients, the *rearrangement rule* and the *cost functions*.

When a group of m items is placed into bucket p, the items in the configuration are rearranged according to a specified rearrangement rule, which is not under the control of the player. Formally, a rearrangement rule is a function R that takes as input the current configuration C, the number m of new items being placed and the bucket p into which they are placed, and determines a new configuration $R(C, m, p)$ with the same total number of items.

The *prefix rearrangement rule* is as follows: all items currently in buckets below p are moved to bucket p. We say that items are *merged into bucket p*. Formally, the new configuration $C' = R(C, m, p)$ satisfies $C'(i) = 0$ for $i < p$, $C'(p) = C(1) + \cdots + C(p) + m$ and $C'(i) = C(i)$ for $i > p$. Most of the bucketing games we'll discuss use the prefix rearrangement function, but in Section 3.1 we'll need another rearrangement rule.

The *cost function* specifies a cost each time a placement is made. For the cost functions we consider the cost of placing a group depends on the current configuration C and the selected bucket p but not on the number m of items being placed. We consider four cost functions but for this extended abstract we need only the *cheap* cost function: In *cheap* bucketing, the cost is the number of items in bucket p before the placement:

$$\mathbf{cost_{cheap}}(C, p) = C(p).$$

Fix a rearrangement rule R and a cost function c. A placement sequence $p_1, \ldots, p_\ell$ and a load sequence $n_1, \ldots, n_\ell$ together determine a sequence of configurations

$B = (B_0, B_1, \ldots, B_\ell)$, called a *bucketing* where B_0 is the empty configuration and for $i \in [1, \ell]$, $B_i = R(B_{i-1}, n_i, p_i)$. Each of these ℓ placements is charged a cost according to the cost rule c. We write $c[R](p_1, \ldots, p_\ell | n_1, \ldots, n_\ell)$ for the sum $\sum_{i=1}^{\ell} c(B_{i-1}, p_i)$, which is the sum of the costs of each of the ℓ rearrangements that are done during the bucketing. If R is the prefix rule, we call B a *prefix bucketing* and denote the cost simply by $c(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell)$. In the case of simple bucketing, $n_1 = \ldots = n_\ell = 1$, we write simply $c[R](p_1, \ldots, p_\ell)$ or $c(p_1, \ldots, p_\ell)$ in the case of simple prefix bucketing.

3.1 Tail Bucketing and Online Labeling

We will need an alternative rearrangement function, called the *tail rearrangement rule*. The bucketing game with this rule is called *tail bucketing*. The tail rearrangement rule $Tail_\beta$ with parameter β acts on configuration C, bucket p and group size m by first moving all items below bucket p to bucket p so that $w = C(1) + \cdots + C(p) + m$ items are in bucket p (as with the prefix rule), but then for j from p down to 1, β fraction of the items in bucket j are passed to bucket $j-1$, until we reach bucket 1. (Here we allow the number of items in a bucket to be non-integral.) Thus for $j \in [2, p]$ bucket j has $w(1-\beta)(\beta)^{p-j}$ items and bucket 1 has $w\beta^{p-1}$ items. We will consider tail bucketing with the cheap cost function.

We now relate the expected cost of randomized online labeling algorithm $\mathbf{A}$ on the sequence $y_1, y_2, \ldots, y_n$ produced by our adversary to the cost of a specific tail bucketing instance. For a lazy online labeling algorithm $\mathbf{A}$ and $t = 1, \ldots, n$, let $f_t, S_t(i), q_t, y_t$ be as defined by our adversary and the algorithm $\mathbf{A}$. Denote $Y = \{y_1, y_2, \ldots, y_n\}$. Set $k = \lfloor 4 \log(m+1) \rfloor$. Let $q_1, \ldots, q_n$ be the sequence of critical levels produced by the algorithm. As mentioned prior to stating Lemma 2, we define q_{n+1} for analysis purposes. For integer $i \in [k]$ define $\bar{i}$ to be $\bar{i} = (k+1) - i$. Define the placement sequence $p_1 = \bar{q}_2, \ldots, p_n = \bar{q}_{n+1}$, and consider the tail bucketing $B_0, \ldots, B_{n+1}$ determined by this placement sequence with parameter $\beta = 1/6$, and all group sizes 1 (so it is a simple bucketing). The following lemma relates the cost of online labeling to the tail bucketing.

Lemma 4. *Let $\{S_t(i) : 1 \le t \le n, 1 \le i \le d_t\}$ be the interval hierarchy computed by* **Adversary(A**$, n, m)$ *and $B_{\mathbf{A}} = (B_0, \ldots, B_n)$ be the corresponding tail-bucketing. Then for any $t \in [1, n]$ and any $j \in [1, d_t]$:*

$$|S_t(j) \backslash S_t(j+1)| \ge B_{t-1}(\bar{j}) - 3.$$

Here, for the case $j = d_t$, we take $S_t(j+1)$ to be $\emptyset$.

Corollary 1. *The cost of randomized labeling algorithm $\mathbf{A}$ with label space $[1, m]$ on $y_1, \ldots, y_n$ satisfies:*

$$\chi_{\mathbf{A}}(y_1, y_2, \ldots, y_n) \ge \frac{1}{40}(\min \mathbf{cost_{cheap}}[Tail_{1/6}](p_1, \ldots, p_n) - 10n),$$

where the minimum is over all placement sequences $(p_1, \ldots, p_n)$ into $\lfloor 4 \log(m+1) \rfloor$ buckets.

Armed with Corollary 1, it now suffices to prove a lower bound on the cheap cost of simple tail bucketing when the number of items is n and the number of buckets is $\lfloor 4\log(m+1)\rfloor$. The lower bound is provided by the next statement:

Lemma 5. *Let $p_1, \ldots, p_n$ be an arbitrary placement sequence into $\lfloor 4\log(m+1)\rfloor$ buckets. Then*

$$\mathbf{cost_{cheap}}[Tail_{1/6}](p_1, \ldots, p_n) \geq \frac{5}{12}\left(\frac{1}{6}\right)^{2^8 c\log(2c)} (n+1)\log(n+1),$$

where $c = \log(m+1)/\log(n+1)$.

Now Lemma 1 follows from the above lemma and Corollary 1.

References

1. Afek, Y., Awerbuch, B., Plotkin, S., Saks, M.: Local management of a global resource in a communication network. J. ACM 43(1), 1–19 (1996)
2. Babka, M., Bulánek, J., Čunát, V., Koucký, M., Saks, M.: On Online Labeling with Polynomially Many Labels. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 121–132. Springer, Heidelberg (2012)
3. Bender, M.A., Cole, R., Demaine, E.D., Farach-Colton, M., Zito, J.: Two simplified algorithms for maintaining order in a list. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 152–164. Springer, Heidelberg (2002)
4. Bender, M., Demaine, E., Farach-Colton, M.: Cache-oblivious B-trees. SIAM J. Comput. 35(2), 341–358 (2005)
5. Bender, M., Duan, Z., Iacono, J., Wu, J.: A locality-preserving cache-oblivious dynamic dictionary. J. Algorithms 53(2), 115–136 (2004)
6. Bird, R., Sadnicki, S.: Minimal on-line labelling. Inf. Process. Lett. 101(1), 41–45 (2007)
7. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press (1998)
8. Brodal, G., Fagerberg, R., Jacob, R.: Cache oblivious search trees via binary trees of small height. In: SODA, pp. 39–48 (2002)
9. Bulánek, J., Koucký, M., Saks, M.: Tight lower bounds for online labeling problem. In: STOC, pp. 1185–1198 (2012)
10. Dietz, P., Seiferas, J., Zhang, J.: A tight lower bound for online monotonic list labeling. SIAM J. Discrete Math. 18(3), 626–637 (2004)
11. Dietz, P., Zhang, J.: Lower bounds for monotonic list labeling. In: Gilbert, J.R., Karlsson, R. (eds.) SWAT 1990. LNCS, vol. 447, pp. 173–180. Springer, Heidelberg (1990)
12. Emek, Y., Korman, A.: New bounds for the controller problem. Distributed Computing 24(3-4), 177–186 (2011)
13. Itai, A., Konheim, A., Rodeh, M.: A sparse table implementation of priority queues. In: Even, S., Kariv, O. (eds.) ICALP 1981. LNCS, vol. 115, pp. 417–431. Springer, Heidelberg (1981)
14. Korman, A., Kutten, S.: Controller and estimator for dynamic networks. In: PODC, pp. 175–184 (2007)
15. Willard, D.: A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. Inf. Comput. 97(2), 150–204 (1992)
16. Zhang, J.: Density Control and On-Line Labeling Problems. PhD thesis, University of Rochester (1993)

Dual Lower Bounds for Approximate Degree and Markov-Bernstein Inequalities*

Mark Bun** and Justin Thaler***

School of Engineering and Applied Sciences
Harvard University, Cambridge, MA
{mbun,jthaler}@seas.harvard.edu

Abstract. The ε-*approximate degree* of a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ is the minimum degree of a real polynomial that approximates f to within ε in the ℓ_∞ norm. We prove several lower bounds on this important complexity measure by explicitly constructing solutions to the dual of an appropriate linear program. Our first result resolves the ε-approximate degree of the two-level AND-OR tree for any constant $\varepsilon > 0$. We show that this quantity is $\Theta(\sqrt{n})$, closing a line of incrementally larger lower bounds [3, 11, 21, 30, 32]. The same lower bound was recently obtained independently by Sherstov using related techniques [25]. Our second result gives an explicit *dual polynomial* that witnesses a tight lower bound for the approximate degree of any symmetric Boolean function, addressing a question of Špalek [34]. Our final contribution is to reprove several Markov-type inequalities from approximation theory by constructing explicit dual solutions to natural linear programs. These inequalities underly the proofs of many of the best-known approximate degree lower bounds, and have important uses throughout theoretical computer science.

1 Introduction

Approximate degree is an important measure of the complexity of a Boolean function. It captures whether a function can be approximated by a low-degree polynomial with real coefficients in the ℓ_∞ norm, and it has many applications in theoretical computer science. The study of approximate degree has enabled progress in circuit complexity [7, 8, 19, 29], quantum computing (where it has been used to prove lower bounds on quantum query complexity, e.g. [2, 5, 14]), communication complexity [4, 10, 17, 27, 31, 33, 34], and computational learning theory (where approximate degree upper bounds underly the best known algorithms for PAC learning DNF formulas and agnostically learning disjunctions) [13, 15].

In this paper, we seek to advance our understanding of this fundamental complexity measure. We focus on proving approximate degree lower bounds

* The full version of this paper is available at http://arxiv.org/abs/1302.6191

** Supported in part by NSF grant CNS-1237235.

*** Supported by an NSF Graduate Research Fellowship and NSF grants CNS-1011840 and CCF-0915922.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 303–314, 2013.

by specifying explicit *dual polynomials*, which are dual solutions to a certain linear program capturing the approximate degree of any function. These polynomials act as certificates of the high approximate degree of a function. Their construction is of interest because these dual objects have been used recently to resolve several long-standing open problems in communication complexity (e.g. [4, 10, 17, 27, 33, 34]). See the survey of Sherstov [26] for an excellent overview of this body of literature.

Our Contributions. Our first result resolves the approximate degree of the function $f(x) = \wedge_{i=1}^{N} \vee_{j=1}^{N} x_{ij}$, showing this quantity is $\Theta(N)$. Known as the two-level AND-OR tree, f is the simplest function whose approximate degree was not previously characterized. A series of works spanning nearly two decades proved incrementally larger lower bounds on the approximate degree of this function, and this question was recently re-posed by Aaronson in a tutorial at FOCS 2008 [1]. Our proof not only yields a tight lower bound, but it specifies an explicit dual polynomial for the high approximate degree of f, answering a question of Špalek [34] in the affirmative.

Our second result gives an explicit dual polynomial witnessing the high approximate degree of any *symmetric* Boolean function, recovering a well-known result of Paturi [22]. Our solution builds on the work of Špalek [34], who gave an explicit dual polynomial for the OR function, and addresses an open question from that work.

Our final contribution is to reprove several classical Markov-type inequalities of approximation theory using simpler ideas from linear programming. These inequalities bound the derivative of a polynomial in terms of its degree. Combined with the well-known symmetrization technique (see e.g. [1, 19]), Markov-type inequalties have traditionally been the primary tool used to prove approximate degree lower bounds on Boolean functions (e.g. [2, 3, 21, 32]). Our proofs of these inequalities specify explicit dual solutions to a natural linear program (that differs from the one used to prove our first two results). While these inequalities have been known for over a century [9, 18], to the best of our knowledge our proof technique is novel, and we believe it sheds new light on these results.

2 Preliminaries

We work with Boolean functions $f : \{-1,1\}^n \to \{-1,1\}$ under the standard convention that 1 corresponds to logical false, and -1 corresponds to logical true. We let $\|f\|_\infty = \max_{x \in \{-1,1\}} |f(x)|$ denote the ℓ_∞ norm of f. The ε-approximate degree of a function $f : \{-1,1\}^n \to \{-1,1\}$, denoted $\deg_\varepsilon(f)$, is the minimum (total) degree of any real polynomial p such that $\|p - f\|_\infty \leq \varepsilon$, i.e. $|p(x) - f(x)| \leq \varepsilon$ for all $x \in \{-1,1\}^n$. We use $\widetilde{\deg}(f)$ to denote $\deg_{1/3}(f)$, and use this to refer to the *approximate degree* of a function without qualification. The choice of $1/3$ is arbitrary, as $\widetilde{\deg}(f)$ is related to $\deg_\varepsilon(f)$ by a constant factor for any constant $\varepsilon \in (0,1)$. We let OR_n and AND_n denote the OR function and AND function on n variables respectively. Define $\widetilde{\mathrm{sgn}}(x) = -1$ if $x < 0$ and 1 otherwise.

In addition to approximate degree, *block sensitivity* is also an important measure of the complexity of a Boolean function. We introduce this measure because functions with low block sensitivity are an "easy case" in the analysis of Theorem 2 below. The block sensitivity $\mathrm{bs}_x(f)$ of a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ at the point x is the maximum number of pairwise disjoint subsets $S_1, S_2, S_3, \cdots \subseteq \{1,2,\ldots,n\}$ such that $f(x) \neq f(x^{S_1}) = f(x^{S_2}) = f(x^{S_3}) = \ldots$ Here, x^S denotes the vector obtained from x by negating each entry whose index is in S. The block sensitivity $\mathrm{bs}(f)$ of f is the maximum of $\mathrm{bs}_x(f)$ over all $x \in \{-1,1\}^n$.

2.1 A Dual Characterization of Approximate Degree

For a subset $S \subset \{1,\ldots,n\}$ and $x \in \{-1,1\}^n$, let $\chi_S(x) = \prod_{i \in S} x_i$. Strong LP-duality yields the following well-known dual characterization of approximate degree (cf. [27]).

Theorem 1. *Let $f : \{-1,1\}^n \to \{-1,1\}$ be a Boolean function. Then $\deg_\varepsilon(f) > d$ if and only if there is a polynomial $\phi : \{-1,1\}^n \to \mathbb{R}$ such that*

$$\sum_{x \in \{-1,1\}^n} f(x)\phi(x) > \varepsilon, \tag{1}$$

$$\sum_{x \in \{-1,1\}^n} |\phi(x)| = 1, \tag{2}$$

and

$$\sum_{x \in \{-1,1\}^n} \phi(x)\chi_S(x) = 0 \text{ for each } |S| \leq d. \tag{3}$$

If ϕ satisfies Eq. (3), we say ϕ has *pure high degree d*. We refer to any feasible solution ϕ to the dual LP as a *dual polynomial* for f.

3 A Dual Polynomial for the AND-OR Tree

Define $\text{AND-OR}_N^M : \{-1,1\}^{MN} \to \{-1,1\}$ by $f(x) = \wedge_{i=1}^M \vee_{j=1}^N x_{ij}$. AND-OR_N^N is known as the two-level AND-OR tree, and its approximate degree has resisted characterization for close to two decades. Nisan and Szegedy proved an $\Omega(N^{1/2})$ lower bound on $\widetilde{\deg}(\text{AND-OR}_N^N)$ in [21]. This was subsequently improved to $\Omega(\sqrt{N \log N})$ by Shi [32], and improved further to $\Omega(N^{2/3})$ by Ambainis [3]. Most recently, Sherstov proved an $\Omega(N^{3/4})$ lower bound in [30], which was the best lower bound prior to our work. The best upper bound is $O(N)$ due to Høyer, Mosca, and de Wolf [11], which matches our new lower bound.

By refining Sherstov's analysis in [30], we will show that $\widetilde{\deg}(\text{AND-OR}_N^M) = \Omega(\sqrt{MN})$, which matches an upper bound implied by a result of Sherstov [28]. In particular, this implies that the approximate degree of the two-level AND-OR tree is $\Theta(N)$.

Theorem 2. $\widetilde{\deg}(\text{AND-OR}_N^M) = \Theta(\sqrt{MN})$.

Independent Work by Sherstov. Independently of our work, Sherstov [25] has discovered the same $\Omega(\sqrt{MN})$ lower bound on $\widetilde{\deg}(\text{AND-OR}_N^M)$. Both his proof and ours exploit the fact that the OR function has a dual polynomial with one-sided error. Our proof proceeds by constructing an explicit dual polynomial for AND-OR_N^M, by combining a dual polynomial for OR_N with a dual polynomial for AND_M. In contrast, Sherstov mixes the primal and dual views: his proof combines a dual polynomial for OR_N with an approximating polynomial p for AND-OR_N^M to construct an approximating polynomial q for AND_M. The proof in [25] shows that q has much lower degree than p, so the desired lower bound on the degree of p follows from known lower bounds on the degree of q.

The proof of [25] is shorter, while our proof has the benefit of yielding an explicit dual polynomial witnessing the lower bound.

3.1 Proof Outline

Our proof is a refinement of a result of Sherstov [30], which roughly showed that approximate degree increases multiplicatively under function composition. Specifically, Sherstov showed the following.

Proposition 1 ([30, Theorem 3.3]). *Let $F : \{-1,1\}^M \to \{-1,1\}$ and $f : \{-1,1\}^N \to \{-1,1\}$ be given functions. Then for all $\varepsilon, \delta > 0$,*

$$\deg_{\varepsilon - 4\delta \,\mathrm{bs}(F)}(F(f,\ldots,f)) \geq \deg_\varepsilon(F) \deg_{1-\delta}(f).$$

Sherstov's proof of Proposition 1 proceeds by taking a dual witness Ψ to the high ε-approximate degree of F, and combining it with a dual witness ψ to the high $(1-\delta)$-approximate degree of f to obtain a dual witness ζ for the high $(\varepsilon - 4\delta \mathrm{bs}(F))$-approximate degree of $F(f,\ldots,f)$. His proof proceeds in two steps: he first shows that ζ has pure high degree at least $\deg_\varepsilon(F)\deg_{1-\delta}(f)$, and then he lower bounds the correlation of ζ with $F(f,\ldots,f)$. The latter step of this analysis yields a lower bound on the correlation of ζ with $F(f,\ldots,f)$ that deteriorates rapidly as the block sensitivity $\mathrm{bs}(F)$ grows.

Proposition 1 itself does not yield a tight lower bound for $\widetilde{\deg}(\text{AND-OR}_N^M)$, because the function AND_M has maximum block sensitivity $\mathrm{bs}(\text{AND}_M) = M$. We address this by refining the second step of Sherstov's analysis in the case where $F = \text{AND}_M$ and $f = \text{OR}_N$. We leverage two facts. First, although the block sensitivity of AND_M is high, it is only high at one input, namely the all-true input. At all other inputs, AND_M has low block sensitivity and the analysis of Proposition 1 is tight. Second, we use the fact that any dual witness to the high approximate degree of OR_N has one-sided error. Namely, if $\psi(x) < 0$ for such a dual witness ψ, then we know that $\psi(x)$ agrees in sign with $\text{OR}_N(x)$. This property allows us to handle the all-true input to AND_M separately: we use it to show that despite the high block-sensitivity of AND_M at the all-true input y, this input nonetheless contributes positively to the correlation between ζ and $F(f,\ldots,f)$.

3.2 Proof of Thm. 2

As in Sherstov's proof of Proposition 1, we define $\zeta : \left(\{-1,1\}^N\right)^M \to \mathbb{R}$ by

$$\zeta(x_1,\ldots,x_M) := 2^M \Psi(\ldots,\widetilde{\operatorname{sgn}}(\psi(x_i)),\ldots)\prod_{i=1}^{M}|\psi(x_i)|, \tag{4}$$

where Ψ and ψ are dual witnesses to the high ε-approximate degree of AND_M and $(1-\delta)$-approximate degree of OR_N, respectively, for suitable ε and δ, and $x_i = (x_{i,1},\ldots,x_{i,N})$. To show that ζ is a dual witness for the fact that the $(1/3)$-approximate degree of AND-OR_N^M is $\Omega(\sqrt{MN})$, it suffices to check that ζ satisfies the conditions of Thm. 1. The only place where our analysis differs from that of Sherstov's is in verifying Expression (1), i.e. that

$$\sum_{(x_1,\ldots,x_M)\in(\{-1,1\}^N)^M} \zeta(x_1,\ldots,x_M)\text{AND-OR}_N^M(x_1,\ldots,x_M) > 1/3. \tag{5}$$

Let $A_1 = \{x \in \{-1,1\}^N : \psi(x) \geq 0, \mathrm{OR}_N(x) = -1\}$ and $A_{-1} = \{x \in \{-1,1\}^N : \psi(x) < 0, \mathrm{OR}_N(x) = 1\}$, so $A_1 \cup A_{-1}$ is the set of all inputs x where the sign of $\psi(x)$ disagrees with $\mathrm{OR}_N(x)$. Notice that $\sum_{x\in A_1\cup A_{-1}}|\psi(x)| < \delta/2$ because ψ has correlation $1-\delta$ with f. A sequence of manipulations found in the full version of this paper shows that the left-hand side of (5) equals

$$\sum_{z\in\{-1,1\}^M} \Psi(z)\cdot\mathbf{E}[\mathrm{AND}_M(\ldots,y_iz_i,\ldots)], \tag{6}$$

where $y \in \{-1,1\}^M$ is a random string whose ith bit independently takes on value -1 with probability $2\sum_{x\in A_{z_i}}|\psi(x)| < \delta$.

All $z \neq -\mathbf{1}_M$ can be handled as in Sherstov's proof of Proposition 1, because AND_M has low block sensitivity at these inputs. These inputs contribute a total of at least $\varepsilon - 4\delta - |\Psi(-\mathbf{1}_M)|$ to Expression (6). We only need to argue that the term corresponding to $z = -\mathbf{1}_M$ contributes $|\Psi(-\mathbf{1}_M)|$ to the correlation. In the full version, we argue that any dual witness for the OR_N function has one-sided error [12]. That is, if $\mathrm{OR}(x) = 1$ (i.e. if $x = \mathbf{1}_N$), then $\widetilde{\operatorname{sgn}}(\psi(x)) = 1$. This implies that A_{-1} is empty; that is, if $\widetilde{\operatorname{sgn}}(\psi(x)) = -1$, then it must be the case that $\mathrm{OR}_N(x) = -1$. Therefore, for $z = -\mathbf{1}_M$, the y_i's are all -1 with probability 1, and hence $\mathbf{E}_y[\mathrm{AND}_M(\ldots,y_iz_i,\ldots)] = \mathrm{AND}_M(-\mathbf{1}_M) = -1$. By the one-sided error of any dual witness for AND_M, $\widetilde{\operatorname{sgn}}(\Psi(-\mathbf{1}_M)) = -1$, and thus the term corresponding to $z = -\mathbf{1}_M$ contributes $-\Psi(z) = |\Psi(z)|$ to Expression (6) as claimed. □

Remark 1. Špalek [34] has exhibited an explicit dual witness showing that the ε-approximate degree of both the AND function and the OR function is $\Omega(\sqrt{n})$, for $\varepsilon = 1/14$ (in fact, we generalize Špalek's construction in the next section to any symmetric function). It is relatively straightforward to modify his construction to handle any constant $\varepsilon \in (0,1)$. With these dual polynomials in hand, the dual solution ζ we construct in our proof is completely explicit. This answers a question of Špalek [34, Section 4] in the affirmative.

4 Dual Polynomials for Symmetric Boolean Functions

In this section, we construct a dual polynomial witnessing a tight lower bound on the approximate degree of any symmetric function. The lower bound we recover was first proved by Paturi [22] via a symmetrization argument combined with the classical Markov-Bernstein inequality from approximation theory (see Section 5). Paturi also provided a matching upper bound. Špalek [34], building on work of Szegedy, presented a dual witness to the $\Omega(\sqrt{n})$-approximate degree of the OR function and asked whether one could construct an analogous dual polynomial for the symmetric t-threshold function [34, Section 4]. We accomplish this in the more general case of arbitrary symmetric functions by extending the ideas underlying Špalek's dual polynomial for OR.

4.1 Symmetric Functions

For a vector $x \in \{-1,1\}^n$, let $|x| = \frac{1}{2}(n - (x_1 + \cdots + x_n))$ denote the number of -1's in x. A Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ is symmetric if $f(x) = f(y)$ whenever $|x| = |y|$. That is, the value of f depends only on the number of inputs that are set to -1.

Let $[n] = \{0, 1, \ldots, n\}$. To each symmetric function f, we can associate a unique univariate function $F : [n] \to \{-1,1\}$ by taking $F(|x|) = f(x)$. Throughout this section, we follow the convention that lower case letters refer to multivariate functions, while upper case letters refer to their univariate counterparts.

We now discuss the dual characterization of approximate degree established in Thm. 1, as it applies to symmetric functions. Following the notation in [34], the standard inner product $p \cdot q = \sum_{x \in \{-1,1\}^n} p(x)q(x)$ on symmetric functions p, q induces an inner product on the associated univariate functions:

$$P \cdot Q := \sum_{i=0}^{n} \binom{n}{i} P(i)Q(i).$$

We refer to this as the *correlation* between P and Q. Similarly, the ℓ_1-norm $\|p\|_1 = \sum_{x \in \{-1,1\}^n} |p(x)|$ induces a norm $\|P\|_1 = \sum_{i=0}^{n} \binom{n}{i} P(i)$. These definitions carry over verbatim when f is real-valued instead of Boolean-valued.

If f is symmetric, we can restrict our attention to symmetric ϕ in the statement of Thm. 1, and it becomes convenient to work with the following reformulation of Thm. 1.

Corollary 1. *A symmetric function $f : \{-1,1\}^n \to \{-1,1\}$ has ε-approximate degree greater than d iff there exists a symmetric function $\phi : \{-1,1\}^n \to \mathbb{R}$ with pure high degree d such that*

$$\frac{\Phi \cdot F}{\|\Phi\|_1} = \frac{\phi \cdot f}{\|\phi\|_1} > \varepsilon.$$

(Here, F and Φ are the univariate function associated to f and ϕ, respectively).

We clarify that the pure high degree of a multivariate polynomial ϕ does not correspond to the smallest degree of a monomial in the associated univariate function Φ. When we talk about the pure high degree of a univariate polynomial Φ, we mean the pure high degree of its corresponding multilinear polynomial ϕ. It is straightforward to check that if ψ is a multivariate polynomial of degree $n-d$, then multiplying ψ by the parity function yields a univariate function $\Phi(k) := \Psi(k) \cdot (-1)^k$ with pure high degree d.

We are now in a position to state the lower bound that we will prove in this section. Paturi [22] completely characterized the approximate degree of a symmetric Boolean function by the location of the layer t closest to the center of the Boolean hypercube such that $F(t-1) \neq F(t)$.

Theorem 3 ([22], Theorem 4). *Given a nonconstant symmetric Boolean function f with associated univariate function F, let $\Gamma(f) = \min\{|2t-n-1| : F(t-1) \neq F(t), 1 \leq k \leq n\}$. Then $\widetilde{\deg}(f) = \Theta(\sqrt{n(n-\Gamma(f))})$.*

Paturi proved the upper bound non-explicitly by appealing to Jackson theorems from approximation theory. He proved the lower bound by combining symmetrization with an appeal to the Markov-Bernstein inequality (see Section 5) – his proof does not yield an explicit dual polynomial. We construct an explicit dual polynomial to prove the following proposition, which is easily seen to imply Paturi's lower bound.

Proposition 2. *Given f and F as above, let $1 \leq t \leq n$ be an integer with $F(t-1) \neq F(t)$. Then $\widetilde{\deg}(f) = \Omega(\sqrt{t(n-t+1)})$.*

Proof Outline. We start with an intuitive discussion of Špalek's construction of a dual polynomial for OR, with the goal of elucidating how we extend the construction to arbitrary symmetric functions. Consider the perfect squares $S = \{k^2 : 0 \leq k^2 \leq n\}$ and the univariate polynomial

$$R(x) = \frac{1}{n!} \prod_{i \in [n] \setminus S} (x-i).$$

This polynomial is supported on S, and for all $k \in S$,

$$\binom{n}{k^2} |R(k^2)| = \binom{n}{k^2} \cdot \frac{1}{n!} \cdot \frac{\prod_{\substack{i \in [n] \\ i \neq k^2}} |k^2 - i|}{\prod_{\substack{i \in S \\ i \neq k^2}} |k^2 - i|} = \frac{1}{\prod_{\substack{i \in S \\ i \neq k^2}} |k^2 - i|}.$$

Note the remarkable cancellation in the final equality. This quotient is maximized at $k = 1$. In other words, the threshold point $t = 1$ makes the largest contribution to the ℓ_1 mass of R. Moreover, one can check that $R(0)$ is only a constant factor smaller than $R(1)$.

Špalek exploits this distribution of the ℓ_1 mass by considering the polynomial $P(x) = R(x)/(x-2)$. The values of $P(x)$ are related to $R(x)$ by a constant multiple for $x = 0, 1$, but $P(k)$ decays as $|P(k^2)| \approx |R(k^2)|/k^2$ for larger values.

This decay is fast enough that a *constant fraction* of the ℓ_1 mass of P comes from the point $P(0)$.[1] Now P is an $(n - \Omega(\sqrt{n}))$-degree univariate polynomial, so we just need to show that $Q(i) = (-1)^i P(i)$ has high correlation with OR. We can write

$$Q \cdot \mathrm{OR} = 2Q(0) - Q \cdot \mathbf{1} = 2Q(0),$$

since the multilinear polynomial associated to Q has pure high degree $\Omega(\sqrt{n})$, and therefore has zero correlation with constant functions. Because a constant fraction of the ℓ_1 mass of Q comes from $Q(0)$, it follows that $|Q \cdot \mathrm{OR}|/\|Q\|_1$ is bounded below by a constant. By perhaps changing the sign of Q, we get a good dual polynomial for OR.

A natural approach to extend Špalek's argument to symmetric functions with a "jump" at t is the following:

1) Find a set S with $|S| = \Omega(\sqrt{t(n-t+1)})$ such that the maximum contribution to the ℓ_1 norm of $R(x) = \frac{1}{n!}\prod_{i \in [n] \setminus S}(x - i)$ comes from the point $x = t$. Equivalently,
$$\binom{n}{j}|R(j)| = \frac{1}{\prod_{\substack{i \in S \\ i \neq j}} |j - i|}$$
is maximized at $j = t$.
2) Define a polynomial $P(x) = R(x)/(x - (t-1))(x - (t+1))$. Dividing $R(x)$ by the factor $(x - t - 1)$ is analogous to Špalek's division of $R(x)$ by $(x - 2)$. We also divide by $(x - t + 1)$ because we will ultimately need our polynomial $P(x)$ to decay faster than Špalek's by a factor of $|x - t|$ as x moves away from the threshold. By dividing by both $(x - t - 1)$ and $(x - t + 1)$, we ensure that most of the ℓ_1 mass of P is concentrated at the points $t - 1, t, t + 1$.
3) Obtain Q by multiplying P by parity, and observe that $Q(t-1)$ and $Q(t)$ have opposite signs. Since $F(t-1)$ and $F(t)$ also have opposite signs, we can ensure that both $t - 1$ and t contribute positive correlation. Suppose these two points contribute a $1/2 + \varepsilon$ constant fraction of the ℓ_1-norm of Q. Then even in the worst case where the remaining points all contribute negative correlation, $Q \cdot F$ is still at least a 2ε fraction of $\|Q\|_1$ and we have a good dual polynomial. Notice that the pure high degree of Q is $|S| + 2$, yielding the desired lower bound.

In the case where $t = \Omega(n)$, we can use the set

$$S = \{t \pm 4\ell : 0 \leq \ell \leq t/4\},$$

yielding a remarkably clean dual polynomial for the majority function. This partial result also gives the right intuition for general t, although the details are somewhat more complicated and spelled out in the full version of this paper. In general, the set S interpolates between the set for OR used by Špalek, and the set described above for linear t. In particular, S contains all points of the form $t \pm 4\ell$, plus additional points corresponding to perfect squares when $t = o(n)$.

[1] It is also necessary to check that $P(2)$ is only a constant factor larger than $P(0)$.

5 A Constructive Proof of Markov-Bernstein Inequalities

The Markov-Bernstein inequality for polynomials with real coefficients asserts that

$$p'(x) \leq \min\left\{\frac{n}{\sqrt{1-x^2}}, n^2\right\} \|p\|_{[-1,1]}, x \in (-1,1)$$

for every real polynomial of degree at most n. Here, and in what follows,

$$\|p\|_{[-1,1]} := \sup_{y \in [-1,1]} |p(y)|.$$

This inequality has found numerous uses in theoretical computer science, especially in conjunction with symmetrization as a method for bounding the ε-approximate degree of various functions (e.g. [2, 8, 13, 16, 21, 22, 27]).

We prove a number of important special cases of this inequality based on linear programming duality. Our proofs are constructive in that we exhibit explicit dual solutions to a linear program bounding the derivative of a constrained polynomial.

The special cases of the Markov-Bernstein inequality that we prove are sufficient for many applications in theoretical computer science. The dual solutions we exhibit are remarkably clean, and we believe that they shed new light on these classical inequalities.

5.1 Proving the Markov-Bernstein Inequality at $x = 0$

The following linear program with uncountably many constraints captures the problem of finding a polynomial $p(x) = c_n x^n + c_{n-1}x^{n-1} + \cdots + c_1 x + c_0$ with real-valued coefficients that maximizes $|p'(0)|$ subject to the constraint that $\|p\|_{[-1,1]} \leq 1$. Below the variables are $c_0, \ldots c_n$, and there is a constraint for every $x \in [-1, 1]$.

$$\begin{array}{rl} & \max c_1 \\ \text{such that} & \sum_{i=0}^{n} c_i x^i \leq 1, \ \forall x \in [-1,1] \\ & -\sum_{i=0}^{n} c_i x^i \leq 1, \ \forall x \in [-1,1] \end{array}$$

We will actually upper bound the value of the following LP, which is obtained from the above by throwing away all but finitely many constraints. Not coincidentally, the constraints that we keep are those that are tight for the primal solution corresponding to the Chebyshev polynomials of the first kind. Throughout this section, we refer to this LP as Primal.

$$\begin{array}{rl} & \max c_1 \\ \text{such that} & \sum_{i=0}^{n} c_i x^i \leq 1, \ \forall x = \cos(k\pi/n), \ k \in \{0, 2, \ldots, n-1\} \\ & -\sum_{i=0}^{n} c_i x^i \leq 1, \ \forall x = \cos(k\pi/n), \ k \in \{1, 3, \ldots, n\} \end{array}$$

The dual to Primal can be written as

$$\begin{aligned} &\min \textstyle\sum_{i=0}^{n} y_i \\ &\text{such that } Ay = e \\ &\qquad y_j \geq 0 \ \forall j \in \{0, \ldots, n\} \end{aligned}$$

where $A_{ij} = (-1)^j \cos^i(j\pi/n)$ and $e = (0, 1, 0, 0, 0, \ldots, 0)^T$. We refer to this linear program as DUAL.

Our goal is to prove that PRIMAL has value at most n. For odd n, it is well-known that this value is achieved by the coefficients of $(-1)^{(n-1)/2} T_n(x)$, the degree n Chebyshev polynomial of the first kind. Our knowledge of this primal-optimal solution informed our search for a dual-optimal solution, but our proof makes no explicit reference to the Chebyshev polynomials, and we do not need to invoke strong LP duality; weak duality suffices. Our arguments rely on a number of trigonometric identities that can all be established by elementary methods.

Proposition 3. *Let $n = 2m + 1$ be odd. Define the $(n + 1) \times (n + 1)$ matrix A by $A_{ij} = (-1)^{j+m} \cos^i(j\pi/n)$ for $0 \leq i, j \leq n$. Then*

$$y = \frac{1}{n}(1/2, \sec^2(\pi/n), \sec^2(2\pi/n), \ldots, \sec^2((n-1)\pi/n), 1/2)^T$$

is the unique solution to $Ay = e_1$, where $e_1 = (0, 1, 0, 0, \ldots, 0)^T$.

Note that y is clearly nonnegative, and thus is the unique feasible solution for DUAL. Therefore it is the dual-optimal solution, and as the entries of y sum to n, it exactly recovers the Markov-Bernstein inequality at $x = 0$:

Corollary 2. *Let p be a polynomial of degree $n = 2m + 1$ with $\|p\|_{[-1,1]} \leq 1$. Then $p'(0) \leq n$.*

While we have recovered the Markov-Bernstein inequality only for odd-degree polynomials at zero, a simple "shift-and-scale" argument recovers the asymptotic bound for any x bounded away from the endpoints $\{-1, 1\}$.

Corollary 3. *Let p be a polynomial of degree n with $\|p\|_{[-1,1]} \leq 1$. Then for any $x_0 \in (-1, 1)$, $|p'(x_0)| \leq \frac{n+1}{1-|x_0|}\|p\|_{[-1,1]}$. In particular, for any constant $\varepsilon \in (0, 1)$, $\|p'\|_{[-1+\varepsilon, 1-\varepsilon]} = O(n)\|p\|_{[-1,1]}$.*

We remark that the full Markov-Bernstein inequality guarantees that $|p'(x)| \leq \frac{n}{\sqrt{1-x^2}}\|p\|_{[-1,1]}$, which has quadratically better dependence on the distance from x to ± 1. However, for x bounded away from ± 1 our bound is asymptotically tight and sufficient for many applications in theoretical computer science, such as proving that the approximate degree of the Majority function on n variables is $\Omega(n)$. Moreover, we can recover the Markov-Bernstein inequality near ± 1 by considering a different linear program. We omit the details from this extended abstract for brevity.

6 Conclusion

The approximate degree is a fundamental measure of the complexity of a Boolean function, with pervasive applications throughout theoretical computer science. We have sought to advance our understanding of this complexity measure by resolving the approximate degree of the AND-OR tree, and reproving old lower bounds through the construction of explicit dual witnesses. Nonetheless, few general results on approximate degree are known, and our understanding of the approximate degree of fundamental classes of functions remains incomplete. For example, the approximate degree of AC^0 remains open [2, 6], as does the approximate degree of approximate majority (see [20, Page 11]).[2]

Resolving these open questions may require moving beyond traditional symmetrization-based arguments, which transform a polynomial p on n variables into a polynomial q on $m < n$ variables in such a way that $\widetilde{\deg}(q) \leq \widetilde{\deg}(p)$, before obtaining a lower bound on $\widetilde{\deg}(q)$. Symmetrization necessarily "throws away" information about p; in contrast, the method of constructing dual polynomials appears to be a very powerful and complete way of reasoning about approximate degree. Can progress be made on these open problems by directly constructing good dual polynomials?

References

1. Aaronson, S.: The polynomial method in quantum and classical computing. In: Proc. of Foundations of Computer Science (FOCS), p. 3 (2008), Slides available at `http://www.scottaaronson.com/talks/polymeth.ppt`
2. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. Journal of the ACM 51(4), 595–605 (2004)
3. Ambainis, A.: Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. Theory of Computing 1(1), 37–46 (2005)
4. Chattopadhyay, A., Ada, A.: Multiparty communication complexity of disjointness. Electronic Colloquium on Computational Complexity (ECCC) 15(002) (2008)
5. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bound by polynomials. Journal of the ACM 48(4), 778–797 (2001)
6. Beame, P., Machmouchi, W.: The quantum query complexity of AC^0. Quantum Information & Computation 12(7-8), 670–676 (2012)
7. Beigel, R.: The polynomial method in circuit complexity. In: Proc. of the Conference on Structure in Complexity Theory, pp. 82–95 (1993)
8. Beigel, R.: Perceptrons, PP, and the polynomial hierarchy. In: Computational Complexity, vol. 4, pp. 339–349 (1994)
9. Bernstein, S.N.: On the V. A. Markov theorem. Trudy Leningr. Industr. In-ta, no 5, razdel fiz-matem nauk, 1 (1938)
10. Buhrman, H., Vereshchagin, N.K., de Wolf, R.: On computation and communication with small bias. In: Proc. of the Conference on Computational Complexity, pp. 24–32 (2007)
11. Høyer, P., Mosca, M., de Wolf, R.: Quantum search on bounded-error inputs. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 291–299. Springer, Heidelberg (2003)

[2] This open problem is due to Srikanth Srinivasan.

12. Gavinsky, D., Sherstov, A.A.: A separation of NP and coNP in multiparty communication complexity. Theory of Computing 6(1), 227–245 (2010)
13. Kalai, A., Klivans, A.R., Mansour, Y., Servedio, R.A.: Agnostically learning halfspaces. SIAM Journal on Computing 37(6), 1777–1805 (2008)
14. Klauck, H., Špalek, R., de Wolf, R.: Quantum and classical strong direct product theorems and optimal time-space tradeoffs. SIAM Journal on Computing 36(5), 1472–1493 (2007)
15. Klivans, A.R., Servedio, R.A.: Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. J. of Comput. and System Sci. 68(2), 303–318 (2004)
16. Klivans, A.R., Sherstov, A.A.: Lower bounds for agnostic learning via approximate rank. Computational Complexity 19(4), 581–604 (2010)
17. Lee, T., Shraibman, A.: Disjointness is hard in the multi-party number-on-the-forehead model. In: Proc. of the Conference on Computational Complexity, pp. 81–91 (2008)
18. Markov, V.: On functions which deviate least from zero in a given interval, St. Petersburg (1892) (Russian)
19. Minsky, M.L., Papert, S.A.: Perceptions: An Introduction to Computational Geometry. MIT Press, Cambridge (1969)
20. Open problems in analysis of Boolean functions. Compiled for the Simons Symposium. CoRR, abs/1204.6447, February 5-11 (2012)
21. Nisan, N., Szegedy, M.: On the degree of boolean functions as real polynomials. Computational Complexity 4, 301–313 (1994)
22. Paturi, R.: On the degree of polynomials that approximate symmetric Boolean functions (Preliminary Version). In: Proc. of the Symp. on Theory of Computing (STOC), pp. 468–474 (1992)
23. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, New York (1986)
24. Sherstov, A.A.: Approximate inclusion-exclusion for arbitrary symmetric functions. Computational Complexity 18(2), 219–247 (2009)
25. Sherstov, A.A.: Approximating the AND-OR tree. Electronic Colloquium on Computational Complexity (ECCC) 20(023) (2013)
26. Sherstov, A.A.: Communication lower bounds using dual polynomials. Bulletin of the EATCS 95, 59–93 (2008)
27. Sherstov, A.A.: The pattern matrix method. SIAM J. Comput. 40(6), 1969–2000 (2011)
28. Sherstov, A.A.: Making polynomials robust to noise. In: Proceedings of Symp. Theory of Computing, pp. 747–758 (2012)
29. Sherstov, A.A.: Separating AC^0 from depth-2 majority circuits. SIAM Journal on Computing 28(6), 2113–2129 (2009)
30. Sherstov, A.A.: The intersection of two halfspaces has high threshold degree. In: Proc. of Foundations of Computer Science (FOCS), pp. 343–362 (2009); To appear in SIAM J. Comput. (special issue for FOCS 2009)
31. Sherstov, A.A.: The multiparty communication complexity of set disjointness. In: Proceedings of Symp. Theory of Computing, pp. 525–548 (2012)
32. Shi, Y.: Approximating linear restrictions of Boolean functions. Manuscript (2002), `http://web.eecs.umich.edu/~shiyy/mypapers/linear02-j.ps`
33. Shi, Y., Zhu, Y.: Quantum communication complexity of block-composed functions. Quantum Information & Computation 9(5), 444–460 (2009)
34. Špalek, R.: A dual polynomial for OR. Manuscript (2008), `http://arxiv.org/abs/0803.4516`

New Doubling Spanners: Better and Simpler*

T.-H. Hubert Chan[1], Mingfei Li[1], Li Ning[1], and Shay Solomon[2]

[1] Department of Computer Science, The University of Hong Kong
{hubert,mfli,lning}@cs.hku.hk
[2] Department of Computer Science and Applied Mathematics,
The Weizmann Institute of Science, Rehovot 76100, Israel
shay.solomon@weizmann.ac.il

Abstract. In a seminal STOC'95 paper, Arya et al. conjectured that spanners for low-dimensional Euclidean spaces with constant maximum degree, hop-diameter $O(\log n)$ and lightness $O(\log n)$ (i.e., weight $O(\log n) \cdot w(\mathsf{MST})$) can be constructed in $O(n \log n)$ time. This conjecture, which became a central open question in this area, was resolved in the affirmative by Elkin and Solomon in STOC'13 (even for doubling metrics).

In this work we present a simpler construction of spanners for doubling metrics with the above guarantees. Moreover, our construction extends in a simple and natural way to provide k-fault tolerant spanners with maximum degree $O(k^2)$, hop-diameter $O(\log n)$ and lightness $O(k^2 \log n)$.

1 Introduction

An n-point metric space (X, d) can be represented by a complete weighted graph $G = (X, E)$, where the weight $w(e)$ of an edge $e = (u, v)$ is given by $d(u, v)$. A *t-spanner* of X is a weighted subgraph $H = (X, E')$ of G (where $E' \subseteq E$ has the same weights) that preserves all pairwise distances to within a factor of t, i.e., $d_H(u, v) \leq t \cdot d(u, v)$ for all $u, v \in X$, where $d_H(u, v)$ is the distance between u and v in H. The parameter t is called the *stretch* of the spanner H. A path between u and v in H with weight at most $t \cdot d(u, v)$ is called a *t-spanner path*.

In this paper we focus on the regime of stretch $t = 1+\epsilon$, for an arbitrarily small $0 < \epsilon < \frac{1}{2}$. In general, there are metric spaces (such as the one corresponding to uniformly weighted complete graph), where the only possible $(1 + \epsilon)$-spanner is the complete graph. A special class of metric spaces, which has been subject to intensive research in the last decade, is the class of *doubling metrics*. The *doubling dimension* of a metric space (X, d), denoted by $\dim(X)$ (or dim when the context is clear), is the smallest value ρ such that every ball in X can be covered by 2^ρ balls of half the radius [12]. A metric space is called *doubling* if its doubling dimension is bounded by some constant. (We will sometimes disregard dependencies on ϵ and dim to avoid cluttered expressions in the text, but we provide these dependencies in all formal statements.) The doubling dimension is a generalization of the Euclidean dimension for arbitrary metric spaces, as the

* This work is supported by the Koshland Center for basic Research.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 315–327, 2013.

Euclidean space $\mathbb{R}^D$ equipped with any of the ℓ_p-norms has doubling dimension $\Theta(D)$ [12]. Spanners for doubling metrics (hereafter, *doubling spanners*), and in particular for low-dimensional Euclidean spaces, have been studied extensively since the mid-eighties; see [6,1,9,2,13,10,3,11,15] and the references therein.

In addition to small stretch and small number of edges, it is often desirable to optimize other parameters depending on the application. First, it is often important for the spanner to achieve a small maximum degree (or shortly, degree), hence having a small number of edges. Second, it is sometimes required that the *hop-diameter* would be small, i.e., every pair of points should be connected by a t-spanner path with a small number of edges (or *hops*). Third, it is desirable that the weight of the spanner would be at most some small factor (called *lightness*) times the weight of a minimum spanning tree (MST) of the metric space.

A natural requirement for a spanner is to be robust against node failures, meaning that even when some of the nodes in the spanner fail, the remaining part still provides a t-spanner. Formally, given a parameter $1 \leq k \leq n-2$, a spanner H of X is called a k-vertex-fault-tolerant t-spanner ((k,t)-VFTS), if for any subset $F \subseteq X$ with $|F| \leq k$, $H \setminus F$ is a t-spanner for $X \setminus F$.

1.1 Our Contribution

The following is the main result of this paper.

Theorem 1 ($(k, 1+\epsilon)$-VFTS with Degree $O(k^2)$, Hop-Diameter $O(\log n)$ and Lightness $O(k^2 \log n)$). *Let (X, d) be an n-point metric space, and let $0 < \epsilon < 1$. Given any parameter $1 \leq k \leq n-2$, there exists a $(k, 1+\epsilon)$-VFTS with degree $\epsilon^{-O(\dim)} \cdot k^2$, hop-diameter $O(\log n)$, and lightness $\epsilon^{-O(\dim)} \cdot k^2 \cdot \log n$. Such a spanner can be constructed in $\epsilon^{-O(\dim)} \cdot n \log n + \epsilon^{-O(\dim)} \cdot k^2 n$ time.*

Research Background. We review the most relevant related work; the readers can refer to [5,8] for a more detailed survey. In a seminal STOC'95 paper, Arya et al. [1] gave several constructions of *Euclidean spanners* that trade between degree, hop-diameter and lightness. In particular, they showed that for any n-point low-dimensional Euclidean space, a $(1+\epsilon)$-spanner with constant degree, hop-diameter $O(\log n)$ and lightness $O(\log^2 n)$ can be built in $O(n \log n)$ time.

Arya et al. [1] conjectured that the lightness bound can be improved to $O(\log n)$, without increasing the stretch, the degree and the hop-diameter of the spanner, and within the same running time $O(n \log n)$. The bound $O(\log n)$ on the lightness is optimal due to a lower bound result by Dinitz et al. [7].

This conjecture was resolved in the affirmative by Elkin and Solomon [8] only recently. In fact, Elkin and Solomon showed a stronger result: their construction works for doubling metrics, and moreover, it provides a general tradeoff between the involved parameters that is tight up to constants in the entire range.

Chan et al. [4] showed that the standard net-tree with cross edge framework (used in [9,2]) can be modified to give a simpler spanner construction with all the desired properties, except for running time $O(n \log n)$. Combining the techniques from a previous work on k-fault tolerant spanners [5], a precursor of this paper's

result was achieved, but with a worse lightness of $O(k^3 \log n)$; moreover, the running time was not analyzed in [4]. Solomon [14] made further improvements to the construction in [4], and achieved improved lightness $O(k^2 \log n)$ and running time $O(n \log n)$. This paper is the result of a collaboration between the authors of the (unpublished) manuscripts [4,14].

1.2 Our Techniques

We first give the main ideas for constructing a spanner when all nodes are functioning. The treatment for fault-tolerance is given in Section 5. We use the *standard net-tree with cross edge framework* from [9,2]: given a metric space (X, d), construct a hierarchical sequence $\{N_i\}$ of nets with geometrically increasing distance scales. For each $x \in N_i \subseteq X$ in level i (hereafter, *level-i net-point*), we have a node (x, i) called *incubator*. The hierarchical net structure induces an *incubator tree* (IncTree) with the incubators as nodes. At each level, if two net-points are close together with respect to the distance scale at that level, we add a *cross edge* between their corresponding incubators.

A basic spanner [9,2] consisting of the tree edges and the cross edges can be shown to have a low stretch. The basic idea is that for any two points u and v, we can start at the corresponding leaf nodes and climb to an appropriate level (depending on $d(u, v)$) to reach net-points u' and v' that are close to u and v, respectively, such that the cross edge $\{u', v'\}$ is guaranteed to exist. Observe that there is a 1-1 correspondence between the leaf nodes and the original points of X (i.e., each leaf corresponds to a unique point), and we later show how to label each internal node with a unique point in X using the incubator-zombie terminology.[1] Next, we analyze each of the involved parameters (degree, hop-diameter, and lightness), and explain how issues that arise can be resolved.

Degree. Since the doubling dimension is constant, each node in the net-tree has a constant number of children and a constant number of incident cross edges. However, in many net-based spanner constructions, each chain of *lonely* nodes (i.e., a chain of nodes each of which has only one child) will be *contracted*; this may increase the degree of the contracted nodes due to cross edges. The idea of constant degree single-sink spanners (used in [2,5]) can be applied to resolve this issue. However, a simpler method is *parent replacement*, used by Gottlieb and Roditty [11] to build a *routing tree* (RouTree) and reroute spanner paths, thus pruning unnecessary cross edges. In Section 3 we use Gottlieb and Roditty's construction to bound the degree of our spanner construction.

Hop-Diameter. Observe that there may be many levels in the net-tree. In this case, in the aforementioned spanner path between u and v, it will take many hops to go from u (respectively, v) to an appropriate ancestor u' (resp., v'). However, this can be easily fixed by adding shortcut edges to subtrees of the contracted routing tree (ConRouTree) at "small" distance scales via the 1-spanner construction with hop-diameter $O(\log n)$ for tree metrics by Solomon and Elkin [15]; this "shortcut spanner" increases both the degree and the lightness by a constant.

[1] The terminology is borrowed from [8], but these terms have different meaning there.

We shall see in Section 4 that there are only $O(\log n)$ levels with "large" distance scales, and hence hop-diameter $O(\log n)$ can achieved.

Lightness. For doubling metrics, lightness comes almost for free. We will show that the total weight of small-scale edges is $O(w(\mathsf{MST}))$. For each of the $O(\log n)$ large-scale levels, the standard analysis in [9,2] uses the fact that for doubling metrics each net-point has a constant number of neighbors at that level. Consequently, the weight of edges from each large-scale level is only a constant times that of an MST, thereby giving lightness $O(\log n)$. See Section 4 for the details.

To summarize, we obtained a spanner on the incubators that consists of (1) edges in ConRouTree, where each chain of lonely nodes is contracted into a single *super incubator*, (2) useful cross edges (which are not pruned after rerouting), and (3) edges in the shortcut spanner for ConRouTree. The resulting *incubator graph* $\mathcal{H}$ has constant degree, hop-diameter $O(\log n)$, lightness $O(\log n)$, and low stretch with respect to the leaf nodes.

The final step is to convert the incubator graph $\mathcal{H}$ (which contains more than n nodes) to a spanner H for the original n-point metric space. One way is to label each node in the net-tree with the corresponding net-point in X. Then, an edge between two nodes induces an edge between their labels. However, this labeling is problematic, due to the hierarchical property of the nets. Although $\mathcal{H}$ has constant degree, if a point in X is used as a label for many nodes, that point will accumulate a large degree. In particular, the point associated with the root node is a net-point at every level, hence this gives rise to a large degree.

The key idea is simple and has been used in [1,11]: we label each node with a point that is nearby with respect to the relevant distance scale (which means that small stretch will still be preserved), such that each label is used only a constant number of times. This guarantees that the degree of the resulting spanner will be constant. In order to describe the labeling process clearly, we find it convenient to use the incubator-zombie terminology.

Incubators Working with Zombies. Each incubator will receive a label that we refer to as a *zombie*, which is identified with a point in X. The incubator graph $\mathcal{H}$ and the zombies naturally induce a spanner on X: if there is an edge between two incubators, then there is an induced edge between the corresponding zombies. It is left to show how we assign zombies to incubators. Each leaf incubator can be simply assigned its original net-point, as there is a 1-1 correspondence between leaf incubators and points in X. Also, since each internal incubator has at least two children in the contracted incubator tree (ConIncTree), each internal incubator can be assigned a unique nearby zombie from its descendant leaves. (To achieve fault-tolerance, we use a *zombie-climbing* process that involves using both ConIncTree and ConRouTree; we will guarantee that each internal incubator holds up to $k+1$ nearby zombies, and each point appears as a zombie in $O(k)$ incubators. We provide the details in Section 5.)

Sketch Analysis. Since each incubator contains a nearby zombie, small stretch and lightness can still be preserved. Since the incubator graph $\mathcal{H}$ has constant degree and each point in X can be the identity of at most two zombies, the

degree of the resulting spanner H is constant too. The hop-diameter of H is $O(\log n)$, as any spanner path (between leaf nodes) in $\mathcal{H}$ with l hops gives rise to a spanner path in H with at most l hops. Finally, the running time is dominated by the subroutines that our construction uses, which have been shown (in the relevant references) to take $O(n \log n)$ time.

2 Preliminaries

Let (X, d) be an n-point doubling metric, and let $1 \leq k \leq n-2$ be the maximum number of failed nodes allowed. We consider the regime of stretch $1 + \epsilon$, for an arbitrarily small $0 < \epsilon < \frac{1}{2}$. We assume that the minimum inter-point distance of X is 1, and let $\Delta := \max_{u,v \in X} d(u, v)$ be the *diameter* of X.

The ball of radius $r > 0$ centered at x is $B(x, r) := \{u \in X : d(x, u) \leq r\}$. A set $Y \subseteq X$ is called an *r-cover* of X if for any point $x \in X$ there is a point $y \in Y$, with $d(x, y) \leq r$. A set Y is an *r-packing* if for any pair of distinct points $y, y' \in Y$, it holds that $d(y, y') > r$. For $r_1 \geq r_2 > 0$, we say that a set $Y \subseteq X$ is an *(r_1, r_2)-net* for X if Y is both an r_1-cover of X and an r_2-packing. Note that such a net can be constructed by a greedy algorithm. By recursively applying the definition of doubling dimension, we can get the following key fact [12].

Fact 1 (Nets Have Small Size [12]). *Let $R \geq 2r > 0$ and let $Y \subseteq X$ be an r-packing contained in a ball of radius R. Then, $|Y| \leq (\frac{R}{r})^{2\dim}$.*

The minimum spanning tree of a metric space (X, d) is denoted by $\mathsf{MST}(X)$ (or simply MST if (X, d) is clear from the context). Also, we denote by $w(\mathsf{MST})$ the weight of MST. Given a spanner H for (X, d), the *lightness* of H is defined as the ratio of the weight of H to the weight of MST.

Fact 2 (Two Lower Bounds for $w(\mathsf{MST})$). *1. $w(\mathsf{MST}) \geq \Delta$.*
2. Let $S \subseteq X$ be an r-packing, with $r \leq \Delta$. Then, $w(\mathsf{MST}) \geq \frac{1}{2} r \cdot |S|$.

Hierarchical Nets. We consider the hierarchical nets that are used by Gottlieb and Roditty [11]. Let $r_i := 5^i$ and $\ell := \lceil \log_5 \Delta \rceil$. Also, let $\{N_i\}_{i \geq 0}^{\ell}$ be a sequence of hierarchical nets, where $N_0 := X$ and for each $i \geq 1$, N_i is a $(3r_i, r_i)$-net for N_{i-1}. (Observe that N_ℓ contains one point.) As mentioned in [11], this choice of parameters is needed to achieve running time $O(n \log n)$.

Net-Tree with Cross Edge Framework. We recap the basic spanner construction [2,11] using the incubator-zombie terminology.
Incubators. For each level i and each $x \in N_i$, there is a corresponding *incubator* $C = (x, i)$, where x is the *identity* of the incubator and i is its level. For $C_1 = (x_1, i_1)$ and $C_2 = (x_2, i_2)$, define $d(C_1, C_2) := d(x_1, x_2)$; for $C_1 = (x_1, i_1)$ and $x_2 \in X$, define $d(C_1, x_2) := d(x_1, x_2)$.

Incubator Tree. The hierarchical nets induce a tree structure (hereafter, the *incubator tree* $\mathsf{IncTree}$) on the incubators as follows. The only incubator at level ℓ is the root, and for each level $0 \leq i < \ell$, each incubator at level i (hereafter,

level-i incubator) has a parent at level $i+1$ within distance $3r_{i+1}$. (Recall that N_{i+1} is a $3r_{i+1}$-cover for N_i.) Hence, every descendant of a level-i incubator can reach it by climbing a path of weight at most $\sum_{j \le i} 3r_j \le 4r_i$.

Cross Edges. In order to achieve stretch $(1+\epsilon)$, in each level cross edges are added between incubators that are close together with respect to the distance scale at that level. Specifically, for each level $0 \le i < \ell$, for all $u, v \in N_i$ such that $u \ne v$ and $d(u,v) \le \gamma r_i$, for some appropriate parameter $\gamma = O(\frac{1}{\epsilon})$, we add a cross edge between the corresponding incubators (u,i) and (v,i) (with weight $d(u,v)$). The basic spanner construction is obtained as the union of the tree ($\mathsf{IncTree}$) edges and the cross edges. The following lemma gives the essence of the cross edge framework; a variant of this lemma appears in [2, Lemma 5.1] and [9]. Since we shall later reroute spanner paths and assign internal incubators with labels (which we refer to as *zombies*), we also give an extended version here.

Lemma 1 (Cross Edge Framework Guarantees Low Stretch). *Consider the cross edge framework as described above.*
(a) *Let $\mu > 0$ be an arbitrary constant. Suppose a graph $\mathcal{H}$ on the incubators contains all cross edges (defined with some appropriate parameter γ depending on μ and ϵ), and for each level $i \ge 1$, each level-$(i-1)$ incubator is connected via a tree edge to some level-i incubator within distance μr_i. Then, $\mathcal{H}$ contains a $(1+\epsilon)$-spanner path $P_{u,v}$ for each $u, v \in X$, obtained by climbing up from the leaf incubators corresponding to u and v to some level-j ancestors u' and v', respectively, where u' and v' are connected by a cross edge and $r_j = O(\epsilon) \cdot d(u,v)$.*
(b) *In the above graph $\mathcal{H}$, if for each level $i \ge 1$, each level-i incubator (u,i) is labeled with a point $\widehat{u}$ such that $d(u,\widehat{u}) = O(r_i)$, then the path $\widehat{P_{u,v}}$ induced by the above path $P_{u,v}$ and the labels is a $(1+O(\epsilon))$-spanner path.*

Lonely Incubators. An incubator is called *lonely* if it has exactly one child incubator (which has the same identity as the parent); otherwise it is *non-lonely*. Observe that the leaf incubators have no children and are non-lonely. For efficiency reasons, a long chain of lonely incubators will be represented implicitly; implementation details can be found in [11]. As we shall later see, for the *zombie-climbing* process (described in Section 5) to succeed we need the property that each internal incubator has at least two children. Observe that at the bottom of a chain $\mathcal{C}$ of lonely incubators is a non-lonely incubator C with the same identity. We shall later contract a chain $\mathcal{C}$ of lonely incubators (together with the non-lonely incubator C at the bottom) into a *super incubator*.

Running Time. All the subroutines that our construction uses have been shown (in the relevant references) to take $O(n \log n)$ running time. Hence, we disregard the running time analysis, except for places which require clarification.

Challenges Ahead. Lemma 1 can be used to achieve low stretch. As lonely incubators need to be contracted, the next issue is that many cross edges will be inherited by the super incubator, which may explode the degree. To overcome this obstacle, in Section 3 we use Gottlieb and Roditty's technique [11] to reroute spanner paths and prune redundant cross edges. In Section 4 we show that

hop-diameter $O(\log n)$ (and lightness $O(\log n)$ too) can be achieved by applying Solomon and Elkin's shortcut spanner [15] to all subtrees at sufficiently small distance scales (less than $\frac{\Delta}{n}$). In Section 5 we describe a zombie-climbing process, which converts a graph on the incubators to a k-fault tolerant spanner on X with all the desired properties, thereby completing the proof of Theorem 1.

3 Reducing Degree via Gottlieb-Roditty's Spanner

By Fact 1, each internal incubator has at most $O(1)^{O(\dim)}$ children in $\mathsf{IncTree}$, and each incubator is incident on at most $\epsilon^{-O(\dim)}$ cross edges. However, the problem that arises is that when a chain of lonely incubators is contracted, the cross edges that are incident on the corresponding super incubator may explode its degree. We employ the *parent replacement* technique due to Gottlieb and Roditty [11] to reroute spanner paths and make some cross edges *redundant*. Our procedure below is a simple modification of subroutines that appear in [11], and hence can be implemented within the same running time $O(n \log n)$.

Routing Tree. We carry out the parent replacement procedure by constructing a *routing tree* ($\mathsf{RouTree}$), which has the same leaf incubators as $\mathsf{IncTree}$; moreover, each level-$(i-1)$ child incubator is connected to a level-i parent incubator with an edge of possibly heavier weight at most $5r_i$ (as opposed to weight at most $3r_i$ as in $\mathsf{IncTree}$). Consider a non-root incubator C in the (uncontracted) $\mathsf{IncTree}$. The parent incubator of C in $\mathsf{RouTree}$ is determined by the following rules.

(1) If either the parent C' of C in $\mathsf{IncTree}$ or the parent of C' is non-lonely, then C will have the same parent C' in $\mathsf{RouTree}$.
(2) For a chain of at least two lonely incubators, we start from the bottom incubator $C_i = (x, i)$ (which is non-lonely) at some level i. If the parent C_{i+1} of $C_i = (x, i)$ in $\mathsf{IncTree}$ is lonely and the parent of C_{i+1} is also lonely, then we try to find a new parent for C_i. Specifically, if there is some point $w \in N_{i+1} \setminus \{x\}$ such that $d(x, w) \leq 5r_{i+1}$ (if there is more than one such point w, we can pick one arbitrarily), then a non-lonely *adopting parent* is found as follows.
(a) If the incubator $\widehat{C}_{i+1} = (w, i+1)$ is non-lonely, then $\widehat{C}_{i+1}$ is designated as the adopting parent of C_i (which means that $\widehat{C}_{i+1}$ will be C_i's parent in $\mathsf{RouTree}$). Similarly, C_i is designated as an *adopted child* of $\widehat{C}_{i+1}$.
(b) If the incubator $\widehat{C}_{i+1}$ is lonely, then it does not adopt C_i. However, we shall see in Lemma 2 that the parent $\widehat{C}_{i+2}$ of $\widehat{C}_{i+1}$ cannot be lonely. Moreover, it is close enough to C_{i+1}, namely, $d(C_{i+1}, \widehat{C}_{i+2}) \leq 5r_{i+2}$. In this case $\widehat{C}_{i+2}$ will adopt C_{i+1} (and will be its parent in $\mathsf{RouTree}$), and C_{i+1} will remain C_i's parent.

Observe that once an adopting parent is found, there is no need to find adopting parents for the rest of the lonely ancestors in the chain, because these lonely ancestors will not adopt, and so they will not be used for routing.

If there is no such nearby point $w \in N_{i+1} \setminus \{x\}$ for $C_i = (x, i)$, then C_i's parent in $\mathsf{RouTree}$ remains C_{i+1}, and we continue to climb up the chain.

Lemma 2 (Lonely Incubators Need Not Adopt). *[Proof in full version] Suppose that an incubator $C_i = (x, i)$ has at least two lonely ancestors (excluding C_i) and there exists a point $w \in N_{i+1} \setminus \{x\}$, such that $d(x, w) \leq 5r_{i+1}$ and the incubator $\widehat{C}_{i+1} = (w, i+1)$ is lonely. Then, the parent $\widehat{C}_{i+2} = (u, i+2)$ of $\widehat{C}_{i+1}$ in* IncTree *is non-lonely; moreover, $d(x, u) \leq 5r_{i+2}$.*

Rerouting Spanner Paths. When we wish to find a spanner path between u and v, we use RouTree to climb to the corresponding ancestor incubators (from an appropriate level) which are connected by a cross edge. Observe that using RouTree, it is still possible to climb from a level-i incubator to a level-$(i+1)$ incubator that is within distance $O(r_{i+1})$ from it. Hence, by Lemma 1, stretch $1 + \epsilon$ can still be preserved.

Degree Analysis. Notice that only non-lonely incubators can adopt, and by Fact 1, each incubator can have only $O(1)^{O(\dim)}$ adopted child incubators. Hence, the degree of RouTree is $O(1)^{O(\dim)}$. Consider a chain of lonely incubators with identity x. If some incubator $C = (x, i)$ has found an adopting parent, then all lonely ancestors of C in the chain will not be used for routing (since a lonely incubator cannot adopt). Thus the cross edges incident on those unused lonely ancestors are redundant. Next, we show that the number of useful cross edges accumulated by a chain of lonely incubators (until an adoption occurs) is small.

Lemma 3 (No Nearby Net-Points Implies Few Cross Edges). *[Proof in full version] Suppose that $x \in N_i$ and all other points in N_i are at distance more than $5r_i$ away (i.e., no adopting parent is found for any child of (x, i)). Then, there are at most $O(\gamma)^{O(\dim)} = \epsilon^{-O(\dim)}$ cross edges with level at most i connecting incubators with identity x and non-descendants of (x, i).*

Pruning Cross Edges and Routing Tree. After the lowest level incubator in a chain of lonely incubators finds an adopting parent, redundant cross edges incident on the ancestors of the adopted incubator can be pruned. By Lemma 3, no matter if an adopting parent is found for a chain, at most $\epsilon^{-O(\dim)}$ remaining cross edges are incident on incubators in the chain for the following reason.

- If no adopting parent is found for a chain, Lemma 3 can be applied to the second lonely incubator (if any) from the top of the chain. Since the top incubator has only $\epsilon^{-O(\dim)}$ cross edges, the entire chain has $\epsilon^{-O(\dim)}$ incident cross edges.
- If some point $w \in N_{i+1} \setminus \{x\}$ is found for an incubator $C_i = (x, i)$ in the parent replacement process described above, Lemma 3 can be applied to C_i (unless C_i is the bottom incubator in the chain, and then we do not need to apply the lemma). Since either C_i or its parent will be adopted (and the cross edges incident on the ancestors of the adopted child are redundant, and will be pruned), it follows that the entire chain has $\epsilon^{-O(\dim)}$ remaining incident cross edges.

For efficiency reasons, observe that we can first build RouTree and add cross edges only for incubators that are actually used for routing. In other words, we do not have to add redundant cross edges that will be pruned later.

Contraction Phase. After finishing the construction of RouTree, we start the contraction phase, which involves contracting all chains of lonely incubators. The above argument implies that the number of cross edges incident on a super incubator is at most $\epsilon^{-O(\dim)}$. Also, since lonely incubators cannot adopt, the degree of the routing tree cannot increase.

• If no adopting parent is found for a chain, the chain will be contracted to a super incubator C, which has the same parent $\overline{C}$ (that itself may be a super incubator corresponding to a contracted chain) in the *contracted incubator tree* (ConIncTree) and the *contracted routing tree* (ConRouTree).
• If an adopting parent $\widehat{C}$ is found for a chain, the chain will be contracted to a super incubator C, whose parent $\overline{C}$ in ConRouTree is different from its parent $\tilde{C}$ in ConIncTree; either one among $\overline{C}$ and $\tilde{C}$ can be a super incubator.

Multiple edges are removed from the resulting multi-graph, keeping just the edge of minimum level between any pair of incident (super) incubators.

Corollary 1 (Constant Degree). ConRouTree *has degree* $O(1)^{O(\dim)}$*, and each incubator (and also super incubator) has* $\epsilon^{-O(\dim)}$ *cross edges.*

4 Achieving Small Hop-Diameter and Lightness

Consider ConRouTree constructed in Section 3. Observe that if the maximum inter-point distance Δ is large enough (exponential in n), the hop-diameter will be as large as $\Theta(n)$. In this section we add edges to shortcut ConRouTree, such that for each level i, any leaf incubator can reach some level-i incubator in $O(\log n)$ hops and within distance $O(r_i)$; this guarantees that the hop-diameter is $O(\log n)$. We also make sure that the lightness will be in check.

Theorem 2 (Spanner Shortcut [15]). *Let T be a tree (whose edges have positive weights) with n nodes and degree* $\mathsf{deg}(T)$*. For the tree metric induced by the shortest-path distances in T, a 1-spanner J with $O(n)$ edges, degree at most* $\mathsf{deg}(T)+4$*, and hop-diameter $O(\log n)$ can be constructed in $O(n \log n)$ time.*

Levels of Super Incubators and Edges. Technically, a super incubator is of the same level as the non-lonely incubator at the bottom of the corresponding chain. The level of an edge before the contractions is defined as the maximum level of its endpoint incubators, and its level after the contractions remains the same.

Shortcut the Low Levels of ConRouTree**.** Let $\widehat{r} = \frac{\Delta}{n}$, and define $\sigma := \lfloor \log_5 \widehat{r} \rfloor$. Observe that the number of levels above σ is $O(\log n)$. We shortcut all maximal subtrees rooted at level at most σ in ConRouTree, i.e., the root of each such subtree is at level at most σ, but its parent is at level greater than σ. Each such subtree is shortcut via the 1-spanner that is given by Theorem 2. Notice that this shortcut procedure adds edges that enable going from each leaf incubator to any of its ancestor incubators in ConRouTree tree in $O(\log n)$ hops. This implies that for any $u, v \in X$, there is a spanner path between the corresponding leaf

incubators with $O(\log n)$ hops and weight at most $(1+\epsilon)\cdot d(u,v)$. Moreover, the shortcut procedure increases the degree of each incubator by at most four.

Large vs Small Scales. We analyze the weight of the spanner by considering the weight contribution from large-scale edges and small-scale edges separately. An edge has a *small scale* if its level is at most σ (we use the convention that a shortcut edge has a small scale); otherwise, it has a *large scale*. We remark that the following lemma remains valid if we increase the weight of each level-i edge by $O(r_i)$; this observation will be used later (in Section 5) when we apply our labeling procedure.

Lemma 4 (Edges are Light). *[Proof in full version] The total weight of all large-scale edges is* $\epsilon^{-O(\dim)}\cdot\log n\cdot w(\mathsf{MST})$*, and the total weight of all small-scale edges is* $\epsilon^{-O(\dim)}\cdot w(\mathsf{MST})$*.*

Incubator Graph $\mathcal{H}$. To summarize, we build an *incubator graph* $\mathcal{H}$ on the incubators (and super incubators) that consists of (1) the edges in ConRouTree, (2) useful cross edges that remain after pruning, and (3) edges used to shortcut the low levels (at most σ) of ConRouTree. The incubator graph $\mathcal{H}$ has degree $\epsilon^{-O(\dim)}$. Moreover, for any $u, v \in X$, there is a path in $\mathcal{H}$ between the corresponding leaf incubators with $O(\log n)$ hops and weight at most $(1+\epsilon)\cdot d(u,v)$. Also, Lemma 4 implies that it has weight $\epsilon^{-O(\dim)}\cdot\log n\cdot w(\mathsf{MST})$. Finally, it is easy to see that $\mathcal{H}$ can be implemented within $\epsilon^{-O(\dim)}\cdot n\log n$ time.

In other words, the incubator graph $\mathcal{H}$ has almost all the desired properties, except that each point in X may be the identity of many incubators (and super incubators). Moreover, we have not considered fault tolerance so far. We will address these issues in Section 5.

5 Incubators Working with Zombies: Fault Tolerance

The incubator graph $\mathcal{H}$ defined at the end of Section 4 achieves all the desired properties, except that the same point may be the identity of many incubators. Moreover, there is a 1-1 correspondence between the leaf incubators and the points in X. In this section we show how to convert $\mathcal{H}$ into a k-fault tolerant spanner H for X that satisfies all the desired properties. To this end we devise a simple labeling procedure (that we refer to below as the *zombie-climbing procedure*), which is convenient to describe via the incubator-zombie terminology.

Zombies. A *zombie* is identified by a point $x \in X$, which is the *identity* of the zombie. When the context is clear, we do not distinguish between a zombie and its identity. After the zombie-climbing procedure is finished, each leaf incubator will contain a zombie whose identity is the same as the leaf's identity, and each internal incubator will contain up to $k+1$ zombies with distinct identities.

Induced Spanner on X. The incubator graph $\mathcal{H}$ together with the zombies induce a spanner H on X in a natural way: two points u and v are neighbors in H if

there are zombies with identities u and v residing in neighboring incubators in $\mathcal{H}$. Hence, it suffices to describe how zombies are assigned to incubators.

The Zombies Are Climbing... We assign zombies to incubators in two stages. In the first stage, we use ConIncTree to assign a single *host* zombie to each incubator; in the second stage, for each internal incubator, we use ConRouTree to collect up to k additional *guest* zombies with distinct identities, which are also different from the identity of the host zombie.

First Stage. Consider ConIncTree, and note that each internal incubator has at least two children. Each incubator is assigned a host zombie as follows. A leaf incubator C creates two zombies with the same identity as itself; one stays in C as its host zombie and the other climbs to C's parent. An internal incubator $\widehat{C}$ receives exactly one zombie from each of its (at least two) children. One of these zombies stays in $\widehat{C}$ as its host zombie, and another one (chosen arbitrarily) climbs to $\widehat{C}$'s parent incubator (if any); extra zombies (which do not become host zombies) are discarded. Since there are $O(n)$ incubators, this procedure takes $O(n)$ time. Observe that each point can appear as the identity of at most two host zombies: once in a leaf incubator, and at most once in an internal incubator.

Second Stage. We use ConRouTree in this step. Each internal incubator C collects up to k guest zombies with distinct identities from the host zombies of C's descendants (in ConRouTree) using a bounded breadth-first search. Specifically, when a descendant incubator $\tilde{C}$ is visited, if its host zombie $\tilde{z}$ is different from the host zombie of C and from all the guest zombies already collected by C, then $\tilde{z}$ will be collected as one of C's guest zombies. The breadth-first search terminates once C has collected k distinct guest zombies or when all C's descendants in ConRouTree have been visited. Since for each breadth-first search, $O(k)$ incubators are visited (as each point can appear as the identity of at most two host zombies), the second stage can be implemented within $O(kn)$ time.

Lemma 5 (Each Point Appears as $O(k)$ Zombies). *[Proof in full version] Each point can be the identity of a zombie in at most $2k+2$ incubators (as either a host or a guest).*

Fault Tolerance. Recall that in the cross edge framework, the low-stretch spanner path between any two points u and v is obtained as follows. We start from the two leaf incubators corresponding to u and v, and climb to the ancestor incubators (according to ConRouTree in our case) in some appropriate level, where a cross edge is guaranteed to exist. Observe that only incubators with the same identity will be contracted, and so this does not change the stretch of the spanner path. The two following lemmas show that for any functioning point u, each ancestor of the leaf incubator corresponding to u (in ConRouTree) contains at least one nearby functioning zombie. Consequently, fault-tolerance is achieved.

Lemma 6 (Every Ancestor in ConRouTree Has a Functioning Zombie). *[Proof in full version] Suppose that at most k points fail. Let u be an arbitrary functioning point, and let C_u be the leaf incubator corresponding to u. Then, every ancestor of C_u in* ConRouTree *contains at least one functioning zombie.*

Lemma 7 (Incubators Contain Nearby Zombies). *[Proof in full version] If an incubator $C = (x, i)$ contains a zombie z, then $d(x, z) = O(r_i)$.*

Tying Up Everything Together – Completing the Proof of Theorem 1 *Stretch and Fault-Tolerance.* Lemmas 6 and 7 state that each incubator contains a functioning zombie that is nearby, which implies that a level-i incubator edge will induce a functioning zombie edge with weight that is greater by at most an additive factor of $O(r_i)$. The second assertion of Lemma 1 implies that the stretch of the resulting spanner is $1 + O(\epsilon)$; we can achieve stretch $(1 + \epsilon)$ by rescaling γ (and other parameters) by an appropriate constant.

Degree. Since the incubator graph $\mathcal{H}$ has degree $\epsilon^{-O(\dim)}$ and each incubator contains at most $k+1$ zombies, it follows that each occurrence of a point as the identity of some zombie will incur a degree of at most $\epsilon^{-O(\dim)} \cdot k$. By Lemma 5, each point can be the identity of at most $2k+2$ zombies, which implies that the degree of H is at most $\epsilon^{-O(\dim)} \cdot k^2$.

Hop-Diameter. Observe that any spanner path in $\mathcal{H}$ (between leaf nodes) with l hops induces a functioning spanner path in H with at most l hops. Hence the hop-diameter of H is $O(\log n)$.

Lightness. As already mentioned, in the proof of Lemma 4, the upper bound still holds if we add $O(r_i)$ to the weight of each level-i incubator edge. Moreover, as only zombies within distance $O(r_i)$ are assigned to a level-i incubator, it follows that each level-i zombie edge has weight at most an additive factor $O(r_i)$ greater than that of the inducing incubator edge. Since every incubator edge induces at most $O(k^2)$ zombie edges, we conclude that the lightness of H is $\epsilon^{-O(\dim)} \cdot k^2 \log n$.

Running Time. As mentioned in Sections 3 and 4, our construction uses subroutines from Gottlieb-Roditty's spanner [11] and Elkin-Solomon's shortcut spanner [15], which have running time at most $\epsilon^{-O(\dim)} \cdot n \log n$. Also, the zombie-climbing procedure takes time $O(kn)$. Finally, observe that there are $\epsilon^{-O(\dim)} \cdot n$ edges in $\mathcal{H}$, each of which induces $O(k^2)$ zombie edges; hence, transforming the incubator graph $\mathcal{H}$ into the ultimate spanner H takes $\epsilon^{-O(\dim)} \cdot k^2 n$ time.

Acknowledgments. The fourth-named author is grateful to Michael Elkin and Michiel Smid for helpful discussions.

References

1. Arya, S., Das, G., Mount, D.M., Salowe, J.S., Smid, M.H.M.: Euclidean spanners: short, thin, and lanky. In: STOC, pp. 489–498 (1995)
2. Chan, H.T.-H., Gupta, A., Maggs, B.M., Zhou, S.: On hierarchical routing in doubling metrics. In: SODA, pp. 762–771 (2005)

3. Chan, T.-H.H., Gupta, A.: Small hop-diameter sparse spanners for doubling metrics. Discrete & Computational Geometry 41(1), 28–44 (2009)
4. Chan, T.-H.H., Li, M., Ning, L.: Incubators vs zombies: Fault-tolerant, short, thin and lanky spanners for doubling metrics. CoRR, abs/1207.0892 (2012)
5. Chan, T.-H.H., Li, M., Ning, L.: Sparse fault-tolerant spanners for doubling metrics with bounded hop-diameter or degree. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 182–193. Springer, Heidelberg (2012)
6. Chew, P.: There is a planar graph almost as good as the complete graph. In: SoCG, pp. 169–177 (1986)
7. Dinitz, Y., Elkin, M., Solomon, S.: Shallow-low-light trees, and tight lower bounds for Euclidean spanners. In: FOCS, pp. 519–528 (2008)
8. Elkin, M., Solomon, S.: Optimal Euclidean spanners: really short, thin and lanky. In: STOC (to appear, 2013)
9. Gao, J., Guibas, L.J., Nguyen, A.: Deformable spanners and applications. In: SoCG, pp. 190–199 (2004)
10. Gottlieb, L.-A., Roditty, L.: Improved algorithms for fully dynamic geometric spanners and geometric routing. In: SODA, pp. 591–600 (2008)
11. Gottlieb, L.-A., Roditty, L.: An optimal dynamic spanner for doubling metric spaces. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 478–489. Springer, Heidelberg (2008)
12. Gupta, A., Krauthgamer, R., Lee, J.R.: Bounded geometries, fractals, and low-distortion embeddings. In: FOCS, pp. 534–543 (2003)
13. Roditty, L.: Fully dynamic geometric spanners. In: SoCG, pp. 373–380 (2007)
14. Solomon, S.: Fault-tolerant spanners for doubling metrics: Better and simpler. CoRR, abs/1207.7040 (2012)
15. Solomon, S., Elkin, M.: Balancing degree, diameter and weight in Euclidean spanners. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 48–59. Springer, Heidelberg (2010)

Maximum Edge-Disjoint Paths in k-Sums of Graphs*

Chandra Chekuri[1,**], Guyslain Naves[2], and F. Bruce Shepherd[3,***]

[1] Dept. of Computer Science, University of Illinois, Urbana, IL 61801, USA
chekuri@illinois.edu
[2] Laboratoire d'Informatique Fondamentale, Faculté des Sciences de Luminy, Marseille, France
guyslain.naves@lif.univ-mrs.fr
[3] Dept. Mathematics and Statistics, McGill University, Montreal, Canada
bruce.shepherd@mcgill.ca

Abstract. We consider the approximability of the maximum edge-disjoint paths problem (MEDP) in undirected graphs, and in particular, the integrality gap of the natural multicommodity flow based relaxation for it. The integrality gap is known to be $\Omega(\sqrt{n})$ even for planar graphs [11] due to a simple topological obstruction and a major focus, following earlier work [14], has been understanding the gap if some constant congestion is allowed. In planar graphs the integrality gap is $O(1)$ with congestion 2 [19,5]. In general graphs, recent work has shown the gap to be $O(\text{polylog}(n))$ [8,9] with congestion 2. Moreover, the gap is $\Omega(\log^{\Omega(c)} n)$ in general graphs with congestion c for any constant $c \geq 1$ [1].

It is natural to ask for which classes of graphs does a constant-factor constant-congestion property hold. It is easy to deduce that for given constant bounds on the approximation and congestion, the class of "nice" graphs is minor-closed. Is the converse true? Does every proper minor-closed family of graphs exhibit a constant-factor constant-congestion bound relative to the LP relaxation? We conjecture that the answer is yes. One stumbling block has been that such bounds were not known for bounded treewidth graphs (or even treewidth 3). In this paper we give a polytime algorithm which takes a fractional routing solution in a graph of bounded treewidth and is able to integrally route a constant fraction of the LP solution's value. Note that we do not incur any edge congestion. Previously this was not known even for series parallel graphs which have treewidth 2. The algorithm is based on a more general argument that applies to k-sums of graphs in some graph family, as long as the graph family has a constant-factor constant-congestion bound. We then use this to show that such bounds hold for the class of k-sums of bounded genus graphs.

1 Introduction

The disjoint paths problem is the following: given an undirected graph $G = (V, E)$ and node pairs $H = \{s_1t_1, \ldots, s_pt_p\}$, are there disjoint paths connecting the given pairs? We use NDP and EDP to refer to the version in which the paths are required to

* A full version of this article is available at http://arxiv.org/abs/1303.4897
** Partially supported by NSF grant CCF-1016684.
*** Work partly done while visiting Microsoft Research in 2011-12 and supported by grants from NSERC.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 328–339, 2013.

be node-disjoint or edge-disjoint. Disjoint path problems are cornerstone problems in combinatorial optimization. The seminal work on graph minors of Robertson and Seymour [17] gives a polynomial time algorithm for NDP (and hence also for EDP) when p is fixed; the algorithmic and structural tools developed for this have led to many other fundamental results. In contrast to the undirected case, the problem in directed graphs is NP-Complete for $p = 2$ [10]. Further, NDP and EDP are NP-Complete in undirected graphs when p is part of the input. The maximization versions of EDP and NDP have also attracted intense interest, especially in connection to its approximability. In the *maximum edge-disjoint path* (MAX EDP) problem we are given an undirected (in this paper) graph $G = (V, E)$, and node pairs $H = \{s_1t_1, \ldots, s_pt_p\}$, called *commodities* or *demands*. MAX EDP asks for a maximum size subset $I \subseteq \{1, 2, \ldots, p\}$ of commodities which is *routable*. A set I is routable if there is a family of edge-disjoint paths $(P_i)_{i\in I}$ where P_i has extremities s_i and t_i for each $i \in I$. In a more general setting, the edges have integer capacities $c : E(G) \to \mathbb{N}$, and instead of edge-disjoint paths, we ask that for each edge $e \in E(G)$, at most $c(e)$ paths of $(P_i)_{i\in I}$ contain e. For any demand $h = st \in H$, denote by $\mathcal{P}_h$ the set of st-paths in G, and $\mathcal{P} = \bigcup_{h\in H} \mathcal{P}_h$. A natural linear programming relaxation of MAX EDP is then:

$$\begin{array}{lll} \max \sum_{h\in H} z_h & \text{subject to} & \\ \quad \sum_{P\in\mathcal{P}_h} x_P = z_h \leq 1 & \quad \text{(for all } h \in H) & \\ \quad \sum_{P\in\mathcal{P}, e\in P} x_P \leq c_e & \quad \text{(for all } e \in E) & \\ \quad x \geq 0 & & \end{array} \tag{1}$$

NP-Completeness of EDP implies that MAX EDP is NP-Hard. In fact, MAX EDP is NP-Hard in capacitated trees for which EDP is trivially solvable. This indicates that MAX EDP inherits hardness also from the selection of the subset of demands to route. As pointed out in [11], a grid example shows that the integrality gap of the multicommodity flow relaxation may be as large as $\Omega(\sqrt{n})$ even in planar graphs. However, the grid example is not robust in the sense that if we allow edge-congestion 2 (or equivalently, if we assume all capacities are initially at least 2), then the example only has a constant-factor gap. This observation led Kleinberg-Tardos [14] to seek better approximations (polylog or constant-factor) for planar graphs in the regime where some low congestion is allowed. With some work, this agenda proved fruitful: a constant-approximation with edge congestion 4 was proved possible in planar graphs [5]; this was improved to (an optimal) edge congestion 2 in [19].

In general graphs, Chuzhoy [8] recently obtained the first poly-logarithmic approximation with constant congestion (14). This was subsequently improved to the optimal congestion of 2 by Chuzhoy and Li [9]. It is also known that, in general graphs, the integrality gap of the flow LP is $\Omega(\log^{\Omega(1/c)} n)$ even if congestion c is allowed; the known hardness of approximation results for MAX EDP with congestion have similar bounds as the integrality gap bounds, see [1].

For any constants $\alpha, \beta \geq 1$, one may ask for which graphs does the LP for MAX EDP admit an integrality gap of α if edge congestion β is allowed. It is natural to require this for any possible collection of demands and any possible assignment of edge capacities. For fixed constants, it is easy to see that the class of such graphs is closed under minors. Is the converse true? That is, do all minor-closed graphs exhibit a

constant-factor constant-congestion (CFCC) integrality gap for MAX EDP? In fact we consider the following stronger conjecture with congestion 2.

Conjecture 1. *Let $\mathcal{G}$ be any proper minor-closed family of graphs. Then the integrality gap of the flow LP for* MAX EDP *is at most a constant $c_{\mathcal{G}}$ when congestion 2 is allowed.*

The preceding conjecture is inherently a geometric question, but one would also anticipate a polytime algorithm for producing the routable sets which establish the gap. In attempting to prove Conjecture 1, one must delve into the structure of minor-closed families of graphs, and in particular the characterization given by Robertson and Seymour [17]. Two minor-closed families that form the building blocks for this characterization are (i) graphs embedded on surfaces of bounded genus (in particular planar graphs), and (ii) graphs with bounded treewidth. For MAX EDP, we have a constant-factor integrality gap with congestion 2 for planar graphs. In [6] it is shown that the integrality gap of the LP for MAX EDP in graphs of treewidth at most k is $O(k \log k \log n)$; note that this is with congestion 1. Existing integrality gap results, when interpreted in terms of treewidth k, show that the integrality gap is $\Omega(k)$ for congestion 1 and $\Omega(\log^{O(1/c)} k)$ for congestion $c > 1$. It was asked in [6] whether the gap is $O(k)$ with congestion 1. In particular, the question of whether the gap is $O(1)$ for $k = 2$ (this is precisely the class of series parallel graphs) was open. In this paper we show the following result.

Theorem 1. *The integrality gap of the flow LP for* MAX EDP *is $2^{O(k)}$ in graphs of treewidth at most k. Moreover, there is a polynomial-time algorithm that given a graph G, a tree decomposition for G of width k, and fractional solution to the LP of value* OPT*, outputs an integral solution of value $\Omega(\mathsf{OPT}/2^{O(k)})$.*

The preceding theorem is a special case of a more general theorem that we prove below. Let $\mathcal{G}$ be a family of graphs. For any integer $k \geq 1$, let $\mathcal{G}_k$ denote the class of graphs obtained from $\mathcal{G}$ by the k-sum operation. The k-sum operation is formally defined in Section 2.1; the structure theorem of Robertson and Seymour is based on the k-sum operation over certain classes of graphs.

Theorem 2. *Let $\mathcal{G}$ be a minor-closed class of graphs such that the integrality gap of the flow LP is α with congestion β. Then the integrality gap of the flow LP for the class $\mathcal{G}_k$ is $2^{O(k)}\alpha$ with congestion $\beta + 3$.*

The preceding theorem is effective in the following sense: there is a polynomial-time algorithm that gives a constant-factor, constant congestion result for $\mathcal{G}_k$ assuming that (i) such an algorithm exists for $\mathcal{G}$ and (ii) there is a polynomial-time algorithm to find a tree decomposition over $\mathcal{G}$ for a given graph $G \in \mathcal{G}_k$.

We give the following as a second piece of evidence towards Conjecture 1.

Theorem 3. *The integrality gap of the flow LP on graphs of genus $g > 0$ is $O(g \log^2(g+1))$ with congestion 3.*[1]

[1] We believe that the congestion bound in the preceding theorem can be improved to 2 with some additional technical work. We do not give a polynomial-time algorithm although we believe that it too is achievable with some (potentially messy) technical work.

Theorems 2 and 3 imply that the class of graphs obtained as k-sums of graphs with genus g is CFCC when k and g are fixed constants. The bottleneck in extending our results to prove Conjecture 1 are planar graphs (or more generally bounded genus graphs) that have "vortices" which play a non-trivial role in the Robertson-Seymour structure theorem.

A Brief Discussion of Technical Ideas and Related Work: The approximability of MAX EDP in undirected and directed graphs has received much attention in the recent years. We refer the reader to some recent papers [9,8,19,1]. A framework based on well-linked decompositions [3] has played an important role in understanding the integrality gap of the flow relaxation in undirected graphs. It is based on recursively cutting the input graph along sparse cuts until the given instance is well-linked. However, this framework loses at least a logarithmic factor in the approximation. The work in [5] obtained a constant-factor approximation for planar graphs by using a more refined decomposition that took advantage of the structure of planar graphs. For graphs of treewidth k, [6] used the well-linked decomposition framework to obtain an $O(k \log k \log n)$-approximation and integrality gap. Our work here shows that one can bypass the well-linked decomposition framework for bounded treewidth graphs, and more generally for k-sums over families of graphs. The key high-level idea is to effectively reduce the (tree)width of one side of a sparse cut if the terminals cannot route to a small set of nodes. Making this work requires a somewhat nuanced induction hypothesis. For bounded-genus graphs, we adapt the well-linked decomposition to effectively reduce the problem to the planar graph case.

There are two streams of questions comparing minimum cuts to maximum flows in graphs. First, the *flow-cut gap* measures the gap between a sparsest cut and a maximum concurrent flow of an instance. The second measures the *throughput-gap* by comparing the maximum throughput flow and the minimum multicut. These gap results have been of fundamental importance in algorithms starting with the seminal work of Leighton and Rao [16]. It is known that the gaps in general undirected graphs are $\Theta(\log n)$; see [20] for a survey. It is also conjectured [12] (the GNRS Conjecture) that the flow-cut gap is $O(1)$ for minor-closed families. This conjecture is very much open and is not known even for planar graphs or treewidth 3 graphs; see [15] for relevant discussion and known results. In contrast, the work of Klein, Plotkin and Rao [13] showed that the throughput-gap is $O(1)$ in any proper minor-closed family of graphs (formally shown in [21]). The focus of these works is on fractional flows, in contrast to our focus on integral routings. Conjecture 1 is essentially asking about the integrality gap of throughput flows. Given the $O(1)$ throughput-gap [13], it can also be viewed as asking whether the gap between the maximum *integer* throughput flow with congestion 2 is within an $O(1)$ factor of the minimum multicut. Analogously for flow-cut gaps, [7] conjectured that the gap between the maximum integer concurrent flow and the sparsest cut is $O(1)$ in minor-free graphs.

Organization: Due to space constraints several technical ingredients needed to prove Theorem 1 and Theorem 2 are presented only in the full version of the article; a summary of these ingredients at a high-level is provided in Section 3 after some preliminaries. The proof of Theorem 3 and discussion of open problems also appear in the full version.

2 Preliminaries

Recall that an instance of MAX EDP consists of a graph G and demand pairs H. In general H can be a multiset, however it is convenient to assume that H is a matching on the nodes of G. Indeed we just have to attach the terminals to leaves created from new nodes. With this assumption we use X to denote the set of terminals (the endpoints of the demand pairs) and M the matching on X that corresponds to the demands. We call the triple (G, X, M) a matching instance of MAX EDP. Let $\bar{x}$ be a feasible solution to the LP relaxation (1). For each node $v \in X$, we also use $x(v)$ to denote the value $\sum_{P \in \mathcal{P}_h} x_P$ where v is an endpoint of the demand h; this is called the *marginal value* of v. We assume that all capacities c_e are 1; this does not affect the integrality gap analysis. Moreover, as argued previously (cf. [2]), at a loss of a factor of 2 in the approximation ratio, the assumption can be made for polynomial-time algorithms that are based on rounding a solution to the flow relaxation.

2.1 k-Sums and the Structure Theorem of Robertson and Seymour

Let G_1 and G_2 be two graphs, and C_i a clique of size k in G_i. The graph G obtained by identifying the nodes of C_1 one-to-one with those of C_2, and then removing some of the edges between nodes of $C_1 = C_2$, is called a *k-sum* of G_1 and G_2. For a class of graphs $\mathcal{G}$, we define the class $\mathcal{G}_k$ of the graphs *obtained from $\mathcal{G}$ by k-sums*, to be the smallest class of graphs such that: (i) $\mathcal{G}$ is included in $\mathcal{G}_k$, and (ii) if G is a k-sum of $G_1 \in \mathcal{G}$ and $G_2 \in \mathcal{G}_k$, then $G \in \mathcal{G}_k$.

Fix a class of graphs $\mathcal{G}$. A tree $\mathcal{T}$ is a *tree decomposition* over $\mathcal{G}$ for a graph $G = (V, E)$, if each node A in $\mathcal{T}$ is associated to a subset of nodes $X_A \subseteq V$, called a *bag*, and the following properties hold:

- (i) for each $v \in V(G)$, the set of nodes of $\mathcal{T}$ whose bags contain v, form a non-empty sub-tree of $\mathcal{T}$,
- (ii) for each edge $uv \in E(G)$, there is a bag with both u and v in it,
- (iii) for any bag X, the graph obtained from $G[X]$ by adding cliques over $X \cap Y$, for every adjacent bag Y, is in $\mathcal{G}$. We denote this graph by $G[[X]]$.

When $\mathcal{G}$ is closed under taking minors, condition (iii) implies that $G[X]$ itself is in $\mathcal{G}$, as well as any graph obtained from $G[X]$ by adding edges in $X \cap Y$, for any adjacent bag Y. Throughout we assume that $\mathcal{G}$ is minor-closed. We sometimes identify the nodes of $\mathcal{T}$ with their respective bags. We also denote by $V(\mathcal{T})$, the union of all bags, and so $V(\mathcal{T}) \subseteq V(G)$.

A set of nodes $X \cap Y$, for X and Y adjacent bags, is called a *separator*. When the tree decomposition $\mathcal{T}$ is minimal (with respect to the number of bags), the separators are disconnecting node sets of G. Thus each edge e of $\mathcal{T}$ identifies a separator, denoted by V_e. For convenience, we usually work with rooted tree decompositions, where an arbitrary node is chosen to be the root. Then, for bag X and its parent Y, we denote by S_X the separator $X \cap Y$.

The *width* of a tree decomposition $\mathcal{T}$ is the maximum cardinality of a separator of $\mathcal{T}$. The *width* of a graph (relative to a graph class $\mathcal{G}$) is the smallest width of a tree

decomposition for that graph. A graph of width k can thus be obtained by k-sums of graphs from $\mathcal{G}$. As a special case, the *treewidth* of a graph G is the smallest k such that G admits a decomposition of width k relative to the class of all graphs with at most $k+1$ nodes.

Let $\mathcal{T}$ be a tree decomposition of a graph G, rooted at a node R. For any edge e of the tree decomposition, let $\overline{\mathcal{T}_e}$ and $\mathcal{T}_e$ be the subtrees obtained from $\mathcal{T}$ by removing e, with $R \in \overline{\mathcal{T}_e}$. We denote by G_e the graph obtained from the induced subgraph of G on node set $V(\mathcal{T}_e)$, and then removing all edges in $V(\overline{\mathcal{T}_e}) \times V(\overline{\mathcal{T}_e})$. Note that $\mathcal{T}_e$ is a tree decomposition of G_e.

We recall informally the graph structure theorem proved by Robertson and Seymour. For $k \in \mathbb{N}$, let $\mathcal{L}^k$ be the graphs obtained in the following way.

- we start from a graph G embeddable on a surface of genus k,
- then we add *vortices* of width k to at most k faces of G,
- then we add at most k *apex* nodes. That is, each of these nodes can be adjacent to an arbitrary subset of nodes.

Then, we consider the closure $\mathcal{L}^k_k$ of $\mathcal{L}^k$ by k-sums. For a graph H, we denote by $\mathcal{K}_H$ the graphs that do not contain an H-minor.

Theorem 4 (Robertson and Seymour [18]). *For any graph H, there is an integer $k > 0$ such that $\mathcal{K}_H \subseteq \mathcal{L}^k_k$.*

In order to prove Conjecture 1, one should be able to use the preceding decomposition theorem, proving that the CFCC property holds for bounded genus graph and is preserved by adding a constant number of vortices and apex nodes, and by taking k-sums. Apex nodes are easy to deal with. This paper provides a proof for bounded genus graphs and for k-sums. This leaves only the cases of vortices as the bottleneck in proving the conjecture.

3 Technical Ingredients

We rely on several technical tools and ingredients that are either explicitly or implicitly used in recent work on MAX EDP. Due to space constraints the full development of these tools is available only in the full version of the article. Here we give a high-level description.

Moving Terminals: The idea here (also leveraged in previous work, cf. [5,6]) is to reduce a MAX EDP instance to a simpler/easier instance by moving the terminals (by sending flow) to a specific set of new locations (nodes). The two instances are equivalent up to an additional constant congestion and constant-factor approximation.

Sparsifiers: Given a graph $G = (V, E)$ and $S \subset V$ a sparsifier is a graph H only on the node set S that acts as a proxy for routing between nodes in S in the original graph G. We are interested in integer sparsifiers of certain type when $|S|$ is small, a constant in our setting. We say that $H = (S, E_H)$ is a (σ, ρ)-*sparsifier* for S in G if the following properties are true: (i) any feasible (fractional) multicommodity flow in

G with the endpoints in S is (fractionally) routable in H with congestion at most σ, and (ii) any integer multicommodity flow in H is integrally routable in G with congestion ρ. A simple argument shows that for any G any $S \subset V$, the existence of a $(|S|^2, 2)$-sparsifier.

Routing through a Small Set of Nodes: The idea here is that if all the flow for a given instance intersects a small set of nodes, say p, then one can in fact obtain an $\Omega(1/p)$-approximation for MAX EDP. This follows the ideas from [4] where the special case of $p = 1$ was exploited.

4 MAX EDP in k-Sums over a Family $\mathcal{G}$

The goal of this section is to prove Theorem 2. Throughout, we assume $\mathcal{G}$ is a minor closed family, and we wish to prove bounds for the family $\mathcal{G}_k$ obtained by k-sums. In particular, we assume that every subgraph on k nodes is included in $\mathcal{G}$.

Let $\mathcal{A}$ be an algorithm/oracle that has the following property: given a MAX EDP instance on a graph $G \in \mathcal{G}$ it integrally routes h pairs with congestion β where h is at least a $1/\alpha$ fraction of the value of an optimum fractional solution to that instance. We call $\mathcal{A}$ an (α, β)-*oracle*. We describe an algorithm using $\mathcal{A}$ to approximate MAX EDP on $\mathcal{G}_k$. The proof is via induction on the width of a decomposition and the number of nodes. One basic step is to take a sparse cut S, lose all the flow crossing that cut, and recurse on both sides. We need to make our recursion on treewidth on the side S to which we charge the flow lost by cutting the graph. On the other side, $V \setminus S$, we simply recurse on the number of nodes. The main difficulty is to show how the treewidth is decreased on the S side. The trick is a trade-off between the main treewidth parameter and some connectivity properties in parts of the graph with higher treewidth. To drive this we need a more refined induction hypothesis rather than basing it only on the width of a tree decomposition.

Given $k \leq p$, a *p-degenerate k-tree decomposition* over $\mathcal{G}$ of a connected graph G is a rooted tree decomposition $\mathcal{T}$ where some leaves (nodes of degree 1) of the tree are labelled *degenerate* and:

- for every node X of $\mathcal{T}$, either $G[[X]] \in \mathcal{G}$ or X is a degenerate leaf (in which case $G[[X]]$ may be arbitrary),
- the separator corresponding to any edge $uv \in \mathcal{T}$ is of size at most k, unless it is incident to a degenerate leaf, in which case it may be up to size p.

A pendant leaf is not necessarily degenerate but if it is not, then it corresponds to a graph in $\mathcal{G}$. We use (k, p)-tree decomposition as a shorthand notation. We call a multiflow in such a graph G *flush* with the decomposition if every flow path that terminates at some node in a degenerate leaf L, also intersects S_L (we recall that S_L is the separator $V(L) \cap V(X)$ where X is the parent node of L).

We may think of graphs with such flush / degenerate decompositions as having an *effective treewidth* of size k. In effect, we can ignore that some separators can be larger, because we have the separate property of flushness which we can leverage via our technical tools.

The next theorem describes the algorithm for converting an LP solution on some graph in $\mathcal{G}_p$, into an integral routing. As we process this solution, the tree decomposition becomes degenerate with the value of p fixed, k is gradually reduced to 0. We will assume that $p > 0$ for if $k = p = 0$ all the separators are empty and the given connected graph G is in $\mathcal{G}$ and we can simply use $\mathcal{A}$.

Theorem 5. *Let $\mathcal{A}$ be an (α, β) oracle for* MAX EDP *in a minor-closed family $\mathcal{G}$. Let G be graph with a (k, p)-tree decomposition $\mathcal{T}$ and suppose that G, H is an instance of* MAX EDP *with a fractional solution $\bar{x}$ that is flush with $\mathcal{T}$. Then there is an algorithm with oracle $\mathcal{A}$, which computes an integral multicommodity flow with congestion $\beta + 3$ and value $\gamma = \frac{\sum_i x_i}{216 \cdot \alpha p^2 3^k}$. Moreover, this algorithm can be used to obtain the following:*

1. *an LP-based approximation algorithm with ratio $O(\alpha p^2 3^p)$ and congestion $\beta + 3$ for* MAX EDP *in $\mathcal{G}_p$*
2. *an algorithm with approximation ratio $O((p+1)3^p)$ and congestion 1 for the class of graphs of treewidth p.*

The proof of the preceding theorem is somewhat long and technical and occupies the rest of the section. To help the exposition we break it up into several components. The proof proceeds by induction on k and the number of nodes. The base case with $k = 0$ is non-trivial and we treat it first.

The Base Case: We can assume without loss of generality that G is connected. Throughout we assume a fixed $p \geq 1$ and consider a decomposition together with a flush fractional routing $\bar{x}$ as described. Let $x : V \to \mathbb{R}$ be the

marginal values of $\bar{x}$. Again we use x_i to denote the common value $x(s_i) = x(t_i)$. Hence $|\bar{x}| = \sum_i x_i = \frac{1}{2} \sum_{v \in V(G)} x(v)$ where we use the notation $|\bar{x}|$ to denote the value of the flow $\bar{x}$.

Now assume that $k = 0$ and $p \geq 1$. We may assume that $\mathcal{T}$ has more than one node otherwise $G \in \mathcal{G}$ and we can apply $\mathcal{A}$. Since G is connected, each separator corresponding to an edge of $\mathcal{T}$ must be non-trivial. Since $k = 0$, each edge of $\mathcal{T}$ must be incident to a degenerate leaf. It follows that $\mathcal{T}$ is a single edge between two degenerate leaves or a star whose leaves are all degenerate. If $\mathcal{T}$ is a single edge between two degenerate leaves, all flow paths intersect the separator of size p associated with the edge; hence using the tool to route through a small set of nodes, there is an integral routing of value at least $(\sum_i x_i)/(12p)$. We therefore restrict our attention to the case when $\mathcal{T}$ is a star with ℓ leaves. Let $G^* = G[X]$ where X is the bag at the center/root; we observe that $G^* \in \mathcal{G}$. Let X_i be the bag at the i'th leaf. We let G_i denote the graph obtained from $G[X_i]$ after removing the edges between the separator nodes $S_i = X \cap X_i$.

The base case of theorem assertion (2) on treewidth p graphs holds as follows. By flushness, all flow paths intersect G^*, which has at most $p + 1$ nodes. Hence there is some node which is receiving at least $(\sum_i x_i)/(p + 1)$ of this flow. Using our technical tool for routing through a small set of nodes, there is a (congestion 1) integral routing of value at least $(\sum_i x_i)/(12(p + 1))$.

Now consider the general case with a minor-closed class $\mathcal{G}$ and $\mathcal{T}$ a star whose center is $G^* \in \mathcal{G}$. We proceed in the following steps:

1. Using the flushness property move the terminals in each G_i to the separator S_i that is contained in G^*.
2. In G replace each G_i by a $(p^2, 2)$ sparsifier on S_i to obtain a new graph G'. Via the sparsifier property, scale the flow in G down by a factor of p^2 to obtain a corresponding feasible flow in G'.
3. Apply the algorithm $\mathcal{A}$ on the new instance in G'.
4. Transfer the routing in G' to a routing in G with additive congestion $+1$ via the sparsifier property.
5. Convert the routing in G into a solution for our original instance before the terminals were moved (incur an additional $+2$ additive congestion).

We describe the steps in more detail. We observe that the graphs G_i are edge-disjoint. The first step is a simple application of the technique for moving terminals, where for each i, we move any terminals in $G_i - S_i$ to the separator S_i via clustering. This is possible because of the flushness assumption; if P is a flow path with an endpoint in $G_i - S_i$ then that path intersects S_i. This incurs a factor 5 loss in the value of the new flow we work with (and it incurs an additive 2 congestion when we convert back to an integral solution for our original instance). To avoid notational overload we let f be the flow for the new instance which is at least $\frac{1}{5}$th of the original flow.

After the preceding step no node in $G_i - S_i$ is the end point of an active flow path. In step (2), we can simultaneously replace each G_i by a $(p^2, 2)$-sparsifier F_i on S_i. Call the new instance G' and note that since we only added edges to the separators S_i, G' is a subgraph of $G[[X]]$ and hence $G' \in \mathcal{G}$. At this step, we also need to convert our flows in G_i's to be flows in G'. The sparsifier guarantees that any multicommodity flow on S_i that is feasible in G_i can be routed in F_i with congestion p^2. Hence, scaling the flow down by p^2 guarantees its feasibility in G'.

We now work with the new flow $\bar{x}'$ in the graph G' and apply $\mathcal{A}$ to obtain a routing of size $|\bar{x}'| / \alpha \geq |\bar{x}| / (5p^2\alpha)$ with congestion β. We must now convert this integral routing to one in G. Again, for each i, there is an embedded integral routing in F_i which will be re-routed in G_i. We incur an additive 1 congestion for this. Finally the way we moved terminals in the first part allows us to route the original pairs in G before they were moved to the separators, incurring an additive congestion of 2, the technical details are given in the full version.

Thus the total number of pairs routed is at least $|\bar{x}| / (5\alpha p^2)$ and the overall congestion of the routing is $\beta + 3$. This proves the base case when $k = 0$.

The Induction Step: Henceforth, we assume that $p \geq k > 0$ and that $\mathcal{T}$ contains at least one edge e with the associated separator V_e (the intersection of the bags at the two end points of e) of size equal to k; otherwise $\mathcal{T}$ is a star with degenerate leaves as in the base case, or we may use $k - 1$. We consider an easy setting when there is a flow g that simultaneously routes $x(v)/6$ amount from each vertex v to the set V_e. (Note that checking the existence of the desired flow to V_e can be done by a simple maximum-flow computation.) We then obtain an integral (congestion 1) flow of size $(\sum_i x_i)/(216k)$

using the technical tool for routing through a small set of nodes, aplied to V_e. This is sufficient to establish the induction step for k.

Assume now that there is no such flow g. Then there is a cut $U \subset V \setminus V_e$ with $c(U) := c(\delta(U)) < \frac{1}{6}x(U)$. We may assume that U is minimal and central ($G[U]$ and $G[V \setminus U]$ are connected). Such a cut can be recovered from the maximum flow computation. We now work with a reduced flow $\bar{x}'$ obtained from $\bar{x}$ by eliminating any flow path that intersects $\delta(U)$. We also let x' be the marginals for $\bar{x}'$. Obviously we have

$$|\bar{x}| - |\bar{x}'| \leq c(U) < \frac{x(U)}{6}.$$

Let $f_U, f_{\bar{U}}$ be the flow vectors obtained from $\bar{x}'$, where f_U only uses the flow paths contained in U, and $f_{\bar{U}}$ uses the flow paths contained in $V \setminus U$. The idea is that we recurse on $G[U]$ and $G[V \setminus U]$. We modify the instance on $G[U]$ to ensure that it has a $(k-1,p)$-tree decomposition and charge the lost flow to this side. The recursion on $G[V \setminus U]$ is based on reducing the number of nodes, the width is not reduced. Reducing the width on the U side and ensuring the flushness property is not immediate; it requires us to modifying f_U and in the process we may lose further flow. We explain this process before analyzing the number of pairs routed by the algorithm.

We note that $G[U]$ and $G[V \setminus U]$ easily admit (k,p)-tree decompositions, by intersecting the nodes of $\mathcal{T}$ with U and $V \setminus U$ respectively, and removing the empty nodes; recall $G[U]$ and $G[V \setminus U]$ are connected. Denote these by $\mathcal{T}_U$ and $\mathcal{T}_{\bar{U}}$ respectively. Degenerate leaves of $\mathcal{T}_U$ and $\mathcal{T}_{\bar{U}}$ are the same as the degenerate leaves of $\mathcal{T}$. Some of these are "split" by the cut, otherwise they are simply assigned to either $\mathcal{T}_U$ or $\mathcal{T}_{\bar{U}}$. If split, the two "halves" go to appropriate sides of the decomposition. The flows $f_U, f_{\bar{U}}$ will be flush with each such degenerate leaf (if the leaf is split, any flow path that crosses the cut is removed).

We proceed in $\mathcal{T}_{\bar{U}}$ by induction on the number of nodes. However, since we charge the lost flow to the cut (i.e., to $\mathcal{T}_U$) we modify $\mathcal{T}_U$ to obtain a $(k-1,p)$-tree decomposition. We state a lemma that accomplishes this.

Lemma 1. *For the residual instance on $G[U]$ with flow f_U we can either route $\frac{1}{216k} \cdot |f_U|/2$ pairs integrally or find a $(k-1,p)$-tree decomposition $\mathcal{T}'_U$ and a reduced flow vector f'_U that is flush with $\mathcal{T}'_U$ with $|f'_U| \geq |f_U|/2$.*

We postpone the proof of the above lemma and proceed to finish the recursive analysis. We apply the induction hypothesis for k on $\mathcal{T}_{\bar{U}}$ with the number of nodes reduced; hence the algorithm routes at least $\frac{|f_U|}{\alpha p^2 216 \cdot 3^k}$ pairs in $G[V \setminus U]$ with congestion at most $\beta + 3$. For $G[U]$ we consider two cases based on the preceding lemma. In the first case the algorithm directly routes $\frac{1}{216k} \cdot |f_U|/2$ pairs integrally in $G[U]$. In the second case we recurse on $G[U]$ with the flow f'_U that is flush with respect to the $(k-1,p)$-tree decomposition $\mathcal{T}'_U$; by the induction hypothesis the algorithm routes at least $\frac{|f'_U|}{\alpha p^2 216 \cdot 3^{k-1}}$ pairs with congestion $\beta + 3$. Since the number of pairs routed in this second case is less than in the first case, we may focus on it, as we now show that the total number of pairs routed in $G[U]$ and $G[V \setminus U]$ satisfies the induction hypothesis for k. We first observe that $|f| \leq |f_U| + |f_{\bar{U}}| + c(U)$. Moreover, $c(U) < x(U)/6$, and since at least $x(U)/2$

flow originated in U, and we lost at most $c(U)$ of this flow, we have $|f_U| \geq 2c(U)$. The total number of pairs routed is at least

$$\frac{|f'_U|}{216\alpha p^2 \cdot 3^{k-1}} + \frac{|f_{\bar{U}}|}{216\alpha p^2 \cdot 3^k} \geq \frac{3\,|f'_U| + |f_{\bar{U}}|}{216\alpha p^2 \cdot 3^k} \geq \frac{(3/2)\,|f_U| + |f_{\bar{U}}|}{216\alpha p^2 \cdot 3^k}$$
$$\geq \frac{|f_U| + |f_{\bar{U}}| + c(U)}{216\alpha p^2 \cdot 3^k} \geq \frac{|\bar{x}|}{216\alpha p^2 \cdot 3^k}$$

which establishes the induction step for k. A very similar analysis shows the slightly stronger bound for treewidth p graphs — we simply have to use the stronger induction hypothesis in the preceding calculations and observe that the base case analysis also proves the desired stronger hypothesis.

Proof of Lemma 1: Recall that $\mathcal{T}_U$ is a (k,p)-degenerate decomposition for $G[U]$ and that f_U is flush with respect to $\mathcal{T}_U$. However, we wish to find a $(k-1,p)$-degenerate decomposition. Recall that U is disjoint from V_e, the separator of size k associated with an edge e of $\mathcal{T}$. We can assume that $\mathcal{T}$ is rooted and without loss of generality that $\mathcal{T}_U$ is a sub-tree of $\mathcal{T}_e$. The reason that $\mathcal{T}_U$ may not be a $(k-1,p)$-tree decomposition is that it may contain a edge e' not incident to a degenerate leaf such that $|V_{e'}| = k$. $V_{e'}$ was then also a separator of G since $\mathcal{T}$ is a (k,p)-tree decomposition. By centrality of U, every node that $V_{e'}$ separates from V_e must then be in U; therefore, letting $U' = V(\mathcal{T}_{e'})$ (the union of all bags contained in $\mathcal{T}_{e'}$), we have $U' \subset U$. We claim that we can route $x(v)/6$ from each $v \in U'$ to $V_{e'}$ in $G' = G_{e'}$ (this is the graph induced by $G[U']$ but edges between the separator nodes $V_{e'}$ removed); this follows from the minimality of U since a cutset induced by $W \subset U' \setminus V_{e'}$ is also a cutset of G. We define a $(k-1,p)$-tree decomposition $\mathcal{T}'_U$, by contracting every *maximal* subtree of $\mathcal{T}_U$ rooted at such a separator of size k. Each such subtree identifies a new degenerate leaf in $\mathcal{T}'_U$. However, the flow f_U may not be flush with $\mathcal{T}'_U$ due to the creation of new degenerate leaves. We try to amend this by dropping flow paths with end points in a new degenerate leaf L that does not intersect the separator S_L. Let f'_U be the residual flow; observe that by definition f'_U is flush with respect to $\mathcal{T}'_U$. Two cases arise.

- If $|f'_U| \geq |f_U|/2$ then we have the desired degenerate $(k-1,p)$-decomposition of $G[U]$.
- Else at least $|f_U|/2$ of the flow is being routed completely within the new degenerate leaves L. Moreover, these graphs and flows are edge-disjoint. Note also in such an L, we have that the terminals involved can simultaneously route $x(v)/6$ each to S_L. We can thus obtain a constant fraction of the profit as $|S_L|$ is bounded (using the technical tool to route through a small set of nodes), separately to every new leaf. In particular, we route at least $\frac{1}{216k} \cdot |f_U|/2$ of the pairs. In this case, we no longer need to recurse on $\mathcal{T}_U$.

This finishes the proof of the lemma.

Finally, the main inductive claim implies the claimed algorithmic results since we can start with a proper p-decomposition $\mathcal{T}$ of $G \in \mathcal{G}_p$ (viewed as a (p,p)-tree decomposition with no degenerate leaves) and an arbitrary multiflow on its support (since there are no degenerate leaves the flushness is satisfied) in order to begin our induction. This finishes the proof. □

References

1. Andrews, M., Chuzhoy, J., Guruswami, V., Khanna, S., Talwar, K., Zhang, L.: Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. Combinatorica 30(5), 485–520 (2010)
2. Chekuri, C., Khanna, S., Shepherd, F.B.: Edge-disjoint paths in planar graphs. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 71–80 (2004)
3. Chekuri, C., Khanna, S., Shepherd, F.B.: Multicommodity flow, well-linked terminals, and routing problems. In: Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, pp. 183–192. ACM (2005)
4. Chekuri, C., Khanna, S., Shepherd, F.B.: An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. Theory of Computing 2(7), 137–146 (2006)
5. Chekuri, C., Khanna, S., Shepherd, F.B.: Edge-disjoint paths in planar graphs with constant congestion. SIAM Journal on Computing 39, 281–301 (2009)
6. Chekuri, C., Khanna, S., Shepherd, F.B.: A note on multiflows and treewidth. Algorithmica 54(3), 400–412 (2009)
7. Chekuri, C., Shepherd, F.B., Weibel, C.: Flow-cut gaps for integer and fractional multiflows. Journal of Combinatorial Theory, Series B 103(2), 248–273 (2013)
8. Chuzhoy, J.: Routing in undirected graphs with constant congestion. In: Proceedings of the 44th Symposium on Theory of Computing, pp. 855–874. ACM (2012)
9. Chuzhoy, J., Li, S.: A polylogarithimic approximation algorithm for edge-disjoint paths with congestion 2. In: Proc. of IEEE FOCS (2012)
10. Fortune, S., Hopcroft, J., Wyllie, J.: The directed subgraph homeomorphism problem. Theoretical Computer Science 10(2), 111–121 (1980)
11. Garg, N., Vazirani, V.V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica 18(1), 3–20 (1997)
12. Gupta, A., Newman, I., Rabinovich, Y., Sinclair, A.: Cuts, trees and ℓ_1-embeddings of graphs. Combinatorica 24(2), 233–269 (2004)
13. Klein, P., Plotkin, S.A., Rao, S.: Excluded minors, network decomposition, and multicommodity flow. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993, pp. 682–690. ACM, New York (1993)
14. Kleinberg, J., Tardos, E.: Approximations for the disjoint paths problem in high-diameter planar networks. Journal of Computers and System Sciences 57(1), 61–73 (1998)
15. Lee, J.R., Sidiropoulos, A.: Genus and the geometry of the cut graph. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 193–201. Society for Industrial and Applied Mathematics (2010)
16. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal of the ACM 46(6), 787–832 (1999)
17. Robertson, N., Seymour, P.D.: Graph minors: a survey. Surveys in Combinatorics 103, 153–171 (1985)
18. Robertson, N., Seymour, P.D.: Graph minors XVI. Excluding a non-planar graph. Journal of Combinatorial Theory, Series B 89(1), 43–76 (2003)
19. Seguin-Charbonneau, L., Shepherd, F.B.: Maximum edge-disjoint paths in planar graphs with congestion 2. In: Proc. of IEEE FOCS, pp. 200–209 (2011)
20. Shmoys, D.: Cut problems and their application to divide-and-conquer. In: Approximation Algorithms for NP-Hard Problems. PWS series in computer science. PWS Pub. Co. (1997)
21. Tardos, É., Vazirani, V.V.: Improved bounds for the max-flow min-multicut ratio for planar and $K_{r,r}$-free graphs. Information Processing Letters 47(2), 77–80 (1993)

On Integrality Ratios for Asymmetric TSP in the Sherali-Adams Hierarchy

Joseph Cheriyan, Zhihan Gao, Konstantinos Georgiou, and Sahil Singla

University of Waterloo, Waterloo, Ontario N2L3G1, Canada
{jcheriyan,z9gao,k2georgiou,s2singla}@uwaterloo.ca

Abstract. We study the ATSP (Asymmetric Traveling Salesman Problem), and our focus is on negative results in the framework of the Sherali-Adams (SA) Lift and Project method.

Our main result pertains to the standard LP (linear programming) relaxation of ATSP, due to Dantzig, Fulkerson, and Johnson. For any fixed integer $t \geq 0$ and small ϵ, $0 < \epsilon \ll 1$, there exists a digraph G on $\nu = \nu(t, \epsilon) = O(t/\epsilon)$ vertices such that the integrality ratio for level t of the SA system starting with the standard LP on G is $\geq 1 + \frac{1-\epsilon}{2t+3} \approx \frac{4}{3}, \frac{6}{5}, \frac{8}{7}, \ldots$. Thus, in terms of the input size, the result holds for any $t = 0, 1, \ldots, \Theta(\nu)$ levels. Our key contribution is to identify a structural property of digraphs that allows us to construct fractional feasible solutions for any level t of the SA system starting from the standard LP. Our hard instances are simple and satisfy the structural property.

There is a further relaxation of the standard LP called the balanced LP, and our methods simplify considerably when the starting LP for the SA system is the balanced LP; in particular, the relevant structural property (of digraphs) simplifies such that it is satisfied by the digraphs given by the well-known construction of Charikar, Goemans and Karloff (CGK). Consequently, the CGK digraphs serve as hard instances, and we obtain an integrality ratio of $1 + \frac{1-\epsilon}{t+1}$ for any level t of the SA system, where $0 < \epsilon \ll 1$ and the number of vertices is $\nu(t, \epsilon) = O((t/\epsilon)^{(t/\epsilon)})$.

Also, our results for the standard LP extend to the PATH ATSP (find a min cost Hamiltonian dipath from a given source vertex to a given sink vertex).

1 Introduction

The Traveling Salesman Problem (TSP) is a celebrated problem in combinatorial optimization, with many connections to theory and practice. The problem is to find a minimum cost tour of a set of cities; the tour should visit each city exactly once. The most well known version of this probelm is the symmetric one (denoted TSP), where the distance (a.k.a. cost) from city i to city j is equal to the distance (cost) from city j to city i. The more general version is called the asymmetric TSP (denoted ATSP), and it does not have the symmetry restriction on the costs. Throughout, we assume that the costs satisfy the triangle inequalities, i.e., the costs are metric.

Linear programming (LP) relaxations play a central role in solving TSP or ATSP, both in practice and in the theoretical setting of approximation algorithms. Many LP

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 340–351, 2013.

relaxations are known for ATSP, see [17] for a recent survey. The most well-known relaxation (and the one that is most useful for theory and practice) is due to Dantzig, Fulkerson and Johnson; we call it the standard LP or the DFJ LP. It has a constraint for every nontrivial cut, and has an indegree and an outdegree constraint for each vertex; see Section 2.1. There is a further relaxation of the standard LP that is of interest; we call it the balanced LP (Bal LP); it is obtained from the standard LP by replacing the indegree and outdegree constraint at each vertex by a balance (equation) constraint. For metric costs, the optimal value of the standard LP is the same as the optimal value of the balanced LP; this is a well-known fact, see [17], [6, Footnote 3].

One key question in the area is the quality of the objective value computed by the standard LP. This is measured by the *integrality ratio* (a.k.a. integrality gap) of the relaxation, and is defined to be the supremum over all instances of the integrality ratio of the instance. The integrality ratio of an instance I is given by $opt(I)/dfj(I)$, where $opt(I)$ denotes the optimum (minimum cost of a tour) of I, and $dfj(I)$ denotes the optimal value of the standard LP relaxation of I; we assume that the optima exist and that $dfj(I) \neq 0$.[1]

For both TSP and ATSP, significant research efforts have been devoted over several decades to prove bounds on the integrality ratio of the standard LP. For TSP, methods based on Christofides' algorithm show that the integrality ratio is $\leq \frac{3}{2}$, whereas the best lower bound known on the integrality ratio is $\frac{4}{3}$. Closing this gap is a major open problem in the area. For ATSP, a recent result of Asadpour et al. [2] shows that the integrality ratio is $\leq O(\log n/\log\log n)$. On the other hand, Charikar, et al. [6] showed a lower bound of 2 on the integrality ratio, thereby refuting an earlier conjecture of Carr and Vempala [5] that the integrality ratio is $\leq \frac{4}{3}$.

Lampis [12] and Papadimitriou and Vempala [16], respectively, have proved hardness-of-approximation thresholds of $\frac{185}{184}$ for TSP and $\frac{117}{116}$ for ATSP; both results assume that $\mathbf{P}\neq\mathbf{NP}$.

Our goal is to prove lower bounds on the integrality ratios for ATSP for the tighter LP relaxations obtained by applying the Sherali-Adams Lift-and-Project method. Before stating our results, we present an overview of Lift-and-Project methods.

1.1 Hierarchies of Convex Relaxations

Over the past 25 years, several methods have been developed in order to obtain tightenings of relaxations in a systematic manner. Assume that each variable y_i is in the interval $[0, 1]$, i.e., the integral solutions are zero/one, and let n denote the number of variables in the original relaxation. The goal is to start with a simple relaxation, and then iteratively obtain a sequence of stronger/tighter relaxations such that the associated polytopes form a nested family that contains (and converges to) the integral hull[2].

These procedures, usually called *Lift-and-Project* hierarchies (or systems, or methods, or procedures), use polynomial reasonings together with the fact that in the 0/1

[1] Although the term integrality ratio is used in two different senses—one refers to an instance, the other to a relaxation (i.e., all instances)—the context will resolve the ambiguity.

[2] By the *integral hull* we mean the convex hull of the zero-one solutions that are feasible for the original relaxation.

domain, general polynomials can be reduced to multilinear polynomials (utilizing the identity $y_i^2 = y_i$), and then finally obtain a stronger relaxation by applying linearization (e.g., for subsets S of $\{1, \ldots, n\}$, the term $\prod_{i \in S} y_i$ is replaced by a variable y_S). In this overview, we gloss over the Project step. In particular, Sherali and Adams [18] devised the Sherali-Adams (SA) system, Lovász and Schrijver [15] devised the Lovász-Schrijver (LS) system, and Lasserre [13] devised the Lasserre system. See Laurent [14] for a survey of these systems; several other Lift-and-Project systems are known, see [9,3].

The index of each relaxation in the sequence of tightened relaxations is known as the *level* in the hierarchy; the level of the original relaxation is defined to be zero. For each of these hierarchies and for any $t = O(1)$, it is known that the relaxation at level t of the hierarchy can be solved to optimality in polynomial time, assuming that the original relaxation has a polynomial-time separation oracle, [19] (additional mild conditions may be needed for some hierarchies). In fact, the relaxation at level n is exact, i.e., the associated polytope is equal to the integral hull.

Over the last two decades, a number of important improvements on approximation guarantees have been achieved based on relaxations obtained from Lift-and-Project systems. See [9] for a recent survey of many such positive results.

Starting with the work of Arora et al. [1], substantial research efforts have been devoted to showing that tightened relaxations (for many levels) fail to reduce the integrality ratio for many combinatorial optimization problems (see [9] for a list of negative results). This task seems especially difficult for the SA system because it strengthens relaxations in a "global manner;" this enhances its algorithmic leverage for deriving positive results, but makes it more challenging to design instances with bad integrality ratios. Moreover, an integrality ratio for the SA system may be viewed as an unconditional inapproximability result for a restricted model of computation, whereas, hardness-of-approximation results are usually proved under some complexity assumptions, such as $\mathbf{P} \neq \mathbf{NP}$. The SA system is known to be more powerful than the LS system, while it is weaker than the Lasserre system; it is incomparable with the LS_+ system (the positive-semidefinite version of the Lovász-Schrijver system [15]).

A key paper by Fernández de la Vega and Kenyon-Mathieu [10] introduced a probabilistic interpretation of the SA system, and based on this, negative results (for the SA system) have been proved for a number of combinatorial problems; also see Charikar et al. [7], and Benabbas, et al. [4]. At the moment, it is not clear that methods based on [10] could give negative results for TSP and its variants, because the natural LP relaxations (of TSP and related problems) have "global constraints."

To the best of our knowledge, there are only two previous papers with negative results for Lift-and-Project methods applied to TSP and its variants. Cheung [8] proves an integrality ratio of $\frac{4}{3}$ for TSP, for $O(1)$ levels of LS_+. For ATSP, Watson [20] proves an integrality ratio of $\frac{3}{2}$ for level 1 of the Lovász-Schrijver hiearchy, starting from the balanced LP (in fact, both the hierarchies LS and SA give the same relaxation at level one).

We mention that Cheung's results [8] for TSP do not apply to ATSP, although at level 0, it is well known that any integrality ratio for the standard LP for TSP applies also to the standard LP for ATSP (this relationship does not hold for level 1 or higher).

1.2 Our Results and Their Significance

Our main contribution is a generic construction of fractional feasible solutions for any level t of the SA system starting from the standard LP relaxation of ATSP. We have a similar but considerably simpler construction when the starting LP for the SA system is the balanced LP. Our results on integrality ratios are direct corollaries.

We have the following results pertaining to the balanced LP relaxation of ATSP: We formulate a property of digraphs that we call the good decomposition property, and given any digraph with this property, we construct a vector y on the edges such that y is a fractional feasible solution to the level t tightening of the balanced LP by the Sherali-Adams system. Charikar, Goemans, and Karloff (CGK) [6] constructed a family of digraphs for which the standard LP has an integrality ratio of 2. We show that the digraphs in the CGK family have the good decomposition property, hence, we obtain an integrality ratio for level t of SA. In more detail, we prove that for any integer $t \geq 0$ and small enough $\epsilon > 0$, there is a digraph G from the CGK family on $\nu = \nu(t, \epsilon) = O((t/\epsilon)^{t/\epsilon})$ vertices such that the integrality ratio of the level-t tightening of Bal LP is $\geq 1 + \frac{1-\epsilon}{t+1} \approx 2, \frac{3}{2}, \frac{4}{3}, \frac{5}{4}, \ldots$ (where $t = 0$ identifies the original relaxation).

Our main result pertains to the standard LP relaxation of ATSP. Our key contribution is to identify a structural property of digraphs that allows us to construct fractional feasible solutions for the level t tightening of the standard LP by the Sherali-Adams system. This construction is much more difficult than the construction for the balanced LP. We present a simple family of digraphs that satisfy the structural property, and this immediately gives our results on integrality ratios. We prove that for any integer $t \geq 0$ and small enough $\epsilon > 0$, there are digraphs G on $\nu = \nu(t, \epsilon) = O(t/\epsilon)$ vertices such that the integrality ratio of the level t tightening of the standard LP on G is $\geq 1 + \frac{1-\epsilon}{2t+3} \approx \frac{4}{3}, \frac{6}{5}, \frac{8}{7}, \frac{10}{9}, \ldots$. The *rank* of a starting relaxation (or polytope) is defined to be the minimum number of tightenings required to find the integral hull (in the worst case). An immediate corollary is that the SA-rank of the standard LP relaxation on a digraph $G = (V, E)$ is linear in $|V|$, whereas, the rank in terms of the number of edges is $\Omega(\sqrt{|E|})$ (since the LP is on a complete digraph, namely, the metric completion).

Our results for the balanced LP and for the standard LP are incomparable, because the SA system starting from the standard LP is strictly stronger than the SA system starting from the balanced LP, although both the level zero LPs have the same optimal value, assuming metric costs. (In fact, there is an example on 5 vertices [11, Figure 4.4, p.60] such that the optimal values of the level 1 tightenings are different: $9\frac{1}{3}$ for the balanced LP and 10 for the standard LP.)

Finally, we extend our main results to the natural realaxation of PATH ATSP (min cost Hamiltonian dipath from a given source vertex to a given sink vertex), and we obtain integrality ratios $\geq 1 + \frac{2-\epsilon}{3t+4} \approx \frac{3}{2}, \frac{9}{7}, \frac{6}{5}, \frac{15}{13}, \ldots$ for the level-t SA tightenings. Our result on PATH ATSP is obtained by "reducing" from the result for ATSP; the idea behind this comes from an analogous result of Watson [20] in the symmetric setting; Watson gives a method for transforming Cheung's [8] result on the integrality ratio for TSP to obtain a lower bound on the integrality ratio for PATH TSP.

The solutions given by our constructions are *not* positive semidefinite; thus, they do not apply to the LS_+ hierarchy nor to the Lasserre hierarchy.

Let us assess our results, and place them in context. Observe that our integrality ratios fade out as the level of the SA tightening increases, and for $t \geq 55$ (roughly) our integrality ratio falls below the hardness threshold of $\frac{117}{116}$ of [16]. Thus, our integrality ratios cannot be optimal, and it is possible that an integrality ratio of 2 can be proved for $O(1)$ levels of the SA system.

On the other hand, our results are not restricted to $t = O(1)$. For example, parameterized with respect to the number of vertices in the input ν, our lower bound for the standard LP holds even for level $t = \Omega(\nu)$, and our lower bound for the balanced LP (which improves on our lower bound for the standard LP) holds even for level $t = \Omega(\log \nu / \log \log \nu)$, thus giving unconditional inapproximability results for these restricted algorithms, even allowing super-polynomial running time.

Moreover, our results (and the fact that they are not optimal) should be contrasted with the known integrality ratio results for the level zero standard LP, a topic that has been studied for decades.

2 Preliminaries

When discussing a digraph (directed graph), we use the terms dicycle (directed cycle), etc., but we use the term edge rather than directed edge or arc. For a digraph $G = (V, E)$ and $U \subseteq V$, $\delta^{out}(U)$ denotes $\{(v, w) \in E : v \in U, w \notin U\}$, the set of edges outgoing from U, and $\delta^{in}(U)$ denotes $\{(v, w) \in E : v \notin U, w \in U\}$. For $x \in \mathbb{R}^E$ and $S \subseteq E$, $x(S)$ denotes $\sum_{e \in S} x_e$.

By the *metric completion* of a digraph $G = (V, E)$ with nonnegative edge costs $c \in \mathbb{R}^E$, we mean the complete digraph G' on V with the edge costs c', where $c'(v, w)$ is taken to be the minimum cost (w.r.t. c) of a v, w dipath of G.

For a positive integer t and a ground set U, let $\mathcal{P}_t$ denote the family of subsets of U of size at most t, i.e., $\mathcal{P}_t = \{S : S \subseteq U, |S| \leq t\}$. We usually take the ground set to be the set of edges of a fixed digraph. Now, let G be a digraph, and let the ground set (for $\mathcal{P}_t$) be $E = E(G)$. Let E' be a subset of E. Let $\mathbf{1}^{E', t}$ denote a vector indexed by elements of $\mathcal{P}_t$ such that for any $S \in \mathcal{P}_t$, $\mathbf{1}_S^{E', t} = 1$ if $S \subseteq E'$, and $\mathbf{1}_S^{E', t} = 0$, otherwise.

We denote set difference by $-$, and we denote the addition (removal) of a single item e to (from) a set S by $S + e$ (respectively, $S - e$), rather than by $S \cup \{e\}$ (respectively, $S - \{e\}$).

2.1 LP Relaxations for Asymmetric TSP

Let $G = (V, E)$ be a digraph with nonnegative edge costs c. Let $\widehat{\mathsf{ATSP}}_{DFJ}(G)$ be the feasible region (polytope) of the following linear program that has a variable x_e for each edge e of G:

$$\begin{aligned}
\text{minimize} \quad & \sum_e c_e x_e \\
\text{subject to} \quad & x\left(\delta^{in}(S)\right) \geq 1, && \forall S: \emptyset \subset S \subset V \\
& x\left(\delta^{out}(S)\right) \geq 1, && \forall S: \emptyset \subset S \subset V \\
& x\left(\delta^{in}(\{v\})\right) = 1, \quad x\left(\delta^{out}(\{v\})\right) = 1, && \forall v \in V \\
& \mathbf{0} \leq \mathbf{x} \leq \mathbf{1}
\end{aligned}$$

In particular, when G is a complete graph with metric costs, the above linear program is the standard LP relaxation of ATSP (a.k.a. DFJ LP).

We obtain the balanced LP (Bal LP) from the standard LP by replacing the two constraints $x\left(\delta^{in}(\{v\})\right) = 1,\ x\left(\delta^{out}(\{v\})\right) = 1$ by the constraint $x\left(\delta^{in}(\{v\})\right) = x\left(\delta^{out}(\{v\})\right)$, for each vertex v. Let $\widehat{\mathsf{ATSP}}_{BAL}(G)$ be the feasible region (polytope) of Bal LP.

2.2 The Sherali-Adams System

Definition 1 (The Sherali-Adams system). *Consider a polytope $\widehat{P} \subseteq [0,1]^n$ over the variables $y_1, \ldots, y_n$, and its description by a system of linear constraints of the form $\sum_{i=1}^n a_i y_i \geq b$; note that the constraints $y_i \geq 0$ and $y_i \leq 1$ for all $i \in \{1, \ldots, n\}$ are included in the system. The level-t Sherali-Adams tightened relaxation $\mathcal{SA}^t(\widehat{P})$ of $\widehat{P}$, is an LP over the variables $\{y_S : S \subseteq \{1, 2, \ldots, n\},\ |S| \leq t+1\}$ (thus, $y \in \mathbb{R}^{\mathcal{P}_{t+1}}$ where $\mathcal{P}_{t+1}$ has ground set $\{1, 2, \ldots, n\}$); moreover, we have $y_\emptyset = 1$. For every constraint $\sum_{i=1}^n a_i y_i \geq b$ of $\widehat{P}$ and for every disjoint $S, Q \subseteq \{1, \ldots, n\}$ with $|S| + |Q| \leq t$, the following is a constraint of the level-t Sherali-Adams relaxation.*

$$\sum_{i=1}^n a_i \sum_{\emptyset \subseteq T \subseteq Q} (-1)^{|T|} y_{S \cup T \cup \{i\}} \geq b \sum_{\emptyset \subseteq T \subseteq Q} (-1)^{|T|} y_{S \cup T}. \tag{1}$$

There are a number of approaches for certifying that $y \in \mathcal{SA}^t(\widehat{P})$ for a given y. One popular approach is to give a probabilistic interpretation to the entries of y, satisfying certain conditions. We follow an alternative approach, that is standard, see [14], [19, Lemma 2.9], but has been rarely used in the context of integrality ratios.

First, we introduce some notation. Given a polytope $\widehat{P} \subseteq [0,1]^n$, consider the cone $P = \{y_\emptyset(1, y) : y_\emptyset \geq 0, y \in \widehat{P}\}$. It is not difficult to see that the SA system can be applied to the cone P, so that the projection in the n original variables can be obtained by projecting any $y \in \mathcal{SA}^t(P)$ with $y_\emptyset = 1$ on the n original variables. Note that $\mathcal{SA}^t(P)$ is a cone, hence, we may have $y \in \mathcal{SA}^t(P)$ with $y_\emptyset \neq 1$; but if $y_\emptyset \neq 0$, we can replace y by $\frac{1}{y_\emptyset} y$. Also, note that $\mathcal{SA}^t(\widehat{P}) = \{y : y_\emptyset = 1, y \in \mathcal{SA}^t(P)\}$ by Definition 1.

For a vector y indexed by subsets of $\{1, \ldots, n\}$ of size at most $t+1$, define a *shift operator* "$*$" as follows: for every $e \in \{1, \ldots, n\}$, let $e * y$ to be a vector indexed

by subsets of $\{1,\ldots,n\}$ of size at most t, such that $(e * y)_S := y_{S+e}$. We have the following folklore fact, [19, Lemma 2.9].

Fact 1. $y \in \mathcal{SA}^t(P)$ ***if and only if*** $e * y \in \mathcal{SA}^{t-1}(P)$, ***and*** $y - e * y \in \mathcal{SA}^{t-1}(P)$, $\forall e \in \{1,\ldots,n\}$.

Eliminating Variables to 0. In our discusssion of the standard LP and the balanced LP, it will be convenient to restrict the support to the edge set of a given digraph rather than the complete digraph. Thus, we assume that some of the variables are absent. Formally, this is equivalent to setting these variables in advance to zero. As long as the nonzero variables induce a feasible solution, we are justified in setting the other variables to zero. See the full version of the paper for further details.

3 SA Applied to the Balanced LP Relaxation of ATSP

3.1 Certifying a Feasible Solution

A strongly connected digraph $G = (V, E)$ is said to have a *good decomposition* with *witness set* $\mathcal{F}$ if the following hold

(i) E partitions into edge-disjoint dicycles $C_1, C_2, \ldots, C_N$, that is, there exist edge-disjoint dicycles $C_1, C_2, \ldots, C_N$ such that $E = \bigcup_{1 \le j \le N} E(C_j)$; let $\mathcal{N}$ denote the set of indices of these dicycles, thus $\mathcal{N} = \{1,\ldots,N\}$;
(ii) moreover, there exists a nonempty subset $\mathcal{F}$ of $\mathcal{N}$ such that for each $j \in \mathcal{F}$ the digraph $G - E(C_j)$ is strongly connected.

Let $\overline{\mathcal{F}}$ denote $\mathcal{N} - \mathcal{F}$. For an edge e, we use *index*(e) to denote the index j of the dicycle $C_j, j \in \mathcal{N}$ that contains e. In this section, by a dicycle C_i, C_j, etc., we mean one of the dicycles $C_1,\ldots,C_N$, and we identify a dicycle C_j with its edge set, $E(C_j)$. See Figure 1 for an illustration of a good decomposition of a digraph.

Informally speaking, our plan is as follows: given a digraph G that has a good decomposition with witness set $\mathcal{F}$, we construct a feasible solution to $\mathcal{SA}^t(\widehat{\mathsf{ATSP}}_{BAL}(G))$ by assigning the same fractional value to the edges of the dicycles C_j with $j \in \mathcal{F}$, while assigning the value 1 to the edges of the dicycles C_i with $i \in \overline{\mathcal{F}}$ (this is not completely correct; we will refine this plan). Let $\mathsf{ATSP}_{BAL}(G)$ be the associated cone of $\widehat{\mathsf{ATSP}}_{BAL}(G)$.

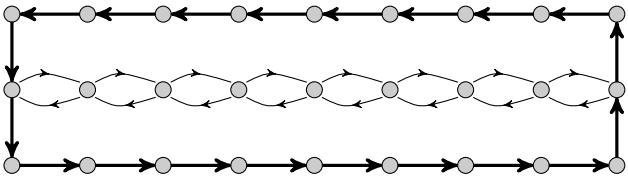


Fig. 1. A digraph G with a good decomposition given by the dicycle with thick edges, and the length 2 dicycles C_j formed by the anti-parallel pairs of thin edges; $G - E(C_j)$ is strongly connected for each dicycle C_j

Definition 2. *Let t be a nonnegative integer. For any set $S \subseteq E$ of size $\leq t+1$, and any subset $\mathcal{I}$ of $\mathcal{F}$, let $F^{\mathcal{I}}(S)$ denote the set of indices $j \in \mathcal{F} - \mathcal{I}$ such that $E(C_j) \cap S \neq \emptyset$; moreover, let $f^{\mathcal{I}}(S)$ denote $|F^{\mathcal{I}}(S)|$, namely, the number of dicycles C_j with indices in $\mathcal{F} - \mathcal{I}$ that intersect S.*

Definition 3. *For a nonnegative integer t and for any subset $\mathcal{I}$ of $\mathcal{F}$, let $y^{\mathcal{I},t}$ be a vector indexed by the elements of $\mathcal{P}_{t+1}$ and defined as follows:*

$$y_S^{\mathcal{I},t} = \frac{t+2-f^{\mathcal{I}}(S)}{t+2}, \quad \forall S \in \mathcal{P}_{t+1}$$

Theorem 2. *Let $G = (V, E)$ be a strongly connected digraph that has a good decomposition, and let $\mathcal{F}$ be the witness set. Then*

$$y^{\mathcal{I},t} \in \mathcal{SA}^t(\widehat{\mathsf{ATSP}}_{BAL}(G)), \quad \forall t \in \mathbb{Z}_+, \forall \mathcal{I} \subseteq \mathcal{F}.$$

Corollary 1. *We have*

$$y^{\emptyset,t} \in \mathcal{SA}^t(\widehat{\mathsf{ATSP}}_{BAL}(G)), \quad \forall t \in \mathbb{Z}_+,$$

and moreover, for each dicycle C_j, $j \in \mathcal{N}$, and each edge e of C_j we have

$$y_e^{\emptyset,t} = \begin{cases} \frac{t+1}{t+2}, & \text{if } j \in \mathcal{F} \\ 1, & \text{otherwise.} \end{cases} \tag{2}$$

When we apply the above theorem to bound the integrality ratio, then we fix $\mathcal{I} = \emptyset$, hence, Corollary 1 suffices. The more general setting of the theorem is essential for our induction proof; we give a high-level explanation in the proof-sketch below. Informally speaking, we assign the value 1 (rather than a fractional value) to the edges of the dicycles C_j with $j \in \mathcal{I} \subseteq \mathcal{F}$. For the sake of exposition, we call the dicycles C_j with $j \in \mathcal{F} - \mathcal{I}$ the *fractional dicycles*, and we call the remaining dicycles C_i (thus $i \in \mathcal{I} \cup \overline{\mathcal{F}}$) the *integral dicycles*.

Proof. To prove Theorem 2, we need to prove $y^{\mathcal{I},t} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$. We prove this by induction on t.

Note that $y_{\emptyset}^{\mathcal{I},t} = 1$ by Definition 3.

The induction basis is important, and it follows easily from the good decomposition property. In Lemma 3 (below) we show that $y^{\emptyset,0} \in \mathcal{SA}^0(\mathsf{ATSP}_{BAL}(G))$. It follows that $y^{\mathcal{I},0} \in \mathcal{SA}^0(\mathsf{ATSP}_{BAL}(G)),\ \forall \mathcal{I} \subseteq \mathcal{F}$ because $y^{\mathcal{I},0} \geq y^{\emptyset,0}$ (this follows from Definitions 2,3, since $F^{\mathcal{I}}(S) \subseteq F^{\emptyset}(S)$).

In the induction step, we assume that $y^{\mathcal{I},t} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$ for some integer $t \geq 0$ (the induction hypothesis), and we apply the recursive definition based on the shift operator, namely, $y^{\mathcal{I},t+1} \in \mathcal{SA}^{t+1}(\mathsf{ATSP}_{BAL}(G))$ iff for each $e \in E$

$$e * y^{\mathcal{I},t+1} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G)), \tag{3}$$

$$y^{\mathcal{I},t+1} - e * y^{\mathcal{I},t+1} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G)). \tag{4}$$

Lemma 1 (below) proves (3) and Lemma 2 (below) proves (4).

We prove that $e * y^{\mathcal{I},\,t+1}$ is in $\mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$ by showing that for some edges e, $e * y^{\mathcal{I},\,t+1}$ is a scalar multiple of $y^{\mathcal{I}',\,t}$, where $\mathcal{I}' \supsetneq \mathcal{I}$ (see Equation (5) in Lemma 1); thus, the induction hinges on the use of $\mathcal{I}$.

Before proving Lemma 1 and Lemma 2, we show that $y^{\mathcal{I},\,t+1}$, restricted to $\mathcal{P}_{t+1}$, can be written as a convex combination of $y^{\mathcal{I},\,t}$ and the integral feasible solution $\mathbf{1}^{E,\,t+1}$. This is used in the proof of Lemma 1; for some of the edges $e \in E$, we show that $e * y^{\mathcal{I},\,t+1} = y^{\mathcal{I},\,t+1}$ (see Equation (5)), and then we have to show that the latter is in $\mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$.

Fact 3. *Let t be a nonnegative integer and let $\mathcal{I}$ be a subset of $\mathcal{F}$. Then for any $S \in \mathcal{P}_{t+1}$ we have* $\quad y_S^{\mathcal{I},\,t+1} = \dfrac{t+2}{t+3} y_S^{\mathcal{I},\,t} + \dfrac{1}{t+3} \mathbf{1}_S^{E,\,t+1}$.

Proof. We have $S \subseteq E$, $|S| \leq t+1$, and we get $\mathbf{1}_S^{E,\,t+1} = 1$ from the definition. Thus,

$$y_S^{\mathcal{I},\,t+1} = \frac{t+3-f^{\mathcal{I}}(S)}{t+3} = \frac{t+2-f^{\mathcal{I}}(S)}{t+3} + \frac{1}{t+3} = \frac{t+2}{t+3} y_S^{\mathcal{I},\,t} + \frac{1}{t+3} \mathbf{1}_S^{E,\,t+1}.$$

Lemma 1. *Suppose that $y^{\mathcal{I},\,t} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$, for each $\mathcal{I} \subseteq \mathcal{F}$. Then* $e * y^{\mathcal{I},\,t+1} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G)), \quad \forall e \in E.$

Proof. For any $S \in \mathcal{P}_{t+1}$, the definition of the shift operator gives $(e * y^{\mathcal{I},\,t+1})_S = y_{S+e}^{\mathcal{I},\,t+1}$. Let $C(e)$ denote the dicycle containing edge e, and recall that *index*(e) denotes the index of $C(e)$.

We first show that

$$e * y_S^{\mathcal{I},\,t+1} = \begin{cases} \frac{t+2}{t+3} y_S^{\mathcal{I}+\mathit{index}(e),\,t} & \text{if } \mathit{index}(e) \in \mathcal{F} - \mathcal{I} \\ y_S^{\mathcal{I},\,t+1} & \text{otherwise} \end{cases} \tag{5}$$

If *index*$(e) \in \mathcal{I} \cup \overline{\mathcal{F}}$, that is, the dicycle $C(e)$ is not "fractional," then Definition 3 directly gives $y_{S+e}^{\mathcal{I},\,t+1} = y_S^{\mathcal{I},\,t+1}$. Otherwise, if *index*$(e) \in \mathcal{F} - \mathcal{I}$, then from Definition 3 we see that if $C(e) \cap S \neq \emptyset$, then $F^{\mathcal{I}}(S+e) = F^{\mathcal{I}}(S)$, and otherwise, $f^{\mathcal{I}}(S+e) = f^{\mathcal{I}}(S) + 1$. Hence,

$$(e * y^{\mathcal{I},\,t+1})_S = \begin{cases} \frac{t+3-f^{\mathcal{I}}(S)}{t+3} & \text{if } C(e) \cap S \neq \emptyset \\ \frac{t+2-f^{\mathcal{I}}(S)}{t+3} & \text{if } C(e) \cap S = \emptyset \end{cases} \tag{6}$$

$$= \frac{t+2}{t+3} y_S^{\mathcal{I}+\mathit{index}(e),\,t} \tag{7}$$

where in the last line we use Definition 3 to infer that $f^{\mathcal{I}+\mathit{index}(e)}(S) = f^{\mathcal{I}}(S) - 1$, if $C(e) \cap S \neq \emptyset$, and $f^{\mathcal{I}+\mathit{index}(e)}(S) = f^{\mathcal{I}}(S)$, otherwise.

Note that Fact 3 along with $y^{\mathcal{I},\,t} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$ implies that $y^{\mathcal{I},\,t+1}$, restricted to $\mathcal{P}_{t+1}$, is in $\mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$ because it can be written as a convex combination of $y^{\mathcal{I},\,t}$ and an integeral feasible solution $\mathbf{1}^{E,\,t+1}$. Equation (5) proves Lemma 1 because both $y^{\mathcal{I}+\mathit{index}(e),\,t}$ and $y^{\mathcal{I},\,t+1}$ (restricted to $\mathcal{P}_{t+1}$) are in $\mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$.

Lemma 2. *Suppose that $y^{\mathcal{I},\,t} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$, for each $\mathcal{I} \subseteq \mathcal{F}$. Then*

$$y^{\mathcal{I},\,t+1} - e * y^{\mathcal{I},\,t+1} \in \mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G)), \quad \forall e \in E.$$

Proof. Let $C(e)$ denote the dicycle containing edge e, and recall that *index*(e) denotes the index of $C(e)$. If *index*$(e) \in \mathcal{I} \cup \overline{\mathcal{F}}$, then we have $F^{\mathcal{I}}(S+e) = F^{\mathcal{I}}(S), \forall S \in \mathcal{P}_{t+1}$, hence, we have $y^{\mathcal{I},\,t+1} = e * y^{\mathcal{I},\,t+1}$, and the lemma follows.

Otherwise, we have *index*$(e) \in \mathcal{F} - \mathcal{I}$. Then, for any $S \in \mathcal{P}_{t+1}$, Equation (6) gives

$$(y^{\mathcal{I},\,t+1} - e * y^{\mathcal{I},\,t+1})_S = \begin{cases} 0 & \text{if } C(e) \cap S \neq \emptyset \\ \frac{1}{t+3} & \text{if } C(e) \cap S = \emptyset \end{cases} \tag{8}$$

$$= \frac{1}{t+3} \mathbf{1}_S^{E-C(e),\,t+1} \tag{9}$$

The good-decomposition property of G implies that $\mathbf{1}^{E-C(e),\,t+1}$ is a feasible integral solution of $\mathcal{SA}^t(\mathsf{ATSP}_{BAL}(G))$.

Lemma 3. *We have* $y^{\emptyset,\,0} \in \mathcal{SA}^0(\mathsf{ATSP}_{BAL}(G))$.

Proof. Observe that $y^{\emptyset,\,0}$ has $|E|+1$ elements, and $y^{\emptyset,\,0}_{\emptyset} = 1$ (by Definition 3); the other $|E|$ elements are indexed by the singleton sets of E. For notational convenience, let $y \in \mathbb{R}^E$ denote the restriction of $y^{\emptyset,\,0}$ to indices that are singleton sets; thus, $y_e = y^{\emptyset,\,0}_{\{e\}}, \forall e \in E$. By Definition 3, $y_e = 1/2$ if $e \in E(C_j)$ where $j \in \mathcal{F}$, and $y_e = 1$, otherwise. We claim that y is a feasible solution to $\widehat{\mathsf{ATSP}}_{BAL}(G)$.

y is clearly in $[0,1]^E$. Moreover, y satisifes the balance-constraint at each vertex because it assigns the same value (either $1/2$ or 1) to every edge in a dicycle $C_j, \forall j \in \mathcal{N}$.

To show feasibility of the cut-constraints, consider any cut $\emptyset \neq U \subset V$. Since $\mathbf{1}^E$ is a feasible solution, there exists an edge $e \in E$ crossing from U to $V-U$. If $e \in E(C_j), j \in \overline{\mathcal{F}}$, then we have $y_e = 1$, which implies $y(\delta^{out}(U)) = y(\delta^{in}(U)) \geq 1$ (from the balance-constraints at the vertices). Otherwise, we have $e \in E(C_j), j \in \mathcal{F}$. Applying the good-decomposition property of G, we see that there exists an edge $e' (\neq e) \in E - E(C_j)$ such that $e' \in \delta^{out}(U)$, i.e., $|\delta^{out}(U)| \geq 2$. Since $y_e \geq \frac{1}{2}$ for each $e \in E$, the cut-constraints $y(\delta^{in}(U)) = y(\delta^{out}(U)) \geq 1$ are satisfied.

The next result presents our first lower bound on the integrality ratio for the level t relaxation of the Sherali-Adams procedure starting with the balanced LP. The relevant instance is a simple digraph on $\Theta(t)$ vertices; see Figure 1. In the next subsection, we present better integrality ratios using the CGK construction, but the CGK digraph is not as simple and it has $\Theta(t^t)$ vertices.

Theorem 4. *Let t be a nonnegative integer, and let $\epsilon \in \mathbb{R}$ satisfy $0 < \epsilon \ll 1$. There exists a digraph on $\nu = \nu(t,\epsilon) = \Theta(t/\epsilon)$ vertices such that the integrality ratio for the level t tightening of the balanced LP (Bal LP) (by the Sherali-Adams system) is $\geq 1 + \frac{1-\epsilon}{2t+3}$.*

3.2 CGK (Charikar-Goemans-Karloff) Construction

We focus on the CGK [6] construction and show in Theorem 5 that the resulting digraph has a good decomposition. This theorem along with a lemma from [6] shows that the integrality ratio is $\geq 1 + \frac{1-\epsilon}{t+1}$ for the level t relaxation of the Sherali-Adams procedure starting with the balanced LP, for any given $0 < \epsilon \ll 1$, see Theorem 6.

The CGK construction gives a digraph G_k with edge costs for each $k = 0, 1, 2, \ldots$, and for any given $k \geq 2$, their hard instance L_k is obtained by a simple modification of G_k, see [6, Section 3].

Theorem 5. *For $k \geq 2$, L_k has a good decomposition with witness set $\mathcal{F}$ such that $\mathcal{F} = \mathcal{N}$, i.e. every cycle in the decomposition can be assigned a fractional value.*

Theorem 6. *Let t be a nonnegative integer, and let $\epsilon \in \mathbb{R}$ satisfy $0 < \epsilon \ll 1$. There exists a digraph on $\nu = \nu(t, \epsilon) = O((t/\epsilon)^{(t/\epsilon)})$ vertices such that the integrality ratio for the level t tightening of the balanced LP (Bal LP) (by the Sherali-Adams system) is $\geq 1 + \frac{1-\epsilon}{t+1}$.*

4 SA Applied to the Standard (DFJ LP) Relaxation of ATSP

We state the main result of this section. Further details can be found in the full version of the paper.

Theorem 7. *Let t be a nonnegative integer, and let $\epsilon \in \mathbb{R}$ satisfy $0 < \epsilon \ll 1$. There exists a digraph on $\nu = \nu(t, \epsilon) = \Theta(t/\epsilon)$ vertices such that the integrality ratio for the level t tightening of the standard LP (DFJ LP) (by the Sherali-Adams system) is $\geq 1 + \frac{1-\epsilon}{2t+3}$.*

5 Path ATSP

We state the main result of this section. See the full version of the paper for further details.

Let $G = (V, E)$ be a digraph with nonnegative edge cost c, and let p and q be two distinguished vertices. In the path ATSP, the goal is to compute a Hamiltonian (or spanning) dipath from p to q with minimum cost in the metric completion of G.

Theorem 8. *Let t be a nonnegative integer, and let $\epsilon \in \mathbb{R}$ satisfy $0 < \epsilon \ll 1$. There exists a digraph on $\nu = \nu(t, \epsilon) = \Theta(t/\epsilon)$ vertices such that the integrality ratio for the level t tightening of the standard LP (PATSP) (by the Sherali-Adams system) is $\geq 1 + \frac{2-\epsilon}{3t+4}$.*

Acknowledgements. We thank a number of colleagues for useful discussions. We are grateful to Sylvia Boyd and Paul Elliott-Magwood for help with ATSP integrality gaps, and to Levent Tuncel for sharing his knowledge of the area.

References

1. Arora, S., Bollobás, B., Lovász, L., Tourlakis, I.: Proving integrality gaps without knowing the linear program. Theory of Computing 2(1), 19–51 (2006)
2. Asadpour, A., Goemans, M.X., Madry, A., Gharan, S.O., Saberi, A.: An O(log n/log log n)-approximation algorithm for the Asymmetric Traveling Salesman Problem. In: Proc. ACM–SIAM SODA 2010, pp. 379–389. SIAM (2010)
3. Au, Y.H., Tunçel, L.: Complexity analyses of Bienstock-Zuckerberg and Lasserre relaxations on the matching and stable set polytopes. In: Günlük, O., Woeginger, G.J. (eds.) IPCO 2011. LNCS, vol. 6655, pp. 14–26. Springer, Heidelberg (2011)
4. Benabbas, S., Chan, S.O., Georgiou, K., Magen, A.: Tight gaps for vertex cover in the Sherali-Adams SDP hierarchy. In: Proc. FSTTCS 2011. LIPIcs, vol. 13, pp. 41–54 (2011)
5. Carr, R., Vempala, S.: On the Held-Karp relaxation for the asymmetric and symmetric traveling salesman problems. Math. Program. 100(3), 569–587 (2004)
6. Charikar, M., Goemans, M.X., Karloff, H.J.: On the integrality ratio for the Asymmetric Traveling Salesman Problem. Math. Oper. Res. 31(2), 245–252 (2006)
7. Charikar, M., Makarychev, K., Makarychev, Y.: Integrality gaps for Sherali-Adams relaxations. In: Proc. ACM STOC 2009, New York, NY, USA, pp. 283–292 (2009)
8. Cheung, K.K.H.: On Lovász–Schrijver lift-and-project procedures on the Dantzig–Fulkerson–Johnson relaxation of the TSP. SIAM Journal on Optimization 16(2), 380–399 (2005)
9. Chlamtáč, E., Tulsiani, M.: Convex relaxations and integrality gaps. In: Anjos, M.F., Lasserre, J.B. (eds.) Handbook on Semidefinite, Conic and Polynomial Optimization. International Series in Operations Research & Management Science, vol. 166, pp. 139–169. Springer, US (2012)
10. de la Vega, W.F., Kenyon-Mathieu, C.: Linear programming relaxations of maxcut. In: Proc. ACM–SIAM SODA 2007, pp. 53–61. ACM Press (2007)
11. Elliott-Magwood, P.: The integrality gap of the Asymmetric Travelling Salesman Problem. PhD thesis, Department of Mathematics and Statistics, University of Ottawa (2008)
12. Lampis, M.: Improved inapproximability for TSP. In: Gupta, A., Jansen, K., Rolim, J., Servedio, R. (eds.) APPROX/RANDOM 2012. LNCS, vol. 7408, pp. 243–253. Springer, Heidelberg (2012)
13. Lasserre, J.B.: An explicit equivalent positive semidefinite program for nonlinear 0-1 programs. SIAM Journal on Optimization 12(3), 756–769 (2002)
14. Laurent, M.: A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0-1 programming. Math. Oper. Res. 28(3), 470–496 (2003)
15. Lovász, L., Schrijver, A.: Cones of matrices and set-functions and 0-1 optimization. SIAM Journal on Optim. 1(2), 166–190 (1991)
16. Papadimitriou, C.H., Vempala, S.: On the approximability of the traveling salesman problem. Combinatorica 26(1), 101–120 (2006)
17. Roberti, R., Toth, P.: Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison. EURO Journal on Transportation and Logistics 1, 113–133 (2012)
18. Sherali, H.D., Adams, W.P.: A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. SIAM Journal on Discrete Mathematics 3(3), 411–430 (1990)
19. Tourlakis, I.: New lower bounds for Approximation Algorithms in the Lovasz-Schrijver Hierarchy. PhD thesis, Department of Computer Science, Princeton University (2006)
20. Watson, T.: Lift-and-project integrality gaps for the Traveling Salesperson Problem. Electronic Colloquium on Computational Complexity (ECCC) 18, 97 (2011)

Counting Matchings of Size k Is #W[1]-Hard

Radu Curticapean

Saarland University, Dept. of Computer Science
curticapean@cs.uni-saarland.de

Abstract. We prove #W[1]-hardness of the following parameterized counting problem: Given a simple undirected graph G and a parameter $k \in \mathbb{N}$, compute the number of matchings of size k in G.

It is known from [1] that, given an edge-weighted graph G, computing a particular weighted sum over the matchings in G is #W[1]-hard. In the present paper, we exhibit a reduction that does not require weights.

This solves an open problem from [5] and adds a natural parameterized counting problem to the scarce list of #W[1]-hard problems. Since the classical version of this problem is well-studied, we believe that our result facilitates future #W[1]-hardness proofs for other problems.

1 Introduction

Let $G = (V, E)$ be an undirected graph on n vertices. A matching M in G is a set of vertex-disjoint edges $M \subseteq E$. For $k \in \mathbb{N}$, a k-matching is a matching with $|M| = k$, and $(n/2)$-matchings are commonly referred to as perfect matchings.

Counting (Perfect) Matchings: Two natural counting problems on matchings are well-studied: The problem #PerfMatch of counting all *perfect* matchings in an input graph G, and the problem #Match of counting *all* matchings in G. The problem #PerfMatch already appeared along with the definition of the complexity class #P in [10] and was among the first problems to be proven #P-complete. In [11], the problem #Match was also proven #P-hard.

Subsequent work identified restricted graph classes on which the problems #PerfMatch and #Match are already #P-hard, as well as some tractable graph classes. For instance, #Match is already hard on planar 3-regular graphs [14], while #PerfMatch admits a polynomial-time algorithm on planar graphs [9]. This last result, and matchings in general, are also central to the new theory of holographic algorithms introduced in [12].

Parameterized Counting Complexity: In a relatively new approach to #Match, and other #P-hard problems in general, counting problems are considered as parameterized problems, see [4]. In such problems, inputs x come with an additional parameter k, and a parameterized counting problem is *fixed-parameter tractable (fpt)* in k if it can be solved in time $f(k)|x|^{O(1)}$ for a computable function f. The class #W[1] and the notion of #W[1]-hardness were both defined in [4], bridging classical counting complexity and parameterized complexity theory.

In parameterized counting problems on *graphs*, the parameter k typically either measures some notion of intricacy of the *input graph* or the intricacy of the

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 352–363, 2013.

structures *to be counted*. Typical parameters associated with the input graph are, e.g., its treewidth, cliquewidth or genus. For instance, a counting analogue of Courcelle's famous theorem [2] is known [7]: Given a graph G of treewidth $\mathrm{tw}(G)$ and a formula $\phi(X)$ in monadic second-order logic over graphs with a free set variable X, counting the sets X with $G \models \phi(X)$ is fpt in $\mathrm{tw}(G)$. This implies that counting perfect (or general) matchings in G is fpt in $\mathrm{tw}(G)$.

A natural parameter associated with the structures to be counted is their *size*. This includes the results that counting k-vertex covers is fpt in k, while counting k-paths, k-cliques or k-cycles are each #W[1]-hard, all proven in [4].

Counting k-Matchings: It was conjectured in [4] that counting k-matchings on bipartite graphs is #W[1]-hard in the parameter k. The problem for general graphs is an open problem in [5]. The conjecture was later backed up by a proof [1] that counting *weighted* k-matchings is indeed #W[1]-hard: Let $G = (V, E, w)$ be an edge-weighted bipartite graph and assign to every matching $M \subseteq E$ the weight $w(M) := \prod_{e \in M} w(e)$. It was shown that, for a particular $w : E \to \mathbb{Z}$, computing the sum $\sum_M w(M)$ over matchings M in G is #W[1]-hard.

Also, the best known algorithms for counting k-matchings exhibit time bounds of the type $f(k)n^{\Theta(k)}$. Among these is [13] with a runtime of $O(2^{k+o(k)}\binom{n}{k/2})$.

Our Result: We show that counting k-matchings is #W[1]-hard on unweighted graphs without multiple edges or self-loops. It is known that weights in the sense of [1] can be simulated by parallel edges. This however creates multigraphs, and standard reductions to simple graphs fail. Our proof relies on a particular gadget construction, which is analyzed by tools from commutative algebra. This technique can probably also be applied to other counting problems.

2 Preliminaries

Parameterized Counting: Let p#Clique be the problem of counting cliques of size k in a graph G, parameterized by k. Define the class #W[1] as the set of parameterized counting problems A with $\mathsf{A} \leq^T_{fpt}$ p#Clique. Here, $\mathsf{A} \leq^T_{fpt} \mathsf{B}$ means that A admits an fpt-algorithm that solves instances (x, k) of A with oracle access to B, under the restriction that all oracle queries (y, k') feature $k' \leq g(k)$ for some computable $g : \mathbb{N} \to \mathbb{N}$. For a more formal definition, consider [4].

Polynomials: Let $\mathbf{x} = (x_1, \ldots, x_s)$ be a tuple of indeterminates and let $\mathbb{N}^{\mathbf{x}}$ be the set of monomials over $\mathbf{x}$. Given a multivariate polynomial $p \in \mathbb{Z}[\mathbf{x}]$ and $\nu \in \mathbb{N}^{\mathbf{x}}$, write $c(\nu) \in \mathbb{Z}$ for the coefficient of monomial ν in p. This gives $p = \sum_\nu c(\nu) \cdot \nu$. Note that only finitely many $c(\nu)$ are non-zero.

Let $\mathbf{x} = \mathbf{y} \dot\cup \mathbf{z}$ be a partition of the indeterminates of p. We can equivalently consider $p \in (\mathbb{Z}[\mathbf{z}])[\mathbf{y}]$. For $\nu \in \mathbb{N}^{\mathbf{y}}$, define $[\nu]p$ as the uniquely determined polynomial $H_\nu \in \mathbb{Z}[\mathbf{z}]$ in the expansion $p(\mathbf{x}) = \sum_{\theta \in \mathbb{N}^{\mathbf{y}}} H_\theta(\mathbf{z}) \cdot \theta$.

Matchings: Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. Define $\mathcal{M}_k[G]$ as the set of k-matchings of G, let $m_k := |\mathcal{M}_k[G]|$ and define $\mathcal{M}[G] := \bigcup_{k \in \mathbb{N}} \mathcal{M}_k[G]$. For a formal indeterminate X, let $M(G; X) := \sum_k m_k X^k$ be the edge-generating

matching polynomial of G. Given $u \in V$ and $M \in \mathcal{M}[G]$, we write $u \in \text{sat}(M)$ and say that u is *matched* by M if $\{u, w\} \in M$ for some $w \in V$.

2.1 Algebraic Independence

Crucial parts of our proof rely on *algebraic independence*, a notion from commutative algebra. A general introduction to this topic is given in [6].

Definition 1. *Let $P = \{p_1, \ldots, p_t\} \subseteq \mathbb{Z}[x_1, \ldots, x_s]$ be a set of polynomials and let $\mathbf{y} = (\dot{p}_1, \ldots, \dot{p}_t)$ be a tuple of indeterminates. An annihilator for P is a polynomial $A \in \mathbb{Z}[\mathbf{y}]$ which annihilates P, i.e., $A(p_1, \ldots, p_t) \equiv 0$. If the only annihilator for P is the zero polynomial, we call P algebraically independent.*

Remark 1. In the previous definition, we wrote $\mathbf{y} = (\dot{p}_1, \ldots, \dot{p}_t)$ to highlight the correspondence between indeterminates and polynomials. In this paper, expressions of the form $\dot{p}$ always denote indeterminates.

Restricting the annihilator A to linear functions without mixed-variable terms yields an alternative definition of linear independence. Algebraic independence generalizes this by allowing "polynomial" instead of only linear combinations.

We require only two ingredients from the theory of algebraic independence: The classical Jacobian criterion allows us to reduce algebraic independence to linear independence, and Lemma 1 allows us to argue about annihilators of "almost-independent" sets. A proof of Theorem 1 can be found in [3].

Theorem 1. *Let $P = \{p_1, \ldots, p_t\} \subseteq \mathbb{Z}[\mathbf{x}]$. Then P is algebraically independent iff $\text{rank}(JP) = t$, where JP denotes the Jacobian matrix $(JP)_{i,j} = \partial p_i / \partial x_j$.*

Lemma 1. *Let $\mathfrak{I} = \mathbb{Z}[\mathbf{x}]$ and let $P, Q \subseteq \mathfrak{I}$ with $P = \{p_1, \ldots, p_r\}$ and $Q = \{q_1, \ldots, q_t\}$ such that $P \cup Q$ is algebraically independent. Let $s = p_1 + \ldots + p_r$.*

Define indeterminates $\dot{s}$, $\mathbf{p} = (\dot{p}_1, \ldots, \dot{p}_r)$ and $\mathbf{q} = (\dot{q}_1, \ldots, \dot{q}_t)$, and define a ring $\mathfrak{O} := \mathbb{Z}[\dot{s}, \mathbf{p}, \mathbf{q}]$. Let $A \in \mathfrak{O}$ be an arbitrary annihilator for the set $\{s\} \cup P \cup Q$.

Let $\nu \in \mathbb{N}^{\mathbf{q}}$ be arbitrary, and consider $[\nu]A \in \mathbb{Z}[\dot{s}, \mathbf{p}]$. Applying the substitution $\dot{s} := \dot{p}_1 + \ldots + \dot{p}_r$ to $[\nu]A$ yields a polynomial $A_\nu \in \mathbb{Z}[\mathbf{p}]$ with $A_\nu \equiv 0$.

Proof. Since A annihilates $\{s\} \cup P \cup Q$, we have $A(s, p_1, \ldots, p_r, q_1, \ldots, q_t) \equiv 0$. Considering A from $(\mathbb{Z}[\dot{s}, \mathbf{p}])[\mathbf{q}]$, this equation can be rewritten as

$$\sum_{\nu \in \mathbb{N}^{\mathbf{q}}} ([\nu]A)(s, p_1, \ldots, p_r) \cdot \nu(q_1, \ldots, q_t) \equiv 0. \tag{1}$$

Note that $([\nu]A)(s, p_1, \ldots, p_r) = A_\nu(p_1, \ldots, p_r)$ since $\dot{s} := \dot{p}_1 + \ldots + \dot{p}_r$ and $s = p_1 + \ldots + p_r$. If $A_\nu \not\equiv 0$ for some ν, then (1) displays a nontrivial annihilator for $P \cup Q$ after substitution of $\dot{s}$, contradicting its independence. □

2.2 Outline of the Reduction

We prove #W[1]-hardness of counting k-matchings by a reduction from the problem p#CC of counting k-partial cycle covers, whose #W[1]-hardness was shown in [1]. Let us first define the notion of k-partial cycle covers:

Definition 2. *[1] Let $G = (V, E)$ be a directed graph and let $t \in \mathbb{N}$. A* t-partial path-cycle cover *C in G is a set $C \subseteq E$ with $|C| = t$ that consists of a vertex-disjoint union of simple paths and cycles.*

Let $\rho(C)$ be the number of paths in C. We call C a t-partial cycle *cover if $\rho(C) = 0$. The set of t-partial path-cycle covers in G with ρ paths is denoted by $\mathcal{PC}_{t,\rho}[G]$. Define $\mathcal{PC}_t[G] := \bigcup_\rho \mathcal{PC}_{t,\rho}[G]$ and extend this to $\mathcal{PC}[G] := \bigcup_t \mathcal{PC}_t[G]$. For $t, \rho \in \mathbb{N}$, let $m_{t,\rho} := |\mathcal{PC}_{t,\rho}[G]|$, if G is clear from the context.*

For compatibility with [1], define $\mathcal{C}_k[G] := \mathcal{PC}_{k,0}[G]$. In the following sections, only two parameterized counting problems will be relevant:

p#CC
Input: directed graph G, $k \in \mathbb{N}$
Parameter: k
Output: $\lvert\mathcal{C}_k[G]\rvert$

p#Match
Input: undirected graph G, $k \in \mathbb{N}$
Parameter: k
Output: $\lvert\mathcal{M}_k[G]\rvert$

It holds that p#Match $\in$ #W[1], as it is subsumed by the more general problem of counting embeddings from [4]. In the following, we sketch the reduction p#CC $\leq^T_{fpt}$ p#Match. The rest of this paper provides its details. We obtain:

Theorem 2. *The problem* p#Match *is* #W[1]*-complete.*

The reduction works as follows: To begin with, we are given a directed graph G and $k \in \mathbb{N}$ as inputs, and we wish to count the number of k-partial cycle covers in G. We are also given an oracle for p#Match that can be queried about the numbers of K-matchings in arbitrary graphs, provided that $K \leq g(k)$, where g is computable. It turns out that our queries even satisfy $K \leq 3k$, and that the reduction can be carried out in polynomial time.

The proof begins in Section 3 with the description of a particular graph transformation: First, we construct an undirected graph G' and a bijection $S : \mathcal{PC}_k[G] \to \mathcal{M}_k[G']$. Next, we apply gadgets to G' to obtain a graph $H = H(G)$ and show that, for $K = 3k$, the quantity $|\mathcal{M}_K[H]|$ can be written as a particular weighted sum over the matchings $M \in \mathcal{M}_k[G']$. The weight of M in this sum depends on the number of paths in its associated path-cycle cover $S^{-1}(M)$.

We proceed to show in Sections 3 and 4 that the weights in this sum in fact allow to distinguish matchings $M \in \mathcal{M}_k[G']$ according to the number of paths in $S^{-1}(M)$. Finally, we use this in Section 4.2 to recover the number of k-partial path-cycle covers with zero paths in G by oracle calls to p#Match.

3 The Gadget Construction

3.1 Global Construction

We want to count k-partial cycle covers in a directed graph G with an oracle for p#Match. Let $n = |V|$. First, we define a graph $S(G)$ as in [1]:

Definition 3. *[1] Given a directed graph $G = (V, E)$, replace each vertex $w \in V$ by vertices w^{in} and w^{out}, and replace each $(u, v) \in E$ by the undirected edge $\{u^{out}, v^{in}\}$. We call the resulting graph the* split graph *$S(G)$. Let $G' = S(G)$.*

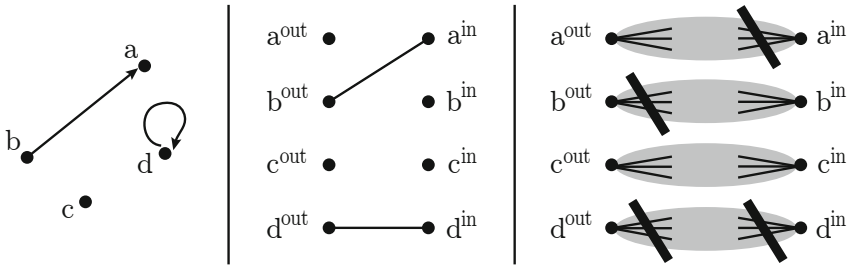


Fig. 1. (left) A partial path-cycle cover C. **(middle)** The matching $M' = S(C)$. **(right)** For $w \in V$, the gray area between $\{w^{out}, w^{in}\}$ symbolizes $\mathcal{V}_w$. Cancelled edges indicate edges in $\mathcal{V}_w$ that cannot be included into M' since $\{w^{out}, w^{in}\}$ is in-blocked, out-blocked or blocked, as seen in the first, second and fourth pair, respectively.

The graph G' is bipartite, and considering S as a function, it induces a bijection between $\mathcal{PC}_t[G]$ and $\mathcal{M}_t[G']$ for all $t \in \mathbb{N}$, as shown in [1]. Consider the left and middle part of Fig. 1 for an example. We also observe the following:

Remark 2. Let $C \in \mathcal{PC}_{t,\rho}[G]$. Since C has ρ paths, there are ρ vertices incident with only an incoming edge in C, another ρ vertices incident with only an outgoing edge, and $t - \rho$ vertices incident with both an incoming and an outgoing edge. The remaining $n - t - \rho$ vertices are not incident with any edge in C.

This translates to $M = S(C)$ as follows: Consider pairs $\{w^{out}, w^{in}\} \subseteq V(G')$, for $w \in V(G)$. There are ρ such pairs with $w^{in} \in \text{sat}(M)$ and $w^{out} \notin \text{sat}(M)$. We call such pairs *in-blocked*. There are another ρ pairs with $w^{out} \in \text{sat}(M)$ and $w^{in} \notin \text{sat}(M)$, which we call *out-blocked*. There are $t - \rho$ pairs with both $w^{out}, w^{in} \in \text{sat}(M)$, which we call *blocked*. The remaining $n - t - \rho$ pairs feature $\text{sat}(M) \cap \{w^{out}, w^{in}\} = \emptyset$. We call these pairs *free*. □

This roughly implies the following: If we can distinguish matchings $M \in \mathcal{M}_t[G']$ according to how many pairs $\{w^{out}, w^{in}\}$ occur in the above states, then we can hope to distinguish t-partial path-cycle covers $C = S^{-1}(M)$ by $\rho(C)$.

In the remaining section, we present a particular construction that achieves exactly this, as will be proven in Section 4. The construction uses a gadget, i.e., an undirected graph $\mathcal{V}$ with two special vertices u^{out} and u^{in} that can be inserted locally into G' to yield a graph $H = H(G)$.[1]

Definition 4. *Given a graph G, define a graph $H = H(G)$ as follows: First, let $G' = S(G)$. For each $w \in V(G)$, add a fresh copy $\mathcal{V}_w$ of $\mathcal{V}$ to G', identifying the vertex $w^{out} \in V(G')$ with $u^{out} \in V(\mathcal{V}_w)$, and identifying $w^{in} \in V(G')$ with $u^{in} \in V(\mathcal{V}_w)$. Note that, by construction, G' appears as a subgraph in H.*

Let $s \in V(G')$ with $s \in \{w^{out}, w^{in}\}$ for $w \in V(G)$. If $M \in \mathcal{M}[H]$ and $s \in \text{sat}(M)$, then $s \in e$ for some $e \in M$. Then either $e \in E(\mathcal{V}_w)$, in which case we call s *internally* matched, or $e \in E(G')$ and s is *externally* matched. If

[1] The actual definition of $\mathcal{V}$ is irrelevant for now and is treated in the next subsection.

s is externally matched, then all edges in M that stem from $E(\mathcal{V}_w)$ must be contained in $E(\mathcal{V}_w - s)$. Thus, when extending a matching $N \in \mathcal{M}[G']$ to some $M \in \mathcal{M}[H]$ by including edges from $\mathcal{V}_w$, we have to distinguish the state of the pair $\{w^{out}, w^{in}\}$ in N.[2] This is illustrated in the right part of Fig. 1.

We account for this by associating four restricted matching polynomials with the gadget $\mathcal{V}$. Recall that $\mathcal{V}$ features special vertices u^{out} and u^{in}. The restricted polynomials $M^A(\mathcal{V})$ are indexed by $A \subseteq \{u^{out}, u^{in}\}$ and are defined like $M(\mathcal{V})$, except that they count only matchings $M \in \mathcal{M}[\mathcal{V}]$ with $\mathrm{sat}(M) \cap \{u^{out}, u^{in}\} \subseteq A$.

Definition 5. *Let $\mathcal{V}$ be a graph with $u^{out}, u^{in} \in V(\mathcal{V})$. For $A \subseteq \{u^{out}, u^{in}\}$, let*

$$M^A(\mathcal{V}; X) := \sum_{\substack{M \in \mathcal{M}[\mathcal{V}] \\ \mathrm{sat}(M) \cap \{u^{out}, u^{in}\} \subseteq A}} X^{|M|}.$$

We consider $\mathcal{V}$ fixed. Define $B := M^{\emptyset}(\mathcal{V})$, $U := M^{\{u^{out}\}}(\mathcal{V})$, $V := M^{\{u^{in}\}}(\mathcal{V})$, and $F := M^{\{u^{out}, u^{in}\}}(\mathcal{V})$. Finally, for $t, \rho \in \mathbb{N}$ with $\rho \leq t$ and $n - t - \rho \geq 0$, define a polynomial $\mathsf{Mix}^{(t,\rho)} \in \mathbb{Z}[X]$ by $\mathsf{Mix}^{(t,\rho)} := B^{t-\rho} \cdot U^{\rho} \cdot V^{\rho} \cdot F^{n-t-\rho}$.

The polynomials $\mathsf{Mix}^{(t,\rho)}$ are crucial in Section 4 because the matching polynomial $M(H)$ can be written as a weighted sum over the path-cycle covers $C \in \mathcal{PC}[G]$ such that $C \in \mathcal{PC}_{t,\rho}[G]$ is weighted by $X^t \cdot \mathsf{Mix}^{(t,\rho)}$. This is stated in the following lemma, which can be proved using standard combinatorial arguments. Recall that $m_{t,\rho}(G) = |\mathcal{PC}_{t,\rho}[G]|$ by Definition 2.

Lemma 2. *Let G be a graph and let $H = H(G)$ as in Definition 4. Then*

$$M(H) = \sum_{0 \leq \rho \leq t \leq n} m_{t,\rho}(G) \cdot X^t \cdot \mathsf{Mix}^{(t,\rho)}.$$

We close this subsection with a remark about the coefficients of B, U, V, F:

Remark 3. Note that $[X^0]\mathsf{D} = 1$ for all $\mathsf{D} \in \{B, U, V, F\}$. Furthermore, it can be verified that $[X^1]F = [X^1](U + V - B)$ if $\{u, v\} \notin E(\mathcal{V})$. The gadget introduced in the next subsection will feature $\{u, v\} \notin E(\mathcal{V})$. □

3.2 Local Construction: The Venn Gadget

We are ready to provide an explicit construction for the gadget $\mathcal{V}$: The *Venn gadget* $\mathcal{V}(\mathbf{x})$ is an undirected graph with special vertices u^{out} and u^{in}, as shown in Fig. 2 on the next page. Its precise manifestation depends on a tuple

$$\mathbf{x} = (a_\emptyset, a_u, a_v, a_{uv},\ b_\emptyset, b_u, b_v, b_{uv},\ c_u, c_v, c_{uv}) \in \mathbb{N}^{11}. \tag{2}$$

The eleven parameters, which will be considered as indeterminates later, are named so as to reflect the particular set system represented by the gadget.

[2] This might evoke memories of *matchgates* in the readership of [12].

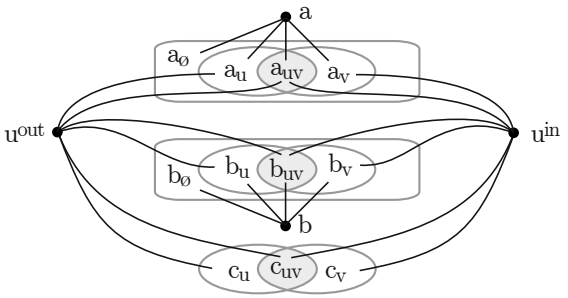


Fig. 2. The Venn gadget $\mathcal{V}$ features named vertices u^{out}, u^{in}, a and b. The other vertices are partitioned into the disjoint sets $a_\emptyset, \ldots, c_{uv}$. In this figure, an edge leading from a special vertex w into a set S symbolizes that w is adjacent to all vertices in S.

Definition 6. *Given a tuple* $\mathbf{x} \in \mathbb{N}^{11}$ *as specified in (2), the* Venn gadget $\mathcal{V}(\mathbf{x})$ *is constructed as follows from the empty graph:*

1. *Create* $(a_\emptyset + a_u + a_v + a_{uv}) + (b_\emptyset + b_u + b_v + b_{uv}) + (c_u + c_v + c_{uv})$ *fresh and unnamed vertices. Abusing notation slightly, group these vertices into sets* $a_\emptyset, \ldots, c_{uv}$ *in the obvious way.*
2. *Create a special vertex* u^{out} *adjacent to all of* $(a_u \cup a_{uv}) \cup (b_u \cup b_{uv}) \cup (c_u \cup c_{uv})$.
3. *Create a special vertex* u^{in} *adjacent to all of* $(a_v \cup a_{uv}) \cup (b_v \cup b_{uv}) \cup (c_v \cup c_{uv})$.
4. *Create a vertex* a *adjacent to all of* $a_\emptyset \cup a_u \cup a_v \cup a_{uv}$.
5. *Create a vertex* b *adjacent to all of* $b_\emptyset \cup b_u \cup b_v \cup b_{uv}$.

Remark 4. Note that constructing $\mathcal{V}(\mathbf{x})$ for different $\mathbf{x} \in \mathbb{N}^{11}$ yields different graphs. Thus, using the gadget $\mathcal{V}(\mathbf{x})$ to construct the graph H in Definition 4 in fact yields a graph $H = H(\mathbf{x})$ that also depends on $\mathbf{x}$.

Furthermore, when considering $\mathbf{x}$ as a tuple of indeterminates, the matching polynomials B, U, V, F associated with $\mathcal{V}$, introduced in Definition 5, are easily seen to be elements in $\mathbb{Z}[X, \mathbf{x}]$. Equivalently, we can define $\mathfrak{I} := \mathbb{Z}[\mathbf{x}]$ and say that $B, U, V, F \in \mathfrak{I}[X]$, where X denotes a formal generating variable. □

We now consider the coefficients of the polynomials $B, U, V, F \in \mathfrak{I}[X]$ from Remark 4. Note that these coefficients are elements of $\mathfrak{I} = \mathbb{Z}[\mathbf{x}]$, and thus in turn polynomials. We show that the set of coefficients is "almost" algebraically independent, in the sense that it allows Lemma 1 to be invoked.

First observe that $\deg(B) = 2$, $\deg(U) = \deg(V) = 3$ and $\deg(F) = 4$, as these are the maximum cardinalities of matchings counted by B, U, V, F, respectively. These four polynomials therefore feature at most 16 non-zero coefficients in total. For $\mathsf{D} \in \{B, U, V, F\}$, abbreviate $\mathsf{D}_i := [X^i]\mathsf{D} \in \mathfrak{I}$.

Furthermore, note that $B_0 = V_0 = U_0 = F_0 = 1$ by Remark 3. We will ignore these four coefficients from now on, for reasons that will become clear in Section 4. Let $\mathcal{Y}$ be the set of all *other* coefficients of B, U, V, F. For convenience:

$$\mathcal{Y} := \{B_1, B_2,\ U_1, U_2, U_3,\ V_1, V_2, V_3,\ F_1, F_2, F_3, F_4\}.$$

Additionally, let $\mathcal{B} := \{B_1, U_1, V_1, F_1\}$. Note that $F_1 = U_1 + V_1 - B_1$ by Remark 3. The set $\mathcal{B}$ is thus algebraically (and even linearly) dependent. We now consider the set $\mathcal{Y}' := \mathcal{Y} \setminus \{F_1\}$. After computing the elements in $\mathcal{Y}'$ explicitly and verifying that $\det(J\mathcal{Y}') \not\equiv 0$, we obtain the following lemma as a corollary from Theorem 1:

Lemma 3. *The set $\mathcal{Y}' \subseteq \mathfrak{I}$ is algebraically independent.*

We can now apply Lemma 1 verbatim to obtain the following corollary. It states a restriction on annihilators for $\mathcal{Y}$ which will be used in Section 4.1.

Corollary 1 (of Lemma 1). *Let $P := \mathcal{B} \setminus \{F_1\}$ and $Q := \mathcal{Y} \setminus \mathcal{B}$. By Lemma 3, the set $P \cup Q = \mathcal{Y}'$ is algebraically independent. Recall that $F_1 = U_1 + V_1 - B_1$.*

Define indeterminates $\dot{F}_1$, $\mathbf{p} = (\dot{B}_1, \dot{U}_1, \dot{V}_1)$ and $\mathbf{q}$, where $\mathbf{q}$ represents Q, and let $\mathbf{y} = (\dot{F}_1, \mathbf{p}, \mathbf{q})$. Let $\mathfrak{O} = \mathbb{Z}[\mathbf{y}]$ and let $A \in \mathfrak{O}$ annihilate $\mathcal{Y} = \{F_1\} \cup P \cup Q$.

Let $\theta^* = \dot{B}_2^b$ with $b > 0$ and consider $[\theta^*]A \in \mathbb{Z}[\dot{F}_1, \mathbf{p}]$. Applying the substitution $\dot{F}_1 := \dot{U}_1 + \dot{V}_1 - \dot{B}_1$ to $[\theta^*]A$ yields a polynomial $A_{\theta^*} \in \mathbb{Z}[\mathbf{p}]$ with $A_{\theta^*} \equiv 0$.

4 Analysis of the Graph Construction

Recall that we wish to determine $m_{k,0}$, where $m_{t,\rho}$ denotes the number of t-partial path-cycle covers with ρ paths in G. We fix k and $K := 3k$. We also fix $\mathbf{y}$ and $\mathfrak{O} = \mathbb{Z}[\mathbf{y}]$ as in Corollary 1, as well as $\mathbf{x}$ and $\mathfrak{I} = \mathbb{Z}[\mathbf{x}]$ as in Remark 4.

The indeterminates in $\mathbf{y}$ correspond to $\mathcal{Y}$ from Section 3.2. We extend this view by considering the polynomials B, U, V, F and $\mathsf{Mix}^{(t,\rho)} \in \mathbb{Z}[X]$ from Definition 5 formally as elements from $\mathfrak{O}[X]$, writing $\mathsf{Mix}_{\mathfrak{O}}^{(t,\rho)}$ to make this explicit:

Definition 7. *For $\mathsf{D} \in \{B, U, V, F\}$, let $\mathsf{D}_{\mathfrak{O}} = \sum_{i=1}^{\deg(\mathsf{D})} \dot{\mathsf{D}}_i X^i \in \mathfrak{O}[X]$. Define $\mathsf{Mix}_{\mathfrak{O}}^{(t,\rho)} \in \mathfrak{O}[X]$ exactly as $\mathsf{Mix}^{(t,\rho)}$ in Definition 5, but replace any D by $\mathsf{D}_{\mathfrak{O}}$.*

Let $\mathsf{Mix}_{\mathfrak{O}} \in \mathfrak{O}^{(K+1)\times(K+1)}$ be the matrix whose entry at (t, ρ) is $[X^{K-t}]\mathsf{Mix}_{\mathfrak{O}}^{(t,\rho)}$ for $0 \leq \rho \leq t \leq K$, and 0 else. Also write $\mathsf{Mix}_{\mathfrak{O}}$ for the set of its entries.

We similarly define $M_{\mathfrak{O}}(H) \in \mathfrak{O}[X]$ by formally replacing coefficients of Venn gadgets with indeterminates from $\mathbf{y}$. Extending Lemma 2, we obtain:

Lemma 4. *Let $H = H(G)$ according to Definition 4. For matrices A, B of the same dimensions, let $A \odot B := \sum_{ij} A_{ij} B_{ij}$. Then*

$$[X^K]M_{\mathfrak{O}}(H) = \underbrace{\begin{pmatrix} [X^K]\mathsf{Mix}_{\mathfrak{O}}^{(0,0)} & \dots & [X^0]\mathsf{Mix}_{\mathfrak{O}}^{(K,0)} \\ & \ddots & \vdots \\ & & [X^0]\mathsf{Mix}_{\mathfrak{O}}^{(K,K)} \end{pmatrix}}_{=\mathsf{Mix}_{\mathfrak{O}}} \odot \begin{pmatrix} m_{0,0} & \dots & m_{K,0} \\ & \ddots & \vdots \\ & & m_{K,K} \end{pmatrix}.$$

This yields a formal "linear combination" of the quantities $m_{t,\rho}$ with coefficients from $\mathfrak{O}$. For $t = k$ and $0 \leq \rho \leq k$, the interesting quantities $m_{k,\rho}$ appear in it as

$$[X^K]M_{\mathfrak{O}}(H) = \ldots + m_{k,0}[X^{2k}]\mathsf{Mix}_{\mathfrak{O}}^{(k,0)} + \ldots + m_{k,k}[X^{2k}]\mathsf{Mix}_{\mathfrak{O}}^{(k,k)} + \ldots \quad (3)$$

In Section 4.1, we substitute the polynomials $\mathcal{Y} \subseteq \mathfrak{I}$ from Section 3.2 into the indeterminates $\mathbf{y}$, yielding a matrix $\mathsf{Mix}_{\mathfrak{I}} \in \mathfrak{I}^{(K+1)\times(K+1)}$. We show that, after this substitution, the polynomial $p^* := [X^{2k}]\mathsf{Mix}_{\mathfrak{I}}^{(k,0)}$ associated with $m_{k,0}$ in (3) is *special*, in the sense that it cannot be written as a linear combination (with rational coefficients) of the other polynomials in $\mathsf{Mix}_{\mathfrak{I}}$.

In Section 4.2, we show that a linear system of equations in the unknowns $m_{t,\rho}$ can be set up from (3) by evaluating[3] the entries of $\mathsf{Mix}_{\mathfrak{I}}$ on $\xi \in \mathbb{N}^{11}$ and using oracle calls on graphs derived by the gadget construction from Section 3. This system will feature $O(k^{11})$ linear equations, whose integer coefficients can be computed in time $n^{O(1)}$. Furthermore, the specialness of p^* will imply that the system can be solved unambiguously for $m_{k,0}$. This proves Theorem 2.

4.1 The Polynomial p^* Is Special

We consider expansions of the polynomials $p \in \mathsf{Mix}_{\mathfrak{O}}$ into monomials over $\mathbf{y}$. This is used to show that, after substitution of $\mathcal{Y}$ from Section 3.2 into $\mathbf{y}$, the polynomial $p^* = [X^{2k}]\mathsf{Mix}_{\mathfrak{I}}^{(k,0)}$ associated with $m_{k,0}$ satisfies the following:

Theorem 3. *Substitute $\mathcal{Y}$ into $\mathbf{y}$ in all of $\mathsf{Mix}_{\mathfrak{O}}$ to obtain the matrix $\mathsf{Mix}_{\mathfrak{I}}$. Then $p^* = [X^{2k}]\mathsf{Mix}_{\mathfrak{I}}^{(k,0)}$ is not in the span of the other entries in $\mathsf{Mix}_{\mathfrak{I}}$. Formally, if*

$$\sum_{0\leq\rho\leq t\leq K} \alpha_{t,\rho} \cdot [X^{K-t}]\mathsf{Mix}_{\mathfrak{I}}^{(t,\rho)} \equiv 0, \tag{4}$$

with $\alpha_{t,\rho} \in \mathbb{Q}$ for all $0 \leq \rho \leq t \leq K$, then $\alpha_{k,0} = 0$.

This theorem will be proven at the end of this subsection. We first consider polynomials $p \in \mathsf{Mix}_{\mathfrak{O}}$ and require some notation for the set of monomials *appearing* in p. Recall that $\mathfrak{O} = \mathbb{Z}[\mathbf{y}]$, and note that $[\theta]p \in \mathbb{Z}$ if $p \in \mathfrak{O}$ and $\theta \in \mathbb{N}^{\mathbf{y}}$.

Definition 8. *For $p \in \mathfrak{O}$, let $\mathfrak{M}p = \{\theta \in \mathbb{N}^{\mathbf{y}} \mid [\theta]p \neq 0\}$. For $P \subseteq \mathfrak{O}$, define $\mathfrak{M}P = \bigcup_{p\in P} \mathfrak{M}p$. If $\theta \in \mathbb{N}^{\mathbf{y}}$ and $\theta \in \mathfrak{M}P$, we say that θ* appears *in P.*

Our proof of Theorem 3 proceeds as follows: We first identify a special monomial $\theta^* \in \mathbb{N}^{\mathbf{y}}$ and show that, among all $p \in \mathsf{Mix}_{\mathfrak{O}}$, the monomial θ^* appears only in $p = p^*$. Using this, we show that containment of p^* in the span of the other polynomials yields an annihilator for $\mathcal{Y}$ that contradicts Corollary 1.

To begin with, we define several quantities associated with monomials in $\mathbb{N}^{\mathbf{y}}$:

Definition 9. *Let $\theta \in \mathbb{N}^{\mathbf{y}}$ and observe that θ is of the form*

$$\theta = (\dot{B}_1^{b_1}\dot{B}_2^{b_2})(\dot{U}_1^{u_1}\dot{U}_2^{u_2}\dot{U}_3^{u_3})(\dot{V}_1^{v_1}\dot{V}_2^{v_2}\dot{V}_3^{v_3})(\dot{F}_1^{f_1}\dot{F}_2^{f_2}\dot{F}_3^{f_3}\dot{F}_4^{f_4}).$$

Define $\mathrm{td}(\theta) := \sum_{i=1}^{4} i(b_i + u_i + v_i + f_i)$. *Let $\Theta := \mathfrak{M}\mathsf{Mix}_{\mathfrak{O}}$. For $\ell \in \mathbb{N}$, let $\Theta_\ell := \Theta \cap \{\theta \mid \mathrm{td}(\theta) = \ell\}$. Let* $\mathrm{occ}(\theta) := (\sum_i b_i, \sum_i u_i, \sum_i v_i, \sum_i f_i)$ *and write* $\mathrm{occ}_B(\theta)$ *for the first entry of* $\mathrm{occ}(\theta)$.

[3] Recall that $\mathfrak{I} = \mathbb{Z}[\mathbf{x}]$, where $\mathbf{x}$ is the tuple of 11 indeterminates from (2) in Section 3.2. Thus, evaluating $\mathsf{Mix}_{\mathfrak{I}}^{(t,\rho)}(\xi)$ at $\xi \in \mathbb{N}^{11}$ yields an integer value.

Example 1. Let $\theta = \dot{B}_1^1 \dot{B}_2^2 \dot{U}_2^4 \dot{V}_2^5 \dot{F}_1^6$. Then $\mathrm{td}(\theta) = 1 \cdot (1+6) + 2 \cdot (2+4+5) = 29$ and $\mathrm{occ}(\theta) = (3,4,5,6)$. Furthermore, we have $\mathrm{occ}_B(\theta) = 3$.

This notation is used for the statement of the following lemma, which follows from a relatively straightforward application of the multinomial theorem.

Lemma 5. *Let $0 \leq t \leq K$. For $a, b_1, \ldots, b_\ell \in \mathbb{N}$ with $s := \sum_i b_i \leq a$, let $\binom{a}{b_1,\ldots,b_\ell} = \frac{a!}{b_1!\ldots b_\ell!(a-s)!}$. With $\theta \in \mathbb{N}^{\mathcal{Y}}$ written as in Definition 9, we have*

$$[X^{K-t}]\mathsf{Mix}_{\mathfrak{D}}^{(t,\rho)} = \sum_{\theta \in \Theta_{K-t}} \underbrace{\binom{t-\rho}{b_1,b_2}\binom{\rho}{u_1,u_2,u_3}\binom{\rho}{v_1,v_2,v_3}\binom{n-t-\rho}{f_1,f_2,f_3,f_4}}_{=:\lambda_{t,\rho}(\theta)} \theta.$$

Corollary 2. A monomial $\theta \in \Theta$ appears in $[X^{K-t}]\mathsf{Mix}_{\mathfrak{D}}^{(t,\rho)}$ iff $\mathrm{td}(\theta) = K - t$ and $\lambda_{t,\rho}(\theta) \neq 0$. The second condition is true iff $\mathrm{occ}(\theta) \leq (t-\rho, \rho, \rho, n-t-\rho)$, where $\leq$ is considered component-wise.

We now define the *special monomial* $\theta^* := \dot{B}_2^k$ and show that it appears only in the previously defined special polynomial $p^* = [X^{2k}]\mathsf{Mix}_{\mathfrak{D}}^{(k,0)}$.

Lemma 6. *If $\theta \in \Theta$ contains $\theta^* = \dot{B}_2^k$ as a factor, then $\theta = \theta^*$. Furthermore, if θ^* appears in $p \in \mathsf{Mix}_{\mathfrak{D}}$, then $p = p^*$. In fact, we have $[\theta^*]p^* = 1$.*

Proof. If $\theta \in \Theta$ contains $\dot{B}_2^k$, then $\mathrm{td}(\theta) \geq 2k$. Since $\theta \in \Theta$, it must appear in $[X^{K-t}]\mathsf{Mix}_{\mathfrak{D}}^{(t,\rho)}$ for some $0 \leq \rho \leq t \leq K$. Then $K - t \geq \mathrm{td}(\theta)$ by Corollary 2. Recall that $K = 3k$, implying $t \leq k$. Since θ contains $\dot{B}_2^k$, we have $\mathrm{occ}_B(\theta) \geq k$. But by Corollary 2, we also have $\mathrm{occ}_B(\theta) \leq t - \rho$.

The last two inequalities and $t \leq k$ imply $\rho = 0$ and $\mathrm{occ}_B(\theta) = k$. Thus θ appears only in p^*. But then $\mathrm{td}(\theta) = 2k$, and thus $\theta = \theta^*$. Finally, $[\theta^*]p^* = \lambda_{k,0}(\theta^*) = 1$ follows independently from Lemma 5. □

This allows us to finish the subsection with the promised proof of Theorem 3.

Proof (of Theorem 3). Assume there were coefficients $\alpha_{t,\rho}$ satisfying (4) with $\alpha_{k,0} \neq 0$. With $\lambda_{t,\rho}(\theta)$ from Lemma 5, write $[X^{K-t}]\mathsf{Mix}_{\mathfrak{D}}^{(t,\rho)} = \sum_{\theta \in \Theta} \lambda_{t,\rho}(\theta) \cdot \theta$ and rearrange (4) to obtain

$$A := \left(\alpha_{k,0} \cdot \sum_{\theta \in \Theta} \lambda_{k,0}(\theta) \cdot \theta \right) + \sum_{\theta \in \Theta} \theta \sum_{\substack{0 \leq \rho \leq t \leq K \\ (t,\rho) \neq (k,0)}} \alpha_{t,\rho} \cdot \lambda_{t,\rho}(\theta) \equiv 0. \tag{5}$$

By Lemma 6, the monomial $\theta^* = \dot{B}_2^k$ appears only in the parentheses and has $\lambda_{k,0}(\theta^*) = 1$. Regrouping (5) yields $A = \alpha_{k,0} \cdot \theta^* + \sum_{\theta \neq \theta^*} \mu(\theta) \cdot \theta$, for new coefficients μ, with the property that $A(B_1, \ldots, F_4) \equiv 0$.

Also by Lemma 6, the only monomial in A that contains θ^* is θ^* itself. Therefore, A is a nontrivial annihilator for the set $\mathcal{Y}$ from Section 3.2, with the property that $[\theta^*]A = \alpha_{k,0} \in \mathbb{Q}$ is non-zero. Corollary 1 then leads to a contradiction: Since $[\theta^*]A \neq 0$ is constant, it is unaffected by the substitution $\dot{F}_1 := \dot{U}_1 + \dot{V}_1 - \dot{B}_1$, thus contradicting $A_{\theta^*} \equiv 0$ from Corollary 1. □

4.2 Deriving Linear Equations from Mix

In this subsection, we complete the proof of Theorem 2. For this, we substitute the elements in $\mathcal{Y}$ from Section 3.2 into $\mathsf{Mix}_{\mathfrak{O}}$. Using the gadget $\mathcal{V}(\mathbf{x})$, we can evaluate the resulting polynomials $\mathsf{Mix}_{\mathfrak{I}}^{(t,\rho)}$ to yield integer values.

Definition 10. *For $\xi \in \mathbb{N}^{11}$, let $\mathsf{Mix}(\xi) \in \mathbb{Z}^{(K+1)\times(K+1)}$ be the matrix obtained from $\mathsf{Mix}_{\mathfrak{I}}$ by evaluating each of its entries $p \in \mathfrak{I}$ at ξ.*

For $\Xi = (\xi_1, \ldots, \xi_D)$ with $\xi_i \in \mathbb{N}^{11}$, let $\mathsf{Mix}(\Xi) \in \mathbb{Z}^{D\times(K+1)^2}$ be such that the i-th row of $\mathsf{Mix}(\Xi)$ contains the entries of $\mathsf{Mix}(\xi_i)$ as a row vector. Consider columns of $\mathsf{Mix}(\Xi)$ to be indexed by pairs (t,ρ) and write $A^{(t,\rho)}$ for column (t,ρ).

Remark 5. If $|\Xi| \leq n^{O(1)}$ and all entries of Ξ have bit-length $n^{O(1)}$, then $\mathsf{Mix}(\Xi)$ can be computed in time $n^{O(1)}$: It holds by Definition 5 that $\mathsf{Mix}_{\mathfrak{O}}^{(t,\rho)} \in \mathfrak{O}[X]$ is the product of n polynomials, each of degree ≤ 4. Any $[X^\ell]\mathsf{Mix}_{\mathfrak{O}}^{(t,\rho)} \in \mathfrak{O}$ can therefore be computed, the elements in $\mathcal{Y}$ can be substituted into it, and the resulting $p \in \mathfrak{I}$ can be evaluated at any ξ_i, all in time $n^{O(1)}$.

In the following, we fix $\Xi = (\xi_1, \ldots, \xi_D)$, with $D = (K+1)^{11}$, to some enumeration of the grid $\{0, \ldots, K\}^{11}$. Furthermore, if $B \in \mathbb{Z}^{\ell\times b^2}$ is a matrix whose columns are indexed by pairs (i,j), and $C \in \mathbb{Z}^{b\times b}$, let $B \odot C \in \mathbb{Z}^\ell$ be defined by $(B \odot C)_t = \sum_{ij} B_{t,(i,j)} C_{ij}$. It can be checked that Lemma 4 implies

$$\mathsf{Mix}(\Xi) \odot \begin{pmatrix} m_{0,0} & \ldots & m_{K,0} \\ & \ddots & \vdots \\ & & m_{K,K} \end{pmatrix} = \begin{pmatrix} [X^K]M(H(\xi_1)) \\ \vdots \\ [X^K]M(H(\xi_D)) \end{pmatrix}, \tag{6}$$

with $H(\xi)$ for $\xi \in \mathbb{N}^{11}$ as in Remark 4. Recall that $[X^K]M(H(\xi)) \in \mathbb{N}$ counts the K-matchings in $H(\xi)$. Since $K = 3k$, we can thus evaluate the right-hand side of (6) by D oracle queries of the form $(H(\xi), K)$ to $\mathsf{p\#Match}$.

We consider (6) as a linear system of equations in the unknowns $m_{t,\rho}$. By Remark 5, $\mathsf{Mix}(\Xi)$ can be evaluated in time $n^{O(1)}$. This implies that a solution to (6) can also be found in time $n^{O(1)}$. The final and crucial step towards the proof of Theorem 2 now consists of showing that all solutions to (6) agree on their values for $m_{k,0}$. For this, we build upon Theorem 3 to show that column $(k,0)$ of $\mathsf{Mix}(\Xi)$ is not contained in the linear span of its other columns.

First, we require a generalization of the fact that every univariate polynomial $p \in \mathbb{Z}[x]$ of degree d that vanishes at $d+1$ points has $p \equiv 0$. This is stated in the following lemma, a corollary of the classical Schwartz-Zippel lemma [8,15].

Lemma 7. *Let $p \in \mathbb{Z}[x_1, \ldots, x_s]$ be a polynomial with $\deg(p) \leq d$. If $p(\xi) = 0$ holds for all $\xi \in \{0, \ldots, d\}^s$, then $p \equiv 0$.* □

From this, we obtain the last missing step for the proof of Theorem 2.

Lemma 8. *If $\sum_{t,\rho} \alpha_{t,\rho} \cdot A^{(t,\rho)} = 0$ for coefficients $\alpha_{t,\rho} \in \mathbb{Q}$, then $\alpha_{k,0} = 0$.*

Proof. Observe that $\deg(p) \leq K$ for $p \in \mathsf{Mix}_{\mathfrak{Y}}$: All monomials θ appearing in $p \in \mathsf{Mix}_{\mathfrak{O}}$ have $\mathrm{td}(\theta) \leq K$, and it can be verified that substituting $\mathcal{Y}$ into $\mathbf{y}$ then yields polynomials of degree $\leq K$. Also recall that $\mathfrak{Y} = \mathbb{Z}[\mathbf{x}]$ with $|\mathbf{x}| = 11$.

Assume there were coefficients $\alpha_{t,\rho}$ with $\alpha_{k,0} \neq 0$ and $\sum_{t,\rho} \alpha_{t,\rho} \cdot A^{(t,\rho)} = 0$. Then $q = \sum_{t,\rho} \alpha_{t,\rho} \cdot [X^{K-t}]\mathsf{Mix}_{\mathfrak{Y}}^{(t,\rho)}$ vanishes on $\{0, \ldots, K\}^{11}$. Thus $q \equiv 0$ by Lemma 7, contradicting Theorem 3 because $\alpha_{k,0} \neq 0$. □

Acknowledgements. The author wishes to thank Mingji Xia for sharing ideas that were fundamental for the Venn gadget from Section 3.2, Markus Bläser for mentioning algebraic independence in the right moment, and an anonymous reviewer, whose comments helped improving the presentation of the paper.

References

1. Bläser, M., Curticapean, R.: Weighted counting of k-matchings is #W[1]-hard. In: Thilikos, D.M., Woeginger, G.J. (eds.) IPEC 2012. LNCS, vol. 7535, pp. 171–181. Springer, Heidelberg (2012)
2. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and Computation 85(1), 12–75 (1990)
3. Ehrenborg, R., Rota, G.-C.: Apolarity and canonical forms for homogeneous polynomials. European Journal of Combinatorics 14(3), 157–181 (1993)
4. Flum, J., Grohe, M.: The parameterized complexity of counting problems. SIAM Journal on Computing, 538–547 (2002)
5. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer-Verlag New York, Inc., Secaucus (2006)
6. Hartshorne, R.: Algebraic geometry. Springer (1977)
7. Makowsky, J.: Algorithmic uses of the Feferman-Vaught theorem. Annals of Pure and Applied Logic 126, 159–213 (2004)
8. Schwartz, J.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM 27(4), 701–717 (1980)
9. Temperley, H., Fisher, M.: Dimer problem in statistical mechanics - an exact result. Philosophical Magazine 6(68) (1961) 1478–6435
10. Valiant, L.: The complexity of computing the permanent. Theoretical Computer Science 8(2), 189–201 (1979)
11. Valiant, L.: The complexity of enumeration and reliability problems. SIAM Journal on Computing 8(3), 410–421 (1979)
12. Valiant, L.: Holographic algorithms. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2004, pp. 306–315 (2004)
13. Vassilevska, V., Williams, R.: Finding, minimizing, and counting weighted subgraphs. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, pp. 455–464. ACM, New York (2009)
14. Xia, M., Zhang, P., Zhao, W.: Computational complexity of counting problems on 3-regular planar graphs. Theor. Comp. Sc. 384(1), 111–125 (2007)
15. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Ng, K.W. (ed.) EUROSAM 1979 and ISSAC 1979. LNCS, vol. 72, pp. 216–226. Springer, Heidelberg (1979)

Faster Exponential-Time Algorithms in Graphs of Bounded Average Degree*

Marek Cygan and Marcin Pilipczuk

Institute of Informatics, University of Warsaw, Poland
{cygan,malcin}@mimuw.edu.pl

Abstract. We first show that the Traveling Salesman Problem in an n-vertex graph with average degree bounded by d can be solved in $\mathcal{O}^\star(2^{(1-\varepsilon_d)n})$ time[1] and exponential space for a constant ε_d depending only on d. Thus, we generalize the recent results of Björklund et al. [TALG 2012] on graphs of bounded degree.

Then, we move to the problem of counting perfect matchings in a graph. We first present a simple algorithm for counting perfect matchings in an n-vertex graph in $\mathcal{O}^\star(2^{n/2})$ time and polynomial space; our algorithm matches the complexity bounds of the algorithm of Björklund [SODA 2012], but relies on inclusion-exclusion principle instead of algebraic transformations. Building upon this result, we show that the number of perfect matchings in an n-vertex graph with average degree bounded by d can be computed in $\mathcal{O}^\star(2^{(1-\varepsilon_{2d})n/2})$ time and exponential space, where ε_{2d} is the constant obtained by us for the Traveling Salesman Problem in graphs of average degree at most $2d$.

Moreover we obtain a simple algorithm that computes a permanent of an $n \times n$ matrix over an arbitrary commutative ring with at most dn nonzero entries using $\mathcal{O}^\star(2^{(1-1/(3.55d))n})$ time and ring operations, improving and simplifying the recent result of Izumi and Wadayama [FOCS 2012].

1 Introduction

Improving upon the 50-years old $\mathcal{O}^\star(2^n)$-time dynamic programming algorithms for the Traveling Salesman Problem by Bellman [1] and Held and Karp [7] is a major open problem in the field of exponential-time algorithms [14]. A similar situation appears when we want to count perfect matchings in a graph: a half-century old $\mathcal{O}^\star(2^{n/2})$-time algorithm of Ryser for bipartite graphs [12] has only recently been transferred to arbitrary graphs [3], and breaking these time complexity barriers seems like a very challenging task.

From a broader perspective, improving upon a trivial brute-force or a simple dynamic programming algorithm is one of the main goals the field of exponential-time algorithms. Although the last few years brought a number of positive results in that direction, most notably the $\mathcal{O}^\star(1.66^n)$ randomized algorithm for finding a Hamiltonian cycle in an undirected graph [2], it is conjectured (the so-called

* Partially supported by NCN grant N206567140 and Foundation for Polish Science.

[1] The $\mathcal{O}^\star$-notation suppresses factors polynomial in the input size.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 364–375, 2013.

Strong Exponential Time Hypothesis [8]) that the problem of satisfying a general CNF-SAT formulae does not admit any exponentially better algorithm than the trivial brute-force one. A number of lower bounds were proven using this assumption [6,10,11].

In 2008 Björklund et al. [5] observed that the classical dynamic programming algorithm for TSP can be trimmed to running time $\mathcal{O}^\star(2^{(1-\varepsilon_\Delta)n})$ in graphs of maximum degree Δ. The cost of this improvement is the use of exponential space, as we can no longer easily translate the dynamic programming algorithm into an inclusion-exclusion formula. The ideas from [5] were also applied to the Fast Subset Convolution algorithm, yielding a similar improvements for the problem of computing the chromatic number in graphs of bounded degree [4]. In this work, we investigate the class of graphs of bounded *average* degree, a significantly broader graph class than this of bounded maximum degree.

In the first part of our paper we generalize the results of [5].

Theorem 1. *For every $d \geq 1$ there exists a constant $\varepsilon_d > 0$ such that, given an n-vertex graph G of average degree bounded by d, in $\mathcal{O}^\star(2^{(1-\varepsilon_d)n})$ time and exponential space one can find in G a smallest weight Hamiltonian cycle.*

We note that in Theorem 1 the constant ε_d depends on d in a doubly-exponential manner, which is worse than the single-exponential behaviour of [5] in graphs of bounded degree.

The proof of Theorem 1 follows the same general approach as the results of [5] — we want to limit the number of states of the classical dynamic programming algorithm for TSP — but, in order to deal with graphs of bounded *average* degree, we need to introduce new concepts and tools. Recall that, by a standard averaging argument, if the average degree of an n-vertex graph G is bounded by d, for any $D \geq d$ there are at most dn/D vertices of degree at least D. However, it turns out that this bound cannot be tight for a large number of values of D at the same time. This simple observation lies at the heart of the proof of Theorem 1, as we may afford a more expensive branching on vertices of degree more than D provided that there are significantly less than dn/D of them.

In the second part, we move to the problem of counting perfect matchings in an n-vertex graph. We start with an observation that this problem can be reduced to a problem of counting some special types of cycle covers, which, in turn, can be done in $\mathcal{O}^\star(2^{n/2})$-time and polynomial space using the inclusion-exclusion principle (see Section 5.1). Note that an algorithm matching this bound in general graphs has been discovered only last year [3], in contrast to the 50-years old algorithm of Ryser [12] for bipartite graphs. Thus, we obtain a new proof of the main result of [3], using the inclusion-exclusion principle instead of advanced algebraic transformations.

Once we develop our inclusion-exclusion-based algorithm for counting perfect matchings, we may turn it into a dynamic programming algorithm and apply the ideas of Theorem 1, obtaining the following.

Theorem 2. *Given an n-vertex graph G of average degree bounded by d, in $\mathcal{O}^\star(2^{(1-\varepsilon_{2d})n/2})$ time and exponential space one can count the number of perfect*

matchings in G where ε_{2d} is the constant given by Theorem 1 for graphs of average degree at most $2d$.

To the best of our knowledge, this is the first result that breaks the $2^{n/2}$-barrier for counting perfect matchings in not necessarily bipartite graphs of bounded (average) degree.

When bipartite graphs are concerned, the classical algorithm of Ryser [12] has been improved for graphs of bounded average degree first by Servedio and Wan [13] and, very recently, by Izumi and Wadayama [9]. Our last result is the following theorem.

Theorem 3. *For any commutative ring R, given an $n \times n$ matrix M with elements from R with at most dn non-zero entries for some $d \geq 2$, one can compute the permanent of M using $\mathcal{O}^\star(2^{(1-1/(3.55d))n})$ time and performing $\mathcal{O}^\star(2^{(1-1/(3.55d))n})$ operations over the ring R. The algorithm may require to use exponential space and store an exponential number of elements from R.*

Note that the number of perfect matchings in a bipartite graph is equal to the permanent of the adjacency matrix of this graph (computed over $\mathbb{Z}$). Hence, we improve the running time of [9,13] in terms of the dependency on d. We would like to emphasise that our proof of Theorem 3 is elementary and does not need the advanced techniques of coding theory used in [9].

Organization of the paper Section 2 contains preliminaries. Next, in Section 3 we prove the main technical tool, that is Lemma 9, used in the proofs of Theorem 1 and Theorem 2. In Section 4 we prove Theorem 1, while in Section 5.1 we first show an inclusion-exclusion based algorithm for counting perfect matchings, which is later modified in Section 5.2 to fit the bounded average degree framework and prove Theorem 2. Finally, Section 6 contains a simple dynamic programming algorithm, proving Theorem 3.

We would like to note that both Section 5.1 and Section 6 are self-contained and do not rely on other sections (in particular do not depend on Lemma 9).

2 Preliminaries

We use standard (multi)graph notation. For a graph $G = (V, E)$ and a vertex $v \in V$ the neighbourhood of v is defined as $N_G(v) = \{u : uv \in E\} \setminus \{v\}$ and the closed neighbourhood of v as $N_G[v] = N_G(v) \cup \{v\}$. The degree of $v \in V$ is denoted $\deg_G(v)$ and equals the number of end-points of edges incident to v. In particular a self-loop contributes 2 to the degree of a vertex. We omit the subscript if the graph G is clear from the context. The average degree of an n-vertex graph $G = (V, E)$ is defined as $\frac{1}{n}\sum_{v \in V} \deg(v) = 2|E|/n$. A cycle cover in a multigraph $G = (V, E)$ is a subset of edges $C \subseteq E$, where each vertex is of degree exactly two if G is undirected or each vertex has exactly one outgoing and one ingoing arc, if G is directed. Note that this definition allows a cycle cover to contain cycles of length 1, i.e. self-loops, as well as taking two different parallel edges as a length 2 cycle (but does not allow using the same edge twice).

For a graph $G = (V, E)$ by $V_{\deg=c}, V_{\deg>c}, V_{\deg\geq c}$ let us denote the subsets of vertices of degree equal to c, greater than c and at least c respectively.

We also need the following well-known bounds.

Lemma 4. *For any $n, k \geq 1$ it holds that*

$$\binom{n}{k} \leq \left(\frac{en}{k}\right)^k.$$

Lemma 5. *For any $n \geq 1$, it holds that $H_{n-1} \geq \ln n$, where $H_n = \sum_{i=1}^{n} \frac{1}{i}$.*

Proof. It is well-known that $\lim_{n\to\infty} H_n - \ln n = \gamma$ where $\gamma > 0.577$ is the Euler-Mascheroni constant and the sequence $H_n - \ln n$ is decreasing. Therefore $H_{n-1} = H_n - \frac{1}{n} \geq \ln n + \gamma - \frac{1}{n}$, hence the lemma is proven for $n \geq 2$ as $\gamma > \frac{1}{2}$. For $n = 1$, note that $H_{n-1} = \ln n = 0$. □

3 Properties of Bounded Average Degree Graphs

This section contains technical results concerning bounded average degree graphs. In particular we prove Lemma 9, which is needed to get the claimed running times in Theorems 1 and 2. However, as the proofs of this section are not needed to understand the algorithms in further sections the reader may decide to see only Definition 8 and the statement of Lemma 9.

Lemma 6. *Given an n-vertex graph $G = (V, E)$ of average degree at most d and maximum degree at most D one can in polynomial time find a set A containing $\lceil \frac{n}{2+4dD} \rceil$ vertices of degree at most $2d$, where for each $x, y \in A$, $x \neq y$ we have $N_G[x] \cap N_G[y] = \emptyset$.*

Proof. Note that $|V_{\deg\leq 2d}| \geq n/2$. We apply the following procedure. Initially we set $A := \emptyset$ and all the vertices are unmarked. Next, as long as there exists an unmarked vertex x in $V_{\deg\leq 2d}$, we add x to A and mark all the vertices $N_G[N_G[x]]$. Since the set $N_G[N_G[x]]$ contains at most $1+2d+2d(D-1) = 1+2dD$ vertices, at the end of the process we have $|A| \geq \frac{n}{2+4dD}$. Clearly this routine can be implemented in polynomial time. □

Lemma 7. *For any $\alpha \geq 0$ and an n-vertex graph $G = (V, E)$ of average degree at most d there exists $D \leq e^\alpha$ such that $|V_{\deg>D}| \leq \frac{nd}{\alpha D}$.*

Proof. By standard counting arguments we have

$$\sum_{i=0}^{\infty} |V_{\deg>i}| = \sum_{i=0}^{\infty} i|V_{\deg=i}| \leq nd.$$

For the sake of contradiction assume that $|V_{\deg>i}| > \frac{nd}{\alpha i}$, for each $i \leq e^\alpha$. Then

$$\sum_{i=0}^{\infty} |V_{\deg>i}| \geq \sum_{i=1}^{\lfloor e^\alpha \rfloor} |V_{\deg>i}| > \frac{nd}{\alpha} \sum_{i=1}^{\lfloor e^\alpha \rfloor} 1/i = \frac{nd}{\alpha} H_{\lfloor e^\alpha \rfloor} \geq nd,$$

where the last inequality follows from Lemma 5. □

In the following definition we capture a superset of the sets used in the dynamic programming algorithms of Theorems 1 and 2.

Definition 8. *For an undirected graph $G = (V, E)$ and two vertices $s, t \in V$ by* $\deg2\text{sets}(G, s, t)$ *we define the set of all subsets $X \subseteq V \setminus \{s, t\}$, for which there exists a set of edges $F \subseteq E$ such that:*

- $\deg_F(v) = 0$ *for each* $v \in V \setminus (X \cup \{s, t\})$,
- $\deg_F(v) = 2$ *for each* $v \in X$,
- $\deg_F(v) \leq 1$ *for* $v \in \{s, t\}$.

Lemma 9. *For every $d \geq 1$ there exists a constant $\varepsilon_d > 0$, such that for an n-vertex graph $G = (V, E)$ of average degree at most d for any $s, t \in V$ the cardinality of* $\deg2\text{sets}(G, s, t)$ *is at most $\mathcal{O}^\star(2^{(1-\varepsilon_d)n})$.*

Proof. Use Lemma 7 with $\alpha = e^{cd}$ for some sufficiently large universal constant c (it suffices to take $c = 20$). Hence we can find an integer $D \leq e^\alpha = e^{e^{cd}}$ such that there are at most $\frac{nd}{\alpha D}$ vertices of degree greater than D in G.

Let $D' = \max(2d, D)$ and $H = G[V_{\deg \leq D'}]$. Moreover let $Y = V_{\deg > D'}$ and recall $|Y| \leq \frac{nd}{\alpha D}$, as $D' \geq D$ and $Y \subseteq V_{\deg > D}$. Note that H contains at least $n/2$ vertices and has average degree upper bounded by d. By Lemma 6 there exists a set $A \subseteq V(H)$ of $\lceil n/(4 + 8dD') \rceil$ vertices having disjoint closed neighbourhoods in H. Note that, since $d \geq 1$ and $D' \geq 2d$:

$$|A| = \left\lceil \frac{n}{4 + 8dD'} \right\rceil \geq \frac{n}{4 + 8dD'} \geq \frac{n}{2dD' + 8dD'} = \frac{n}{10dD'}. \tag{1}$$

If $n \leq \frac{8edD'}{4-e}$, $n = \mathcal{O}(1)$ and the claim is trivial. Otherwise:

$$|A| = \left\lceil \frac{n}{4 + 8dD'} \right\rceil < \frac{n}{8dD'} + 1 < \frac{n}{2edD'}. \tag{2}$$

Moreover, as $d \geq 1$ and $D' = \max(2d, D) \leq 2dD$, for sufficiently large c we have:

$$|Y| \leq \frac{nd}{\alpha D} \leq \frac{n}{20dD'} \cdot \frac{40d^3}{e^{cd}} < \frac{n}{20dD'} \leq \frac{|A|}{2}. \tag{3}$$

Consider an arbitrary set $X \in \deg2\text{sets}(G, s, t)$, and a corresponding set $F \subseteq E$ from Definition 8. Define Z_X as the set of vertices $x \in X \cap V(H)$ such that $N_H(x) \cap X = \emptyset$. Note that F is a set of paths and cycles, where each vertex of Z_X is of degree two, hence F contains at least $2|Z_X|$ edges between Z_X and Y, as any path/cycle of F visiting a vertex of Z_X has to enter from Y and leave to Y. Hence by the upper bound of 2 on the degrees in F we have $|Z_X| \leq |Y|$.

For each $x \in A \setminus (Z_X \cup \{s, t\})$ we have that $N_H[x] \cap X \neq \{x\}$ and $|N_H[x]| \leq 2d + 1$. By definition, if $x \in A \cap Z_X$, we have $N_H[x] \cap X = \{x\}$. Therefore, for fixed v and $A \cap Z_X$ there are at most

$$2^n \left(\frac{2^{2d+1} - 1}{2^{2d+1}}\right)^{|A \setminus (Z_X \cup \{s,t\})|} \left(\frac{1}{2}\right)^{|A \cap Z_X|} \leq 2^{n+2} \left(\frac{2^{2d+1} - 1}{2^{2d+1}}\right)^{|A|}$$

choices for $X \in \deg2\text{sets}(G, s)$.

Moreover, there are at most $\sum_{i=0}^{|Y|} \binom{|A|}{i} \leq n\binom{|A|}{|Y|}$ choices for $Z_X \cap A$. Thus

$$|\mathrm{deg2sets}(G,s,t)| \leq 2^{n+2} \cdot \left(\frac{2^{2d+1}-1}{2^{2d+1}}\right)^{|A|} \cdot n\binom{|A|}{|Y|}. \tag{4}$$

Let us now estimate $\binom{|A|}{|Y|}$ by Lemma 4. Since $D \leq D'$, $|Y| \leq \frac{nd}{\alpha D}$ and by (2) and (3):

$$\binom{|A|}{|Y|} \leq \left(\frac{e|A|}{|Y|}\right)^{|Y|} \leq \left(e\frac{n}{2edD'} \cdot \frac{\alpha D}{nd}\right)^{\frac{nd}{\alpha D}} \leq \left(\frac{\alpha}{2d^2}\right)^{\frac{nd}{\alpha D}} < \alpha^{\frac{nd}{\alpha D}}. \tag{5}$$

By the standard inequality $1 - x \leq e^{-x}$ we have that

$$(2^{2d+1}-1)/2^{2d+1} = (1 - 1/2^{2d+1}) \leq e^{-1/2^{2d+1}}. \tag{6}$$

Using (1), (5) and (6) we obtain that

$$\binom{|A|}{|Y|}\left(\frac{2^{2d+1}-1}{2^{2d+1}}\right)^{|A|/2} \leq \exp\left(\frac{nd\ln\alpha}{\alpha D} - \frac{n}{20dD'2^{2d+1}}\right).$$

Plugging in $\alpha = e^{cd}$ and using the fact that $e^{10d} > 40d^2$ for $d \geq 1$ we obtain:

$$\binom{|A|}{|Y|}\left(\frac{2^{2d+1}-1}{2^{2d+1}}\right)^{|A|/2} \leq \exp\left(\frac{ncd}{e^{(c-10)d}20d \cdot 2dD} - \frac{n}{20dD'2^{2d+1}}\right).$$

Since $D' = \max(2d, D) \leq 2dD$ and $e^{5d} > d2^{2d+1}$ as $d \geq 1$, we get

$$\binom{|A|}{|Y|}\left(\frac{2^{2d+1}-1}{2^{2d+1}}\right)^{|A|/2} \leq \exp\left(\frac{n}{20dD'2^{2d+1}}\left(\frac{c}{e^{(c-15)d}} - 1\right)\right).$$

Finally, for sufficiently large c, as $d \geq 1$, we have $c < e^{(c-15)d}$ and

$$\binom{|A|}{|Y|}\left(\frac{2^{2d+1}-1}{2^{2d+1}}\right)^{|A|/2} < 1. \tag{7}$$

Consequently, plugging (7) into (4) and using (1) and (6) we obtain:

$$\begin{aligned}|\mathrm{deg2sets}(G,s,t)| &< n2^{n+2}\left(\frac{2^{2d+1}-1}{2^{2d+1}}\right)^{|A|/2} \\ &\leq n2^{n+2}\exp\left(-\frac{n}{2^{2d+1}\cdot 20dD'}\right) \\ &\leq n2^{n+2}\exp\left(-\frac{n}{2^{2d+1}\cdot 20d\cdot e^{e^{cd}}}\right).\end{aligned}$$

This concludes the proof of the lemma. Note that the dependency on d in the final constant ε_d is doubly-exponential. □

4 Algorithm for TSP

To prove Theorem 1, it suffices to solve in $\mathcal{O}^\star(2^{(1-\varepsilon_d)n})$ time the following problem. We are given an undirected n-vertex graph $G = (V, E)$ of average degree at most d, vertices $a, b \in V$ and a weight function $c : E \to \mathbb{R}_+$. We are to find the cheapest Hamiltonian path between a and b in G, or verify that no Hamiltonian ab-path exists.

We solve the problem by the standard dynamic programing approach. That is for each $a \in X \subseteq V$ and $v \in X$ we compute $t[X][v]$, which is the cost of the cheapest path from a to v with the vertex set X. The entry $t[V][b]$ is the answer to our problem. Note that it is enough to consider only such pairs (X, v), for which there exists an av-path with the vertex set X.

We first set $t[\{a\}][a] = 0$. Then iteratively, for each $i = 1, 2, \ldots, n-1$, for each $u \in V$, for each $X \subseteq V$ such that $|X| = i$, $a, u \in X$ and $t[X][u]$ is defined, for each edge $uv \in E$ where $v \notin X$, if $t[X \cup \{v\}][v]$ is undefined or $t[X \cup \{v\}][v] > t[X][u] + c(uv)$, we set $t[X \cup \{v\}][v] = t[X][u] + c(uv)$.

Finally, note that if $t[X][v]$ is defined then $X \setminus \{a, v\} \in \mathrm{deg2sets}(G, a, v)$. Hence, the complexity of the above algorithm is within a polynomial factor from $\sum_{v \in V} |\mathrm{deg2sets}(G, a, v)|$, which is bounded by $\mathcal{O}^\star(2^{(1-\varepsilon_d)n})$ by Lemma 9.

5 Counting Perfect Matchings

In this section we design algorithms counting the number of perfect matchings in a given graph. First, in Section 5.1 we show an inclusion-exclusion based algorithm, which given an n-vertex graph computes the number of its perfect matchings in $\mathcal{O}^\star(2^{n/2})$ time and polynomial space. This matches the time and space bounds of the algorithm of Björklund [3]. Next, in Section 5.2 we show how the algorithm from Section 5.1 can be reformulated as a dynamic programming routine (using exponential space), which together with Lemma 9 will imply the running time claimed in Theorem 2.

5.1 Inclusion-Exclusion Based Algorithm

In the following theorem we show an algorithm computing the number of perfect matchings of an undirected graph in $\mathcal{O}^\star(2^{n/2})$ time and polynomial space, thus matching the time and space complexity of the algorithm by Björklund [3].

Theorem 10. *Given an n-vertex graph $G = (V, E)$ in $\mathcal{O}^\star(2^{n/2})$ time and polynomial space one can count the number of perfect matchings in G.*

Proof. Clearly we can assume that n is even. Consider the edges of G being black and let $V = \{v_0, \ldots, v_{n-1}\}$. Now we add to the graph a perfect matching of red edges $E_R = \{v_{2i}v_{2i+1} : 0 \leq i < n/2\}$ obtaining a multigraph G'. Denote $e_i = v_{2i}v_{2i+1} \in E_R$.

We say that a cycle or a walk in G' is *alternating*, if, when we traverse the cycle (walk), the colours of the edges alternate; in particular, an alternating cycle or

closed walk is of even length. Observe that for any perfect matching $M \subseteq E$ the multiset $M \cup E_R$ is a cycle cover (potentially with 2-cycles), where all the cycles are *alternating*. Moreover, for any cycle cover Y of G' composed of alternating cycles the set $Y \setminus E_R$ is a perfect matching in G. This leads us to the following observation.

Observation 11. *The number of perfect matchings in G equals the number of cycle covers in G' where each cycle is alternating.*

We are going to compute the number of cycle covers of G' with alternating cycles using the inclusion-exclusion principle over the set of red edges E_R.

For an edge $e_i \in E_R$, we say that a closed walk C is e_i-*nice* if it is alternating, traverses e_i exactly once and does not traverse any edge $e_j \in E_R$ for $j < i$. Note that for an alternating walk C, if C contains a vertex v_{2j} or v_{2j+1}, it needs to traverse the edge $e_j \in E_R$, as this is the only red edge incident to v_{2j} and v_{2j+1}. A closed walk is *nice* if it is e_i-nice for some $e_i \in E_R$; note that, in this case, the edge e_i is defined uniquely. For a positive integer r let us define the universe Ω_r as the set of r-tuples, where each of the r coordinates contains a nice closed walk in G' and the total length of all the walks equals n. For $0 \leq i < n/2$ let $A_{r,i} \subseteq \Omega_r$ be the set of r-tuples, where at least one walk contains the arc e_i. Note that by the observations we made so far the number of perfect matchings in G equals $\sum_{1 \leq r \leq n/2} |\bigcap_{0 \leq i < n/2} A_{r,i}|/r!$, as the tuples in Ω_r are ordered and in any tuple of $\bigcap_{0 \leq i < n/2} A_{r,i}$ all walks are pairwise different. Therefore from now on we assume r to be fixed. By the inclusion-exclusion principle

$$\left| \bigcap_{0 \leq i < n/2} A_{r,i} \right| = \sum_{I \subseteq \{0,\ldots,n/2-1\}} (-1)^{|I|} \left| \bigcap_{i \in I} (\Omega_r \setminus A_{r,i}) \right| ,$$

where we define $\bigcap_{i \in I} (\Omega_r \setminus A_{r,i})$ for $I = \emptyset$ as Ω_r. Hence to prove the theorem it is enough to compute the value $|\bigcap_{i \in I} (\Omega_r \setminus A_{r,i})|$ for a given $I \subseteq \{0, \ldots, n/2-1\}$ in polynomial time. Let G'_I be the graph G' with all the endpoints of edges e_i for $i \in I$ removed. Let $p_{a,q}$ be the number of e_a-nice closed walks in G'_I of length q. Note that the value $p_{a,q}$ can be computed in polynomial time by standard dynamic programming algorithm, filling in a table $t_p[b][\hat{q}]$, $a \leq b < n/2, 0 \leq \hat{q} < q$, where $t_p[b][\hat{q}]$ is the number of alternating walks W of length $\hat{q}$ in G'_I with the first edge e_a, the last edge e_b, which visit e_a only once and does not visit any edge e_c for $c < a$.

Finally, having the values $p_{a,q}$ is enough to compute $|\bigcap_{i \in I} (\Omega_r \setminus A_i)|$ by the standard knapsack type dynamic programming. That is, we fill in a table $t[\hat{r}][q]$, $0 \leq \hat{r} \leq r$, $0 \leq q \leq n/2$, where $t[\hat{r}][q]$ is the number of $\hat{r}$-tuples of nice closed walks in G'_I of total length q. □

5.2 Dynamic Programming Based Algorithm

To prove Theorem 2 we want to reformulate the algorithm from Section 5.1, to use dynamic programming instead of the inclusion exclusion principle. This

causes the space complexity to be exponential, however it will allow us to use Lemma 9 to obtain an improved running time for bounded average degree graphs.

Assume that we are given an n-vertex undirected graph $G = (V, E)$, where n is even, and we are to count the number of perfect matchings in G. We are going to construct an undirected multigraph G' having only $n/2$ vertices, where the edges of G' will be labeled with unordered pairs of vertices of G, i.e. with edges of G. As the set of vertices of $G' = (V', E')$ we take $V' = \{v'_0, \ldots, v'_{n/2-1}\}$. For each edge $v_a v_b$ of G we add to G' exactly one edge: $v'_{\lfloor a/2 \rfloor} v'_{\lfloor b/2 \rfloor}$ labeled with $\{v_a, v_b\}$. For an edge $e' \in E'$ by $\ell(e')$ let us denote the label of e'. Note that G' may contain self-loops and parallel edges. Observe that if the graph G is of average degree d, then the graph G' is of average degree $2d$.

In what follows we count the number of particular cycle covers of G', where we use the labels of edges to make sure that a cycle going through a vertex $v'_i \in V'$ never uses two edges of G' corresponding to two edges of G incident to the same vertex.

Lemma 12. *The number of perfect matchings in G equals the number of cycle covers $C \subseteq E'$ of G', where $\bigcup_{e \in C} \ell(e) = V$.*

Proof. We show a bijection between perfect matchings in G and cycle covers C of G' satisfying the condition $\bigcup_{e \in C} \ell(e) = V$.

Let M be a perfect matching in G. As $f(M)$ we define $f(M) = \{v'_{\lfloor a/2 \rfloor} v'_{\lfloor b/2 \rfloor} : v_a v_b \in M\}$. Note that $f(M)$ is a cycle cover and moreover $\bigcup_{e \in f(M)} \ell(e) = V$. In the reverse direction, for a cycle cover $C \subseteq E'$ of G', consider a set of edges $h(C)$ defined as $h(C) = \{\ell(e) : e \in C\}$. Clearly the condition $\bigcup_{e \in C} \ell(e) = V$ implies that $h(C)$ is a perfect matching, and moreover $h = f^{-1}$. □

Observe, that if a cycle cover $C \subseteq E'$ of G' does not satisfy $\bigcup_{e \in C} \ell(e) = V$, then there is a vertex $v'_i \in V'$, such that the two edges of C incident to v'_i do not have disjoint labels. Intuitively this means we are able to verify the condition $\bigcup_{e \in C} \ell(e) = V$ locally, which is enough to derive the following dynamic programming routine.

Lemma 13. *One can compute the number of cycle covers C of G' satisfying $\bigcup_{e \in C} \ell(e) = V$ in $\mathcal{O}^\star(\sum_{s,t \in V} |\mathrm{deg2sets}(G', s, t)|)$ time and space.*

Proof. An *ordered r-cycle cover* of a graph H is a tuple of r cycles in H, whose union is a cycle cover of H. As each cycle cover of H that contains exactly r cycles can be ordered into exactly $r!$ different ordered r-cycle covers, it is sufficient to count, for any $1 \leq r \leq n/2$, the number of ordered r-cycle covers C in G' such that each two edges in C have disjoint labels. In the rest of the proof, we focus on one fixed value of r.

For $0 \leq q \leq r$ and $X \subseteq V'$ as $t[q][X]$ let us define the number of ordered q-cycle covers in $G'[X]$ where each two edges have disjoint labels; note that $t[r][V']$ is exactly the value we need. Moreover for $0 \leq q < r$, $X \subseteq V'$, $v'_a, v'_b \in X$, $a < b$ and $x \in \{v_{2b}, v_{2b+1}\}$ as $t_2[q][X][v'_a][v'_b][x]$ we define the number of pairs (C, P) where

- C is an ordered q-cycle cover of $G'[Y]$ for some $Y \subseteq X \setminus \{v'_a, v'_b\}$;
- P is a $v'_a v'_b$-path with the vertex set $X \setminus Y$ that does not contain any vertex v'_c with $c < a$;
- any two edges of $C \cup P$ have disjoint labels;
- the label of the edge of P incident to v'_a contains v_{2a};
- the label of the edge of P incident to v'_b contains x.

Note that we have the following border values: $t[0][\emptyset] = 1$ and $t[0][X] = 0$ for $X \neq \emptyset$.

Consider an entry $t_2[q][X][v'_a][v'_b][x]$, and let (C, P) be one of the pairs counted in it. We have two cases: either P is of length 1 or longer. The number of pairs (C, P) in the first case equals $t[q][X \setminus \{v'_a, v'_b\}] \cdot |\{v'_a v'_b \in E' : \ell(v'_a v'_b) = \{v_{2a}, x\}\}|$. In the second case, let $v'_c v'_b$ be the last edge of P; note that $c > a$ by the assumptions on P. The label of $v'_c v'_b$ equals $\{v_{2c}, x\}$ or $\{v_{2c+1}, x\}$. Thus, the number of elements (C, P) in the second case equals $\sum_{v'_c \in X \setminus \{v'_a, v'_b\}} \sum_{y \in \{v_{2c}, v_{2c+1}\}} t_2[q][X \setminus \{v'_b\}][v'_a][v'_c][y \oplus 1] \cdot |\{v'_c v'_b \in E' : \ell(v'_c v'_b) = \{y, x\}\}|$, where for $y = v'_r$ we define $y \oplus 1 = v'_{r \oplus 1}$.

Let us now move to the entry $t[q][X]$ and let C be an ordered q-cycle cover in $G'[X]$. Again, there are two cases: either the last cycle of C (henceforth denoted W) is of length 1 or longer. The number of the elements C of the first type equals $\sum_{v'_a \in X} t[q-1][X \setminus \{v'_a\}] \cdot |\{v'_a v'_a \in E'\}|$. In the second case, let v'_a be the lowest-numbered vertex on W and let $e = v'_a v'_b$ be the edge of W where $v_{2a+1} \in \ell(e)$. Note that both v'_a and e are defined uniquely; moreover, $a < b$ and no vertex v'_c with $c < a$ belongs to W. Thus the number of elements C of the second type equals $\sum_{v'_a, v'_b \in X, a<b} \sum_{x \in \{v_{2b}, v_{2b+1}\}} t_2[q-1][X][v'_a][v'_b][x \oplus 1] \cdot |\{v'_a v'_b \in E' : \ell(v'_a v'_b) = \{v_{2a+1}, x\}\}|$.

So far we have given recursive formulas, that allow computing the entries of the tables t and t_2. However the values $t[q][X]$, $t_2[q][X][v'_a][v'_b][x]$ for $X \notin \bigcup_{s,t \in V'} \mathrm{deg2sets}(G', s, t)$ are equal to zero. The last step of the proof is to show how to perform the dynamic programming computation in a time complexity within a polynomial factor from the number of non-zero entries of the table. We do that in a bottom-up manner, that is iteratively, for each $q = 1, 2, \ldots, r$, for each $i = 1, 2, \ldots, n$, we want to compute the values of non-zero entries $t[q][X]$ for all sets X of cardinality i and then compute the values of non-zero entries $t_2[q][X][*][*][*]$ for all sets X of cardinality i. Having the non-zero entries for the pairs (q', i') where $q' < q$, $i' \leq i$ one can compute the list of non-zero entries $t[q][X]$ for $|X| = i$ by investigating to which recursive formulas the non-zero entries for (q', i') contribute to. Analogously having the non-zero entries for the pairs (q', i') where $q' \leq q$, $i' < i$ we generate the non-zero entries $t_2[q][X][*][*][*]$ for $|X| = i$, which finishes the proof of the lemma. □

Theorem 2 follows directly from the Lemma 9 together with Lemma 13.

6 Counting Perfect Matchings in Bipartite Graphs

In this section we prove Theorem 3. Let R be an arbitrary commutative ring, and $A = (a_{i,j})_{1 \leq i,j \leq n}$ be an $n \times n$ matrix with elements from R with at most dn non-zero elements, for some $d \geq 2$. We are to compute the permanent of A.

Let $C \subseteq \{1, 2, \ldots, n\}$ be the set of columns of A containing $\lfloor k/(\alpha d) \rfloor$ columns with the smallest number of non-zero entries, where $\alpha \geq 2$ is a constant to be determined later. Let $R = \{1 \leq i \leq n : \exists_{j \in C} : a_{i,j} \neq 0\}$. Observe that $|R| \leq k/\alpha$, as the average number of non-zero entries in the columns of C is at most d. Without loss of generality assume that $R = \{n - |R| + 1, n - |R| + 2, \ldots, n\}$, that is, we order the rows of A such that the rows of R appear at the bottom of the matrix. In particular for any $1 \leq i \leq k(1 - 1/\alpha)$ and $j \in C$ we have $a_{i,j} = 0$.

Consider the following standard dynamic programming approach. For a subset X of columns of A define $t[X]$ as the permanent of $(a_{i,j})_{1 \leq i \leq |X|, j \in X}$, a $|X| \times |X|$ submatrix of A. Note that we are to compute $t[\{1, 2, \ldots, n\}]$. Observe that the following recursive formula allows to compute the entries of the table t, where we sum over the element used in the permanent computation in the $|X|$-th row of A:

$$t[X] = \sum_{j \in X} a_{|X|,j} t[X \setminus \{j\}] \,,$$

where $t[\emptyset]$ is defined as 1.

Let us upper bound the number of sets X, for which $t[X]$ is non-zero. If $1 \leq |X| \leq (1 - 1/\alpha)k$ and $t[X] \neq 0$, then $X \cap C = \emptyset$, as $a_{i,j} = 0$ for $j \in C$ and $i \leq (1 - 1/\alpha)k$. Consequently there are at most $2^{k - \lfloor k/(\alpha d) \rfloor} \leq 2^{1 + (1 - 1/(\alpha d))k}$ sets X with $t[X] \neq 0$ of cardinality at most $(1 - 1/\alpha)k$. At the same time there are at most $k\binom{k}{\lceil k/\alpha \rceil}$ sets of cardinality greater than $(1 - 1/\alpha)k$. By using the binary entropy function, we get $\binom{k}{\lceil k/\alpha \rceil} = \mathcal{O}^\star(2^{H(1/\alpha)k})$, where $H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$. For $d \geq 2$ and $\alpha = 3.55$ we have $2^{H(1/\alpha)} \leq 2^{1 - 1/(\alpha d)}$. Consequently if we skip the computation of values $t[X]$ for sets X of cardinality at most $(1 - 1/\alpha)k$, such that $X \cap C \neq \emptyset$, we obtain the claimed running time, which finishes the proof of Theorem 3.

Note that the constant $\alpha = 3.55$ can be improved if we have a stronger lower bound on d. However, in our analysis it is crucial that $\alpha > 2$.

7 Conclusions and Open Problems

We would like to conclude with two open problems that arise from our work. First, can our ideas be applied to obtain an $\mathcal{O}^\star(2^{(1-\varepsilon)n})$ time algorithm for computing the chromatic number of graphs of bounded average degree? For graphs of bounded maximum degree such an algorithm is due to Björklund et al. [4].

Second, can we make a similar improvements as in our work if only polynomial space is allowed? To the best of our knowledge, this question remains open even in graphs of bounded maximum degree.

Acknowledgements. We would like to thank anonymous referees for their helpful remarks and comments.

References

1. Bellman, R.: Dynamic programming treatment of the travelling salesman problem. J. ACM 9, 61–63 (1962)
2. Björklund, A.: Determinant Sums for Undirected Hamiltonicity. In: FOCS, pp. 173–182. IEEE Computer Society (2010)
3. Björklund, A.: Counting perfect matchings as fast as Ryser. In: Rabani, Y. (ed.) SODA, pp. 914–921. SIAM (2012)
4. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Trimmed Moebius inversion and graphs of bounded degree. Theory Comput. Syst. 47(3), 637–654 (2010)
5. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: The traveling salesman problem in bounded degree graphs. ACM Transactions on Algorithms 8(2), 18 (2012)
6. Cygan, M., Dell, H., Lokshtanov, D., Marx, D., Nederlof, J., Okamoto, Y., Paturi, R., Saurabh, S., Wahlström, M.: On problems as hard as CNF-SAT. In: IEEE Conference on Computational Complexity, pp. 74–84. IEEE (2012)
7. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. J. Soc. Ind. Appl. Math. 10, 196–210 (1962)
8. Impagliazzo, R., Paturi, R.: On the Complexity of k-SAT. J. Comput. Syst. Sci. 62(2), 367–375 (2001)
9. Izumi, T., Wadayama, T.: A new direction for counting perfect matchings. In: FOCS, pp. 591–598. IEEE Computer Society (2012)
10. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs on bounded treewidth are probably optimal. In: Randall, D. (ed.) SODA, pp. 777–789. SIAM (2011)
11. Patrascu, M., Williams, R.: On the Possibility of Faster SAT Algorithms. In: SODA, pp. 1065–1075 (2010)
12. Ryser, H.: Combinatorial Mathematics. The Carus mathematical monographs. Mathematical Association of America (1963)
13. Servedio, R.A., Wan, A.: Computing sparse permanents faster. Inf. Process. Lett. 96(3), 89–92 (2005)
14. Woeginger, G.J.: Exact algorithms for NP-hard problems: A survey. In: Jünger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization - Eureka, You Shrink! LNCS, vol. 2570, pp. 185–207. Springer, Heidelberg (2003)

A Robust Khintchine Inequality, and Algorithms for Computing Optimal Constants in Fourier Analysis and High-Dimensional Geometry*

Anindya De[1,**], Ilias Diakonikolas[2,***], and Rocco Servedio[3,†]

[1] UC Berkeley
anindya@cs.berkeley.edu
[2] University of Edinburgh
ilias.d@ed.ac.uk
[3] Columbia University
rocco@cs.columbia.edu

Abstract. This paper makes two contributions towards determining some well-studied optimal constants in Fourier analysis of Boolean functions and high-dimensional geometry.

1. It has been known since 1994 [GL94] that every linear threshold function has squared Fourier mass at least $1/2$ on its degree-0 and degree-1 coefficients. Denote the minimum such Fourier mass by $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$, where the minimum is taken over all n-variable linear threshold functions and all $n \geq 0$. Benjamini, Kalai and Schramm [BKS99] have conjectured that the true value of $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$ is $2/\pi$. We make progress on this conjecture by proving that $\mathbf{W}^{\leq 1}[\mathbf{LTF}] \geq 1/2 + c$ for some absolute constant $c > 0$. The key ingredient in our proof is a "robust" version of the well-known Khintchine inequality in functional analysis, which we believe may be of independent interest.
2. We give an algorithm with the following property: given any $\eta > 0$, the algorithm runs in time $2^{\mathrm{poly}(1/\eta)}$ and determines the value of $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$ up to an additive error of $\pm\eta$. We give a similar $2^{\mathrm{poly}(1/\eta)}$-time algorithm to determine *Tomaszewski's constant* to within an additive error of $\pm\eta$; this is the minimum (over all origin-centered hyperplanes H) fraction of points in $\{-1, 1\}^n$ that lie within Euclidean distance 1 of H. Tomaszewski's constant is conjectured to be $1/2$; lower bounds on it have been given by Holzman and Kleitman [HK92] and independently by Ben-Tal, Nemirovski and Roos [BTNR02]. Our algorithms combine tools from anti-concentration of sums of independent random variables, Fourier analysis, and Hermite analysis of linear threshold functions.

1 Introduction

This paper is inspired by a belief that simple mathematical objects should be well understood. We study two closely related kinds of simple objects: n-dimensional linear

* A full version of this paper may be found at http://arxiv.org/abs/1207.2229
** Research supported by NSF award CCF-1118083.
*** Research supported by a Simons Postdoctoral Fellowship.
† Research supported in part by NSF awards CCF-0915929 and CCF-1115703.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 376–387, 2013.

threshold functions $f(x) = \mathrm{sign}(w \cdot x - \theta)$, and n-dimensional origin-centered hyperplanes $H = \{x \in \mathbb{R}^n : w \cdot x = 0\}$. Benjamini, Kalai and Schramm [BKS99] and Tomaszewski [Guy86] have posed the question of determining two universal constants related to halfspaces and origin-centered hyperplanes respectively; we refer to these quantities as "the BKS constant" and "Tomaszewski's constant." While these constants arise in various contexts including uniform-distribution learning and optimization theory, little progress has been made on determining their actual values over the past twenty years. In both cases there is an easy upper bound which is conjectured to be the correct value; Gotsman and Linial [GL94] gave the best previously known lower bound on the BKS constant in 1994, and Holzmann and Kleitman [HK92] gave the best known lower bound on Tomaszewski's constant in 1992.

We give two main results. The first of these is an improved lower bound on the BKS constant; a key ingredient in the proof is a "robust" version of the well-known Khintchine inequality, which we believe may be of independent interest. Our second main result is a pair of algorithms for computing the BKS constant and Tomaszewski's constant up to any prescribed accuracy. The first algorithm, given any $\eta > 0$, runs in time $2^{\mathrm{poly}(1/\eta)}$ and computes the BKS constant up to an additive η, and the second algorithm runs in time $2^{\mathrm{poly}(1/\eta)}$ and has the same guarantee for Tomaszewski's constant.

1.1 Background and Problem Statements

First Problem: Low-Degree Fourier Weight of Linear Threshold Functions. A *linear threshold function*, henceforth denoted simply LTF, is a function $f : \{-1,1\}^n \to \{-1,1\}$ of the form $f(x) = \mathrm{sign}(w \cdot x - \theta)$ where $w \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ (the univariate function $\mathrm{sign} : \mathbb{R} \to \mathbb{R}$ is $\mathrm{sign}(z) = 1$ for $z \geq 0$ and $\mathrm{sign}(z) = -1$ for $z < 0$). The values $w_1, \ldots, w_n$ are the *weights* and θ is the *threshold.* Linear threshold functions play a central role in many areas of computer science such as concrete complexity theory and machine learning, see e.g. [DGJ$^+$10] and the references therein.

It is well known [BKS99, Per04] that LTFs are highly noise-stable, and hence they must have a large amount of Fourier weight at low degrees. For $f : \{-1,1\}^n \to \mathbb{R}$ and $k \in [0,n]$ let us define $\mathbf{W}^k[f] = \sum_{S \subseteq [n], |S|=k} \widehat{f}^2(S)$ and $\mathbf{W}^{\leq k}[f] = \sum_{j=0}^{k} \mathbf{W}^j[f]$; we will be particularly interested in the Fourier weight of LTFs at levels 0 and 1. More precisely, for $n \in \mathbf{N}$ let $\mathbf{LTF}_n$ denote the set of all n-dimensional LTFs, and let $\mathbf{LTF} = \cup_{n=1}^{\infty} \mathbf{LTF}_n$. We define the following universal constant:

Definition 1. *Let* $\mathbf{W}^{\leq 1}[\mathbf{LTF}] \stackrel{def}{=} \inf_{h \in \mathbf{LTF}} \mathbf{W}^{\leq 1}(h) = \inf_{n \in \mathbf{N}} \mathbf{W}^{\leq 1}[\mathbf{LTF}_n]$, *where* $\mathbf{W}^{\leq 1}[\mathbf{LTF}_n] \stackrel{def}{=} \inf_{h \in \mathbf{LTF}_n} \mathbf{W}^{\leq 1}(h)$.

Benjamini, Kalai and Schramm (see Remark 3.7 of [BKS99]) and subsequently O'Donnell (see the Conjecture following Theorem 2 of Section 5.1 of [O'D12]) have conjectured that $\mathbf{W}^{\leq 1}[\mathbf{LTF}] = 2/\pi$, and hence we will sometimes refer to $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$ as "the BKS constant." As $n \to \infty$, a standard analysis of the n-variable Majority function shows that $\mathbf{W}^{\leq 1}[\mathbf{LTF}] \leq 2/\pi$. Gotsman and Linial [GL94] observed that $\mathbf{W}^{\leq 1}[\mathbf{LTF}] \geq 1/2$ but until now no better lower bound was known. We note that since the universal constant $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$ is obtained by taking the infimum over an infinite set, it is not *a priori* clear whether the computational problem of computing or even approximating $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$ is decidable.

Jackson [Jac06] has shown that improved lower bounds on $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$ translate directly into improved noise-tolerance bounds for agnostic weak learning of LTFs in the "Restricted Focus of Attention" model of Ben-David and Dichterman [BDD98]. Further motivation for studying $\mathbf{W}^{\leq 1}[f]$ comes from the fact that $\mathbf{W}^{1}[f]$ is closely related to the noise stability of f (see [O'D12]). In particular, if $\mathbf{NS}_\rho[f]$ represents the noise stability of f when the noise rate is $(1-\rho)/2$, then it is known that $\frac{d\mathbf{NS}_\rho[f]}{d\rho}\Big|_{\rho=0} = \mathbf{W}^{1}[f]$. This means that for a function f with $\mathbf{E}[f] = 0$, we have $\mathbf{NS}_\rho[f] \to \rho \cdot \mathbf{W}^{\leq 1}[f]$ as $\rho \to 0$. Thus, at very large noise rates, $\mathbf{W}^{1}[f]$ quantifies the size of the "noisy boundary" of mean-zero functions f.

Second Problem: How Many Hypercube Points Have Distance at Most 1 from an Origin-Centered Hyperplane? For $n \in \mathbb{N}$ and $n > 1$, let $\mathbb{S}^{n-1}$ denote the n-dimensional sphere $\mathbb{S}^{n-1} = \{w \in \mathbb{R}^n : \|w\|_2 = 1\}$, and let $\mathbb{S} = \cup_{n>1}\mathbb{S}^{n-1}$. Each unit vector $w \in \mathbb{S}^{n-1}$ defines an origin-centered hyperplane $H_w = \{x \in \mathbb{R}^n : w \cdot x = 0\}$. Given a unit vector $w \in \mathbb{S}^{n-1}$, we define $\mathbf{T}(w) \in [0,1]$ to be $\mathbf{T}(w) = \mathbf{Pr}_{x \in \{-1,1\}^n}[|w \cdot x| \leq 1]$, the fraction of hypercube points in $\{-1,1\}^n$ that lie within Euclidean distance 1 of the hyperplane H_w. We define the following universal constant, which we call "Tomaszewski's constant:"

Definition 2. *Define* $\mathbf{T}(\mathbb{S}) \stackrel{def}{=} \inf_{w \in \mathbb{S}} \mathbf{T}(w) = \inf_{n \in \mathbb{N}} \mathbf{T}(\mathbb{S}^{n-1})$, *where* $\mathbf{T}(\mathbb{S}^{n-1}) \stackrel{def}{=} \inf_{w \in \mathbb{S}^{n-1}} \mathbf{T}(w)$.

Tomaszewski [Guy86] has conjectured that $\mathbf{T}(\mathbb{S}) = 1/2$. The main result of Holzman and Kleitman [HK92] is a proof that $3/8 \leq \mathbf{T}(\mathbb{S})$; the upper bound $\mathbf{T}(\mathbb{S}) \leq 1/2$ is witnessed by the vector $w = (1/\sqrt{2}, 1/\sqrt{2})$. As noted in [HK92], the quantity $\mathbf{T}(\mathbb{S})$ has a number of appealing geometric and probabilistic reformulations. Similar to the BKS constant, since $\mathbf{T}(\mathbb{S})$ is obtained by taking the infimum over an infinite set, it is not immediately evident that any algorithm can compute or approximate $\mathbf{T}(\mathbb{S})$.[1]

An interesting quantity in its own right, Tomaszewski's constant also arises in a range of contexts in optimization theory, see e.g. [So09, BTNR02]. In fact, the latter paper proves a lower bound of $1/3$ on the value of Tomaszewski's constant independently of [HK92], and independently conjectures that the optimal lower bound is $1/2$.

1.2 Our Results

A Better Lower Bound for the BKS Constant $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$. Our first main result is the following theorem:

Theorem 1 (Lower Bound for the BKS constant). *There exists a universal constant $c' > 0$ such that $\mathbf{W}^{\leq 1}[\mathbf{LTF}] \geq \frac{1}{2} + c'$.*

This is the first improvement on the [GL94] lower bound of $1/2$ since 1994. We actually give two quite different proofs of this theorem, which are sketched in the "Techniques" subsection below.

[1] Whenever we speak of "an algorithm to compute or approximate" one of these constants, of course what we really mean is an algorithm that outputs the desired value *together with a proof of correctness of its output value*.

An Algorithm for Approximating the BKS Constant $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$. Our next main result shows that in fact there *is* a finite-time algorithm that approximates the BKS constant up to any desired accuracy:

Theorem 2 (Approximating the BKS constant). *There is an algorithm that, on input an accuracy parameter $\epsilon > 0$, runs in time $2^{\mathrm{poly}(1/\epsilon)}$ and outputs a value Γ_ϵ such that*

$$\mathbf{W}^{\leq 1}[\mathbf{LTF}] \leq \Gamma_\epsilon \leq \mathbf{W}^{\leq 1}[\mathbf{LTF}] + \epsilon. \tag{1}$$

An Algorithm for Approximating Tomaszewski's Constant $\mathbf{T}(\mathbb{S})$. Our final main result is an algorithm that approximates $\mathbf{T}(\mathbb{S})$ up to any desired accuracy:

Theorem 3 (Approximating Tomaszewski's constant). *There is an algorithm that, on input $\epsilon > 0$, runs in time $2^{\mathrm{poly}(1/\epsilon)}$ and outputs a value Γ_ϵ such that*

$$\mathbf{T}(\mathbb{S}) \leq \Gamma_\epsilon \leq \mathbf{T}(\mathbb{S}) + \epsilon. \tag{2}$$

1.3 Our Techniques for Theorem 1: Lower-Bounding the BKS Constant

It is easy to show that it suffices to consider the level-1 Fourier weight $\mathbf{W}^1$ of LTFs that have threshold $\theta = 0$ and have $w \cdot x \neq 0$ for all $x \in \{-1,1\}^n$, so we confine our discussion to such zero-threshold LTFs. To explain our approaches to lower bounding $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$, we recall the essentials of the simple argument of [GL94] that gives a lower bound of $1/2$. The key ingredient of their argument is the well-known Khintchine inequality from functional analysis:

Definition 3. *For a unit vector $w \in \mathbb{S}^{n-1}$ we define $\mathbf{K}(w) \stackrel{def}{=} \mathbf{E}_{x \in \{-1,1\}^n}[|w \cdot x|]$ to be the "Khintchine constant for w."*

The following is a classical theorem in functional analysis (we write e_i to denote the unit vector in $\mathbb{R}^n$ with a 1 in coordinate i):

Theorem 4 (Khintchine inequality, [Sza76]). *For $w \in \mathbb{S}^{n-1}$ any unit vector, we have $\mathbf{K}(w) \geq 1/\sqrt{2}$, with equality holding if and only if $w = \frac{1}{\sqrt{2}}(\pm e_i \pm e_j)$ for some $i \neq j \in [n]$.*

Szarek [Sza76] was the first to obtain the optimal constant $1/\sqrt{2}$, and subsequently several simplifications of his proof were given [Haa82, Tom87, LO94]; we shall give a simple self-contained proof in Section 2.1 below, which is quite similar to Filmus's [Fil12] translation of the [LO94] proof into "Fourier language." With Theorem 4 in hand, the Gotsman-Linial lower bound is almost immediate:

Proposition 1 ([GL94]). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be a zero-threshold LTF $f(x) = \mathrm{sign}(w \cdot x)$ where $w \in \mathbb{R}^n$ has $\|w\|_2 = 1$. Then $\mathbf{W}^1[f] \geq (\mathbf{K}(w))^2$.*

Proof. We have that $\mathbf{K}(w) = \mathbf{E}_x[f(x)(w \cdot x)] = \sum_{i=1}^n \widehat{f}(i) w_i \leq \sqrt{\sum_{i=1}^n \widehat{f}^2(i)} \cdot \sqrt{\sum_{i=1}^n w_i^2} = \sqrt{\mathbf{W}^1[f]}$ where the first equality uses the definition of f, the second is Plancherel's identity, the inequality is Cauchy-Schwarz, and the last equality uses the assumption that w is a unit vector. □

First Proof of Theorem 1: A "Robust" Khintchine Inequality. Given the strict condition required for equality in the Khintchine inequality, it is natural to expect that if a unit vector $w \in \mathbb{R}^n$ is "far" from $\frac{1}{\sqrt{2}}(\pm e_i \pm e_j)$, then $\mathbf{K}(w)$ should be significantly larger than $1/\sqrt{2}$. We prove a robust version of the Khintchine inequality which makes this intuition precise. Given a unit vector $w \in \mathbb{S}^{n-1}$, define $d(w)$ to be $d(w) = \min \|w - w^*\|_2$, where w^* ranges over all $4\binom{n}{2}$ vectors of the form $\frac{1}{\sqrt{2}}(\pm e_i \pm e_j)$. Our "robust Khintchine" inequality is the following:

Theorem 5 (Robust Khintchine inequality). *There exists a universal constant $c > 0$ such that for any $w \in \mathbb{S}^{n-1}$, we have $\mathbf{K}(w) \geq \frac{1}{\sqrt{2}} + c \cdot d(w)$.*

Armed with our robust Khintchine inequality, the simple proof of Proposition 1 suggests a natural approach to lower-bounding $\mathbf{W}^{\leq 1}[\mathbf{LTF}]$. If w is such that $d(w)$ is "large" (at least some absolute constant), then the statement of Proposition 1 immediately gives a lower bound better than $1/2$. So the only remaining vectors w to handle are highly constrained vectors which are almost exactly of the form $\frac{1}{\sqrt{2}}(\pm e_i \pm e_j)$. A natural hope is that the Cauchy-Schwarz inequality in the proof of Proposition 1 is not tight for such highly constrained vectors, and indeed this is essentially how we proceed (modulo some simple cases in which it is easy to bound $\mathbf{W}^{\leq 1}$ above $1/2$ directly).

Second Proof of Theorem 1: Anticoncentration, Fourier Analysis of LTFs, and LTF Approximation. Our second proof of Theorem 1 employs several sophisticated ingredients from recent work on structural properties of LTFs [OS11, MORS10]. The first of these ingredients is a result (Theorem 6.1 of [OS11]) which essentially says that any LTF $f(x) = \mathrm{sign}(w \cdot x)$ can be perturbed very slightly to another LTF $f'(x) = \mathrm{sign}(w' \cdot x)$ (where both w and w' are unit vectors). The key properties of this perturbation are that (i) f and f' are extremely close, differing only on a tiny fraction of inputs in $\{-1, 1\}^n$; but (ii) the linear form $w' \cdot x$ has some nontrivial "anti-concentration" when x is distributed uniformly over $\{-1, 1\}^n$, meaning that very few inputs have $w' \cdot x$ very close to 0.

Why is this useful? It turns out that the anti-concentration of $w' \cdot x$, together with results on the degree-1 Fourier spectrum of "regular" halfspaces from [MORS10], lets us establish a lower bound on $\mathbf{W}^{\leq 1}[f']$ that is strictly greater than $1/2$. Then the fact that f and f' agree on almost every input in $\{-1, 1\}^n$ lets us argue that the original LTF f must similarly have $\mathbf{W}^{\leq 1}[f]$ strictly greater than $1/2$. Interestingly, the lower bound on $\mathbf{W}^{\leq 1}[f']$ is proved using the Gotsman-Linial inequality $\mathbf{W}^{\leq 1}[f'] \geq (\mathbf{K}(w))^2$; in fact, the anti-concentration of $w' \cdot x$ is combined with ingredients in the simple Fourier proof of the (original, non-robust) Khintchine inequality (specifically, an upper bound on the total influence of the function $\ell(x) = |w' \cdot x|$) to obtain the result. Because of space constraints we give this second proof in the full version of the paper.

1.4 Our Techniques for Theorem 2: Approximating the BKS Constant

As in the previous subsection, it suffices to consider only zero-threshold LTFs $\mathrm{sign}(w \cdot x)$. Our algorithm turns out to be very simple (though its analysis is not):

Let $K = \Theta(\epsilon^{-24})$. Enumerate all K-variable zero-threshold LTFs, and output the value $\Gamma_\epsilon \stackrel{\text{def}}{=} \min\{\mathbf{W}^1[f] : f \text{ is a zero-threshold } K\text{-variable LTF.}\}$.

It is well known (see e.g. [MT94]) that there exist $2^{\Theta(K^2)}$ distinct K-variable LTFs, and it is straightforward to confirm that they can be enumerated in output-polynomial time. Thus the above simple algorithm runs in time $2^{\mathrm{poly}(1/\epsilon)}$; the challenge is to show that the value Γ_ϵ thus obtained indeed satisfies Equation (1).

A key ingredient in our analysis is the notion of the "critical index" of an LTF f. The critical index was implicitly introduced and used in [Ser07] and was explicitly used in [DS09, DGJ+10, OS11, DDFS12] and other works. To define the critical index we need to first define "regularity":

Definition 4 (regularity). *Fix $\tau > 0$. We say that a vector $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$ is* τ-regular *if $\max_{i\in[n]} |w_i| \leq \tau\|w\| = \tau\sqrt{w_1^2 + \cdots + w_n^2}$. A linear form $w \cdot x$ is said to be τ-regular if w is τ-regular, and similarly an LTF is said to be τ-regular if it is of the form $\mathrm{sign}(w \cdot x - \theta)$ where w is τ-regular.*

Regularity is a helpful notion because if w is τ-regular then the Berry-Esséen theorem tells us that for uniform $x \in \{-1, 1\}^n$, the linear form $w \cdot x$ is "distributed like a Gaussian up to error τ." This can be useful for many reasons (as we will see below).

Intuitively, the critical index of w is the first index i such that from that point on, the vector $(w_i, w_{i+1}, \ldots, w_n)$ is regular. A precise definition follows:

Definition 5 (critical index). *Given a vector $w \in \mathbb{R}^n$ such that $|w_1| \geq \cdots \geq |w_n| > 0$, for $k \in [n]$ we denote by σ_k the quantity $\sqrt{\sum_{i=k}^n w_i^2}$. We define the* τ-critical index $c(w, \tau)$ of w *as the smallest index $i \in [n]$ for which $|w_i| \leq \tau \cdot \sigma_i$. If this inequality does not hold for any $i \in [n]$, we define $c(w, \tau) = \infty$.*

Returning to Theorem 2, since our algorithm minimizes over a proper subset of all LTFs, it suffices to show that for any zero-threshold LTF $f = \mathrm{sign}(w \cdot x)$, there is a K-variable zero-threshold LTF g such that $\mathbf{W}^1[g] - \mathbf{W}^1[f] < \epsilon$. At a high level our proof is a case analysis based on the size of the δ-critical index $c(w, \delta)$ of the weight vector w, where we choose the parameter δ to be $\delta = \mathrm{poly}(\epsilon)$. The first case is relatively easy: if the δ-critical index is large, then it is known that the function f is very close to some K-variable LTF g. Since the two functions agree almost everywhere, it is easy to show that $|\mathbf{W}^1[f] - \mathbf{W}^1[g]| \leq \epsilon$ as desired.

The case that the critical index is small is much more challenging. In this case it is by no means true that f can be well approximated by an LTF on few variables – consider, for example, the majority function. We deal with this challenge by developing a novel *variable reduction technique* which lets us construct a $\mathrm{poly}(1/\epsilon)$-variable LTF g whose level-1 Fourier weight closely matches that of f.

How is this done? The answer again comes from the critical index. Since the critical index $c(w, \delta)$ is small, we know that except for the "head" portion $\sum_{i=1}^{c(w,\delta)-1} w_i x_i$ of the linear form, the "tail" portion $\sum_{i=c(w,\delta)}^n w_i x_i$ of the linear form "behaves like a Gaussian." Guided by this intuition, our variable reduction technique proceeds in three steps. In the first step, we replace the tail coordinates $x_T = (x_{c(w,\delta)}, \ldots, x_n)$ by independent Gaussian random variables and show that the degree-1 Fourier weight of the

corresponding "mixed" function (which has some ± 1-valued inputs and some Gaussian inputs) is approximately equal to $\mathbf{W}^1[f]$. In the second step, we replace the tail random variable $w_T \cdot G_T$, where G_T is the vector of Gaussians from the first step, by a *single* Gaussian random variable G, where $G \sim \mathcal{N}(0, \|w_T\|^2)$. We show that this transformation exactly preserves the degree-1 weight. At this point we have reduced the number of variables from n down to $c(w, \delta)$ (which is small in this case!), but the last variable is Gaussian rather than Boolean. As suggested by the Central Limit Theorem, though, one may try to replace this Gaussian random variable by a normalized sum of independent ± 1 random variables $\sum_{i=1}^{M} z_i/\sqrt{M}$. This is exactly the third step of our variable reduction technique. Via a careful analysis, we show that by taking $M = \mathrm{poly}(1/\epsilon)$, this operation preserves the degree-1 weight up to an additive ϵ. Combining all these steps, we obtain the desired result.

1.5 Our Techniques for Theorem 3: Approximating Tomaszewski's Constant

The first step of our proof of Theorem 3 is similar in spirit to the main structural ingredient of our proof of Theorem 2: we show that given any $\epsilon > 0$, there is a value $K_\epsilon = \mathrm{poly}(1/\epsilon)$ such that it suffices to consider linear forms $w \cdot x$ over K_ϵ-dimensional space, i.e. for any $n \in \mathbf{N}$ we have $\mathbf{T}(\mathbb{S}^{n-1}) \leq \mathbf{T}(\mathbb{S}^{K_\epsilon - 1}) \leq \mathbf{T}(\mathbb{S}^{n-1}) + \epsilon$. Similar to the high-level outline of Theorem 2, our proof again proceeds by fixing any $w \in \mathbb{S}^{n-1}$ and doing a case analysis based on whether the critical index of w is "large" or "small." However, the technical details of each of these cases is quite different from the earlier proof. In the "small critical index" case we employ Gaussian anti-concentration (which is inherited by the "tail" random variable $w_T x_T$ since the tail vector w_T is regular), and in the "large critical index" case we use an anti-concentration result from [OS11].

Unlike the previous situation for the BKS constant, at this point more work remains to be done for approximating Tomaszewski's constant. While there are only $2^{\mathrm{poly}(1/\epsilon)}$ many halfspaces over $\mathrm{poly}(1/\epsilon)$ many variables and hence a brute-force enumeration could cover all of them in $2^{\mathrm{poly}(1/\epsilon)}$ time for the BKS constant, here we must contend with the fact that $\mathbb{S}^{K_\epsilon - 1}$ is an uncountably infinite set, so we cannot naively minimize over all its elements. Instead we take a dual approach and exploit the fact that while there are uncountably infinitely many vectors in $\mathbb{S}^{K_\epsilon - 1}$, there are only 2^{K_ϵ} many hypercube points in $\{-1, 1\}^{K_\epsilon}$, and (with some care) the desired infimum over all unit vectors can be formulated in the language of existential theory of the reals. We then use an algorithm for deciding existential theory of the reals (see [Ren88]) to compute the infimum. Because of space constraints we prove Theorem 3 in the full version of the paper.

Discussion. It is interesting to note that determining Tomaszewski's constant is an instance of the well-studied generic problem of understanding tails of Rademacher sums. For the sake of discussion, let us define $\mathbf{T}_{\mathsf{in}}(w, a) = \mathbf{Pr}_{x \in \{-1,1\}^n}[|w \cdot x| \leq a]$ and $\mathbf{T}_{\mathsf{out}}(w, a) = \mathbf{Pr}_{x \in \{-1,1\}^n}[|w \cdot x| \geq a]$ where $w \in \mathbb{S}^{n-1}$. Further, let $\mathbf{T}_{\mathsf{in}}(a) = \inf_{w \in \mathbb{S}} \mathbf{T}_{\mathsf{in}}(w, a)$ and $\mathbf{T}_{\mathsf{out}}(a) = \inf_{w \in \mathbb{S}} \mathbf{T}_{\mathsf{out}}(w, a)$. Note that Tomaszewski's constant $\mathbf{T}(\mathbb{S})$ is simply $\mathbf{T}_{\mathsf{in}}(1)$. Much effort has been expended on getting sharp estimates for $\mathbf{T}_{\mathsf{in}}(a)$ and $\mathbf{T}_{\mathsf{out}}(a)$ for various values of a (see e.g. [Pin12, Ben04]). As a representative example, Bentkus and Dzindzalieta [BD12] proved that $\mathbf{T}_{\mathsf{in}}(a) \geq \frac{1}{4} + \frac{1}{4} \cdot \sqrt{2 - \frac{2}{a^2}}$ for $a \in (1, \sqrt{2}]$. Similarly, Pinelis [Pin94] showed that there is an absolute constant $c > 0$

such that $\mathbf{T}_{\mathsf{out}}(a) \geq 1 - c \cdot \frac{\phi(a)}{a}$ where $\phi(x)$ is the density function of the standard normal $\mathcal{N}(0,1)$ (note this beats the standard Hoeffding bound by a factor of $1/a$).

On the complementary side, Montgomery-Smith [MS90] proved that there is an absolute constant $c' > 0$ such that $\mathbf{T}_{\mathsf{out}}(a) \geq e^{-c' \cdot a^2}$ for all $a \leq 1$. Similarly, Oleszkiewicz [Ole96] proved that $\mathbf{T}_{\mathsf{out}}(1) \geq 1/10$. The conjectured lower bound on $\mathbf{T}_{\mathsf{out}}(1)$ is $7/32$ (see [HK94]). While we have not investigated this in detail, we suspect that our techniques may be applicable to some of the above problems. Finally, we note that apart from being of intrinsic interest to functional analysts and probability theorists, the above quantities arise frequently in the optimization literature (see [HLNZ08, BTNR02]). Related tail bounds have also found applications in extremal combinatorics (see [AHS12]).

2 Proof of Theorem 5: A "Robust" Khintchine Inequality

It will be convenient for us to reformulate Theorems 4 and 5 as follows: Let us say that a unit vector $w = (w_1, \ldots, w_n) \in \mathbb{S}^{n-1}$ is *proper* if $w_i \geq w_{i+1} \geq 0$ for all $i \in [n-1]$. Then we may state the "basic" Khintchine inequality with optimal constant, Theorem 4, in the following equivalent way:

Theorem 6 (Khintchine inequality, [Sza76]). *Let $w \in \mathbb{R}^n$ be a proper unit vector. Then $\mathbf{K}(w) \geq 1/\sqrt{2}$, with equality if and only if $w = w^* \stackrel{def}{=} (1/\sqrt{2}, 1/\sqrt{2}, 0, \ldots, 0)$.*

And we may restate our "robust" Khintchine inequality, Theorem 5, as follows:

Theorem 7 (Robust Khintchine inequality). *There exists a universal constant $c > 0$ such that the following holds: Let $w \in \mathbb{R}^n$ be a proper unit vector. Then $\mathbf{K}(w) \geq 1/\sqrt{2} + c \cdot \|w - w^*\|_2$,where $w^* \stackrel{def}{=} (1/\sqrt{2}, 1/\sqrt{2}, 0, \ldots, 0)$.*

Before we proceed with the proof of Theorem 7, we give a simple Fourier analytic proof of the "basic" Khintchine inequality with optimal constant, $\mathbf{K}(w) \geq 1/\sqrt{2}$. (We note that this is a well-known argument by now; it is given in somewhat more general form in [Ole99] and in [KLO96].) We then build on this to prove Theorem 7.

2.1 Warm-Up: Simple Proof That $\mathbf{K}(w) \geq 1/\sqrt{2}$

We consider the function $\ell(x) = |\sum_{i=1}^n w_i x_i|$ where $\sum_i w_i^2 = 1$ and will show that $\mathbf{K}(w) = \mathbf{E}_x[\ell(x)] \geq 1/\sqrt{2}$. Noting that $\mathbf{E}_x[(\ell(x))^2] = 1$, we have $(\mathbf{E}[\ell(x)])^2 = 1 - \mathrm{Var}[\ell]$, so it suffices to show that $\mathrm{Var}[\ell] \leq 1/2$. This follows directly by combining the following claims. The first bound is an improved Poincaré inequality for even functions:

Fact 8. *(Poincaré inequality) Let $f : \{-1,1\}^n \to \mathbb{R}$ be even. Then* $\mathrm{Var}[f] \leq (1/2) \cdot \mathrm{Inf}(f)$.

Proof. Since f is even, we have that $\widehat{f}(S) = 0$ for all S with odd $|S|$. We can thus write

$$\mathrm{Inf}(f) = \sum_{S \subseteq [n], |S| \text{ even}} |S| \cdot \widehat{f}^2(S) \geq 2 \cdot \sum_{\emptyset \neq S, |S| \text{ even}} \widehat{f}^2(S) = 2 \cdot \sum_{\emptyset \neq S \subseteq [n]} \widehat{f}^2(S) = 2 \cdot \mathrm{Var}[f].$$

The second is an upper bound on the influences in ℓ as a function of the weights:

Lemma 1. *Let* $\ell(x) = |\sum_{i=1}^n w_i x_i|$. *For any* $i \in [n]$, *we have* $\mathrm{Inf}_i(\ell) \leq w_i^2$.

Proof. Recall that $\mathrm{Inf}_i(\ell) = \mathbf{E}_x\left[\mathrm{Var}_{x_i}\left[\ell(x)\right]\right] = \mathbf{E}_x\left[\mathbf{E}_{x_i}[\ell^2(x)] - (\mathbf{E}_{x_i}[\ell(x)])^2\right]$. We claim that for any $x \in \{-1,1\}^n$, it holds that $\mathrm{Var}_{x_i}[\ell(x)] \leq w_i^2$, which yields the lemma. To show this claim we write $\ell(x) = |w_i x_i + c_i|$, where $c_i = \sum_{j \neq i} w_j \cdot x_j$ does not depend on x_i.

Since $\ell^2(x) = c_i^2 + w_i^2 + 2c_i w_i x_i$, it follows that $\mathbf{E}_{x_i}[\ell^2(x)] = c_i^2 + w_i^2$, and clearly $\mathbf{E}_{x_i}[\ell(x)] = (1/2) \cdot (|w_i - c_i| + |w_i + c_i|)$. We consider two cases based on the relative magnitudes of c_i and w_i.

If $|c_i| \leq |w_i|$, we have $\mathbf{E}_{x_i}[\ell(x)] = (1/2) \cdot (\mathrm{sign}(w_i)(w_i - c_i) + \mathrm{sign}(w_i)(w_i + c_i))$ $= |w_i|$. Hence, in this case $\mathrm{Var}_{x_i}[\ell(x)] = c_i^2 \leq w_i^2$. If on the other hand $|c_i| > |w_i|$, then we have $\mathbf{E}_{x_i}[\ell(x)] = (1/2) \cdot (\mathrm{sign}(c_i)(c_i - w_i) + \mathrm{sign}(c_i)(c_i + w_i)) = |c_i|$, so again $\mathrm{Var}_{x_i}[\ell(x)] = w_i^2$ as desired. □

The bound $\mathbf{K}(w) \geq 1/\sqrt{2}$ follows from the above two claims using the fact that ℓ is even and that $\sum_i w_i^2 = 1$.

2.2 Proof of Theorem 7

Let $w \in \mathbb{R}^n$ be a proper unit vector and denote $\tau = \|w - w^*\|_2$. To prove Theorem 7, one would intuitively want to obtain a robust version of the simple Fourier-analytic proof of Theorem 6 from the previous subsection. Recall that the latter proof boils down to the following:

$$\mathrm{Var}[\ell] \leq (1/2) \cdot \mathrm{Inf}(\ell) = (1/2) \cdot \sum_{i=1}^n \mathrm{Inf}_i(\ell) \leq (1/2) \cdot \sum_{i=1}^n w_i^2 = 1/2$$

where the first inequality is Fact 8 and the second is Lemma 1. While it is clear that both inequalities can be individually tight, one could hope to show that both inequalities cannot be tight simultaneously. It turns out that this intuition is not quite true, however it holds if one imposes some additional conditions on the weight vector w. The remaining cases for w that do not satisfy these conditions can be handled by elementary arguments.

We first note that without loss of generality we may assume that $w_1 = \max_i w_i > 0.3$, for otherwise Theorem 7 follows directly from the following result of König *et al*:

Theorem 9 ([KSTJ99]). *For a proper unit vector* $w \in \mathbb{R}^n$, *we have* $\mathbf{K}(w) \geq \sqrt{2/\pi} - (1 - \sqrt{2/\pi})w_1$.

Indeed, if $w_1 \leq 0.3$, the above theorem gives that $\mathbf{K}(w) \geq 1.3\sqrt{2/\pi} - 0.3 > 0.737 > 1/\sqrt{2} + 3/100 \geq 1/\sqrt{2} + (1/50)\tau$, where the last inequality follows from the fact that $\tau \leq \sqrt{2}$ (as both w and w^* are unit vectors). Hence, we will henceforth assume that $w_1 > 0.3$.

The preceding discussion leads us to the following definition:

Definition 6 (canonical vector). *We say that a proper unit vector* $w \in \mathbb{R}^n$ *is* canonical *if it satisfies the following conditions: (a)* $w_1 \in [0.3, 1/\sqrt{2} + 1/100]$; *and (b)* $\tau = \|w - w^*\|_2 \geq 1/5$.

The following lemma establishes Theorem 7 for non-canonical vectors:

Lemma 2. *Let w be a proper non-canonical vector. Then $\mathbf{K}(w) \geq 1/\sqrt{2}+(1/1000)\tau$, where $\tau = \|w - w^*\|_2$.*

The proof of Lemma 2 is elementary, using only basic facts about symmetric random variables, but sufficiently long that we give it in the full version. For canonical vectors we show:

Theorem 10. *There exist universal constants $c_1, c_2 > 0$ such that: Let $w \in \mathbb{R}^n$ be canonical. Consider the mapping $\ell(x) = |w \cdot x|$. Then at least one of the following statements is true : (1) $\mathrm{Inf}_1(\ell) \leq w_1^2 - c_1$; (2) $\mathbf{W}^{>2}[\ell] \geq c_2$.*

This proof is more involved, using Fourier analysis and critical index arguments (see the full version). We proceed now to show that for canonical vectors, Theorem 7 follows from Theorem 10. To see this we argue as follows: Let $w \in \mathbb{R}^n$ be canonical. We will show that there exists a universal constant $c > 0$ such that $\mathbf{K}(w) \geq 1/\sqrt{2} + c$; as mentioned above, since $\tau < \sqrt{2}$, this is sufficient for our purposes. Now recall that

$$\mathbf{K}(w) = \mathbf{E}_x[\ell(x)] = \widehat{\ell}(0) = \sqrt{1 - \mathrm{Var}[\ell]}. \tag{3}$$

In both cases, we will show that there exists a constant $c' > 0$ such that

$$\mathrm{Var}[\ell] \leq 1/2 - c'. \tag{4}$$

From this (3) gives $\mathbf{K}(w) \geq \sqrt{1/2 + c'} = 1/\sqrt{2} + c''$ where $c'' > 0$ is a universal constant, so to establish Theorem 7 it suffices to establish (4).

Suppose first that statement (1) of Theorem 10 holds. In this case we exploit the fact that Lemma 1 is not tight. We can write

$$\mathrm{Var}[\ell] \leq (1/2) \cdot \mathrm{Inf}(f) \leq (1/2) \cdot \left(w_1^2 - c_1 + \sum_{i=2}^{n} w_i^2 \right) \leq (1/2) - c_1/2,$$

giving (4). Now suppose that statement (2) of Theorem 10 holds, i.e. at least a c_2 fraction of the total Fourier mass of ℓ lies above level 2. Since ℓ is even, this is equivalent to the statement $\mathbf{W}^{\geq 4}[\ell] \geq c_2$. In this case, we prove a better upper bound on the variance because Fact 8 is not tight. In particular, we have

$$\mathrm{Inf}(\ell) \geq 2\mathbf{W}^2[\ell]+4\mathbf{W}^{\geq 4}[\ell] = 2\left(\mathrm{Var}[\ell] - \mathbf{W}^{\geq 4}[\ell]\right)+4\mathbf{W}^{\geq 4}[\ell] = 2\,\mathrm{Var}[\ell]+2\mathbf{W}^{\geq 4}[\ell]$$

which yields $\mathrm{Var}[\ell] \leq (1/2)\mathrm{Inf}(\ell) - \mathbf{W}^{\geq 4}[\ell] \leq (1/2) - c_2$, again giving (4) as desired.

3 Proof of Theorem 1 Using Theorem 5

We first observe that it suffices to prove the theorem for balanced LTFs, i.e. LTFs $f : \{-1,1\}^n \to \{-1,1\}$ with $\widehat{f}(\emptyset) = \mathbf{E}[f] = 0$. (Note that any balanced LTF can be represented with a threshold of 0, i.e. $f(x) = \mathrm{sign}(w \cdot x)$ for some $w \in \mathbb{R}^n$.) To see this, let $f : \{-1,1\}^n \to \{-1,1\}$ be an arbitrary n-variable threshold function, i.e.

$f(x) = \mathrm{sign}(w_0 + \sum_{i=1}^n w_i x_i)$, and note that we may assume that $w_0 \neq w \cdot x$ for all $x \in \{-1,1\}^n$. Consider the $(n+1)$-variable balanced LTF $g : (x,y) \to \{-1,1\}$, where $y \in \{-1,1\}$, defined by $g(x,y) = \mathrm{sign}(w_0 y + \sum_{i=1}^n w_i x_i)$. Then it is easy to see that $\widehat{g}(y) = \mathbf{E}[f]$ and $\widehat{g}(i) = \widehat{f}(i)$ for all $i \in [n]$. Therefore, $\mathbf{W}^{\leq 1}[f] = \mathbf{W}^1[g] = \mathbf{W}^{\leq 1}[g]$.

Let $f = \mathrm{sign}(w \cdot x)$ be an LTF. We may assume that w is a proper unit vector, i.e. that $\|w\|_2 = 1$ and $w_i \geq w_{i+1} > 0$ for $i \in [n-1]$. We can also assume that $w \cdot x \neq 0$ for all $x \in \{-1,1\}^n$. We distinguish two cases: If w is "far" from w^* (i.e. the worst-case vector for the Khintchine inequality), the desired statement follows immediately from our robust inequality (Theorem 5). For the complementary case, we use a separate argument that exploits the structure of w. More formally, we have the following two cases:

Let $\tau > 0$ be a sufficiently small universal constant, to be specified.

[Case I: $\|w - w^*\|_2 \geq \tau$]. In this case, Proposition 1 and Theorem 5 give us

$$\mathbf{W}^1[f] \geq (\mathbf{K}(w))^2 \geq (1/\sqrt{2} + c\tau)^2 \geq 1/2 + \sqrt{2}c\tau$$

which completes the proof of Theorem 1 for Case I.

[Case II: $\|w - w^*\|_2 \leq \tau$]. In this case the idea is to consider the restrictions of f obtained by fixing the variables x_1, x_2 and argue based on their bias. See the full version for details.

References

[AHS12] Alon, N., Huang, H., Sudakov, B.: Nonnegative k-sums, fractional covers, and probability of small deviations. J. Combin. Theory, Series B 102(3), 784–796 (2012)

[BD12] Bentkus, V., Dzindzalieta, D.: A tight Gaussian bound for weighted sums of Rademacher random variables. Technical report (2012) (preprint)

[BDD98] Ben-David, S., Dichterman, E.: Learning with restricted focus of attention. Journal of Computer and System Sciences 56(3), 277–298 (1998)

[Ben04] Bentkus, V.: On Hoeffding's Inequality. Annals of Probability 32, 1650–1673 (2004)

[BKS99] Benjamini, I., Kalai, G., Schramm, O.: Noise sensitivity of Boolean functions and applications to percolation. Inst. Hautes Études Sci. Publ. Math. 90, 5–43 (1999)

[BTNR02] Ben-Tal, A., Nemirovski, A., Roos, C.: Robust Solutions of Uncertain Quadratic and Conic-Quadratic Problems. SIAM J. Optimization 13(2), 535–560 (2002)

[DDFS12] De, A., Diakonikolas, I., Feldman, V., Servedio, R.: Near-optimal solutions for the Chow Parameters Problem and low-weight approximation of halfspaces. In: Proc. 44th ACM Symposium on Theory of Computing (STOC), pp. 729–746 (2012)

[DGJ+10] Diakonikolas, I., Gopalan, P., Jaiswal, R., Servedio, R., Viola, E.: Bounded independence fools halfspaces. SIAM J. on Comput. 39(8), 3441–3462 (2010)

[DS09] Diakonikolas, I., Servedio, R.: Improved approximation of linear threshold functions. In: Proc. 24th CCC, pp. 161–172 (2009)

[Fil12] Filmus, Y.: Khintchine-Kahane using Fourier Analysis (2012), Posted at `http://www.cs.toronto.edu/~yuvalf/KK.pdf`

[GL94] Gotsman, C., Linial, N.: Spectral properties of threshold functions. Combinatorica 14(1), 35–50 (1994)

[Guy86] Guy, R.K.: Any answers anent these analytical enigmas? American Math. Monthly 93, 279–281 (1986)

[Haa82] Haagerup, U.: The best constants in the Khintchine inequality. Studia Math. 70, 231–283 (1982)
[HK92] Holzman, R., Kleitman, D.J.: On the product of sign vectors and unit vectors. Combinatorica 12(3), 303–316 (1992)
[HK94] Hitczenko, P., Kwapień, S.: On the Rademacher series. In: Probability in Banach Spaces, pp. 31–36 (1994)
[HLNZ08] He, S., Luo, Z., Nie, J., Zhang, S.: Semidefinite Relaxation Bounds for Indefinite Homogenous Quadratic Optimization. SIAM J. Optimization 19, 503–523 (2008)
[Jac06] Jackson, J.C.: Uniform-distribution learnability of noisy linear threshold functions with restricted focus of attention. In: Lugosi, G., Simon, H.U. (eds.) COLT 2006. LNCS (LNAI), vol. 4005, pp. 304–318. Springer, Heidelberg (2006)
[KLO96] Kwapień, S., Latala, R., Oleszkiewicz, K.: Comparison of moments of sums of independent random variables and differential inequalities. J. Funct. Anal. 136, 258–268 (1996)
[KSTJ99] König, H., Schütt, C., Tomczak-Jaegermann, N.: Projection constants of symmetric spaces and variants of Khintchine's inequality. J. Reine Agnew. Math. 511, 1–42 (1999)
[LO94] Latala, R., Oleszkiewicz, K.: On the best constant in the Khinchin-Kahane inequality. Studia Math. 109, 101–104 (1994)
[MORS10] Matulef, K., O'Donnell, R., Rubinfeld, R., Servedio, R.: Testing halfspaces. SIAM J. on Comput. 39(5), 2004–2047 (2010)
[MS90] Montgomery-Smith, S.: The distribution of Rademacher sums. In: Proceedings of the American Mathematical Society, pp. 517–522 (1990)
[MT94] Maass, W., Turan, G.: How fast can a threshold gate learn? In: Computational Learning Theory and Natural Learning Systems: Volume I: Constraints and Prospects, pp. 381–414. MIT Press (1994)
[O'D12] O'Donnell, R.: Analysis of boolean functions. Technical report (2012), Book serialization available at `http://analysisofbooleanfunctions.org/`
[Ole96] Oleszkiewicz, K.: On the Stein property of Rademacher sequences. Probability and Mathematical Statistics 16, 127–130 (1996)
[Ole99] Oleszkiewicz, K.: Comparison of moments via Poincaré-type inequality. In: Contemporary Mathematics, vol. 234, pp. 135–148 (1999)
[OS11] O'Donnell, R., Servedio, R.: The Chow Parameters Problem. SIAM J. on Comput. 40(1), 165–199 (2011)
[Per04] Peres, Y.: Noise stability of weighted majority (2004), `http://arxiv.org/abs/math/0412377`
[Pin94] Pinelis, I.: Extremal probabilistic problems and Hotelling's t^2 test under a symmetry condition. Ann. Statist. 22, 357–368 (1994)
[Pin12] Pinelis, I.: An asymptotically Gaussian bound on the Rademacher tails. Electronic Journal of Probability 17, 1–22 (2012)
[Ren88] Renegar, J.: A faster PSPACE algorithm for deciding the existential theory of the reals. In: IEEE Annual Symposium on Foundations of Computer Science, pp. 291–295 (1988)
[Ser07] Servedio, R.: Every linear threshold function has a low-weight approximator. Comput. Complexity 16(2), 180–209 (2007)
[So09] So, A.M.-C.: Improved approximation bound for quadratic optimization problems with orthogonality constraints. In: SODA, pp. 1201–1209 (2009)
[Sza76] Szarek, S.J.: On the best constants in the Khinchine inequality. Studia Math. 58, 197–208 (1976)
[Tom87] Tomaszewski, B.: A simple and elementary proof of the Khintchine inequality with the best constant. Bull. Sci. Math. 111(2), 103–109 (1987)

Combining Binary Search Trees*

Erik D. Demaine[1], John Iacono[2,**],
Stefan Langerman[3,***], and Özgür Özkan[2,†]

[1] Massachusetts Institute of Technology
[2] Polytechnic Institute of New York University
[3] Université Libre de Bruxelles

Abstract. We present a general transformation for combining a constant number of binary search tree data structures (BSTs) into a single BST whose running time is within a constant factor of the minimum of any "well-behaved" bound on the running time of the given BSTs, for any online access sequence. (A BST has a *well-behaved* bound with $f(n)$ overhead if it spends at most $\mathcal{O}(f(n))$ time per access and its bound satisfies a weak sense of closure under subsequences.) In particular, we obtain a BST data structure that is $\mathcal{O}(\log\log n)$ competitive, satisfies the working set bound (and thus satisfies the static finger bound and the static optimality bound), satisfies the dynamic finger bound, satisfies the unified bound with an additive $\mathcal{O}(\log\log n)$ factor, and performs each access in worst-case $\mathcal{O}(\log n)$ time.

1 Introduction

Binary search trees (BSTs) are one of the most fundamental and well-studied data structures in computer science. Yet, many fundamental questions about their performance remain open. While information theory dictates the worst-case running time of a single access in an n node BST to be $\Omega(\log n)$, which is achieved by many BSTs (e.g., [2]), BSTs are generally not built to execute a single access, and there is a long line of research attempting to minimize the overall running time of executing an online access sequence. This line of work was initiated by Allen and Munro [1], and then by Sleator and Tarjan [15] who invented the splay tree. Central to splay trees and many of the data structures in the subsequent literature is the BST model. The BST model provides a precise model of computation, which is not only essential for comparing different BSTs, but also allows the obtaining of lower bounds on the optimal offline BST.

In the BST model, the elements of a totally ordered set are stored in the nodes of a binary tree and a BST data structure is allowed at unit cost to manipulate the tree by following the parent, left-child, or right-child pointers at each node or rotate the node with its parent. We give a formal description of

* See [9] for the full version.

** Research supported by NSF Grant CCF-1018370.

*** Directeur de Recherches du F.R.S.-FNRS. Research partly supported by the F.R.S.-FNRS and DIMACS.

† Research supported by GAANN Grant P200A090157 from the US Department of Education.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 388–399, 2013.

the model in Section 1.1. A common theme in the literature since the invention of splay trees concerns proving various bounds on the running time of splay trees and other BST data structures [4, 6, 7, 12, 15]. Sleator and Tarjan [15] proved a number of upper bounds on the performance of splay trees. The *static optimality* bound requires that any access sequence is executed within a constant factor of the time it would take to execute it on the best static tree for that sequence. The *static finger* bound requires that each access x is executed in $\mathcal{O}(\log d(f, x))$ amortized time where $d(f, x)$ is the number of keys between any fixed finger f and x. The *working set* bound requires that each access x is executed in $\mathcal{O}(\log w(x))$ amortized time where $w(x)$ is the number of elements accessed since the last access to x. Cole [6] and Cole et al. [7] later proved that splay trees also have the *dynamic finger* bound which requires that each access x is executed in $\mathcal{O}(\log d(y, x))$ amortized time where y is the previous item in the access sequence. Iacono [14] introduced the *unified* bound, which generalizes and implies both the dynamic finger and working set bounds. Bose et al. [4] presented layered working set trees, and showed how to achieve the unified bound with an additive cost of $\mathcal{O}(\log \log n)$ per access, by combining them with the skip-splay trees of Derryberry and Sleator [12].

A BST data structure satisfies the dynamic optimality bound if it is $\mathcal{O}(1)$-competitive with respect to the best offline BST data structure. Dynamic optimality implies all other bounds of BSTs. The existence of a dynamically optimal BST data structure is a major open problem. While splay trees were conjectured by Sleator and Tarjan to be dynamically optimal, despite decades of research, there were no online BSTs known to be $o(\log n)$-competitive until Demaine et al. invented Tango trees [8] which are $\mathcal{O}(\log \log n)$-competitive. Later, Wang et al. [17] presented a variant of Tango trees, called multi-splay trees, which are also $\mathcal{O}(\log \log n)$-competitive and retain some bounds of splay trees. Bose et al. [3] gave a transformation where given any BST whose amortized running time per access is $\mathcal{O}(\log n)$, they show how to deamortize it to obtain $\mathcal{O}(\log n)$ worst-case running time per access while preserving its original bounds.

Results and Implications. In this paper we present a structural tool to combine bounds of BSTs from a certain general class of BST bounds, which we refer to as well-behaved bounds. Specifically, our method can be used to produce an online BST data structure which combines well-behaved bounds of all known BST data structures. In particular, we obtain a BST data structure that is $\mathcal{O}(\log \log n)$ competitive, satisfies the working set bound (and thus satisfies the static finger bound and the static optimality bound), satisfies the dynamic finger bound, satisfies the unified bound with an additive $\mathcal{O}(\log \log n)$, and performs each access in worst-case $\mathcal{O}(\log n)$ time. Moreover, we can add to this list any well-behaved bound realized by a BST data structure.

Note that requiring the data structures our method produces to be in the BST model precludes the possibility of a trivial solution such as running all data structures in parallel and picking the fastest.

Our result has a number of implications. First, it could be interpreted as a weak optimality result where our method produces a BST data structure which is $\mathcal{O}(1)$-competitive with respect to a constant number of given BST data structures whose actual running times are well-behaved. In comparison, a dynamically optimal BST data structure, if one exists, would be $\mathcal{O}(1)$-competitive with

respect to all BST data structures. On the other hand, the existence of our method is a necessary condition for the existence of a dynamically optimal BST. Lastly, techniques introduced in this paper (in particular the simulation of multiple fingers in Section 2) may be of independent interest for augmenting a BST in nontrivial ways, as we do here. Indeed, they are also used in [5].

1.1 Preliminaries

The BST Model. Given a set S of elements from a totally ordered universe, where $|S| = n$, a BST data structure T stores the elements of S in a rooted tree, where each node in the tree stores an element of S, which we refer to as the key of the node. The node also stores three pointers pointing to its parent, left child, and right child. Any key contained in the left subtree of a node is smaller than the key stored in the node; and any key contained in the right subtree of a node is greater than the key stored in the node. Each node can store data in addition to its key and the pointers.

Although BST data structures usually support insertions, deletions, and searches, in this paper we consider only successful searches, which we call *accesses*. To implement such searches, a BST data structure has a single pointer which we call the finger, pointed to a node in the BST T. The finger initially points to the root of the tree before the first access. Whenever a finger points to a node as a result of an operation o we say the node is *touched*, and denote the node by $N(o)$. An access sequence $(x_1, x_2, \ldots, x_m)$ satisfies $x_i \in S$ for all i. A BST data structure executes each access i by performing a sequence of unit-cost operations on the finger—where the allowed unit-cost operations are following the left-child pointer, following the right-child pointer, following the parent pointer, and performing a rotation on the finger and its parent—such that the node containing the search key x_i is touched as a result of these operations. Any augmented data stored in a node can be modified when the node is touched during an access. The running time of an access is the number of unit-cost operations performed during that access.

An *offline* BST data structure executes each operation as a function of the entire access sequence. An *online* BST data structure executes each operation as a function of the prefix of the access sequence ending with the current access. Furthermore, as coined by [3], a *real-world* BST data structure is one which can be implemented with a constant number of $\mathcal{O}(\log n)$ bit registers and $\mathcal{O}(\log n)$ bits of augmented data at each node.

The Multifinger-BST Model. The Multifinger-BST model is identical to the BST model with one difference: in the Multifinger-BST model we have access to a constant number of fingers, all initially pointing to the root.

We now formally define what it means for a BST data structure to simulate a Multifinger-BST data structure.

Definition 1. *A BST data structure S* simulates *a Multifinger-BST data structure M if there is a correspondence between the ith operation $op_M[i]$ performed by M and a contiguous subsequence $op_S[j_i], op_S[j_i + 1], \ldots, N(op_S[j_{i+1} - 1])$ of the operations performed by S, for $j_1 < j_2 < \cdots$, such that the touched nodes satisfy $N(op_M[i]) \in \{N(op_S[j_i]), N(op_S[j_i + 1]), \ldots, N(op_S[j_{i+1} - 1])\}$.*

For any access sequence X, there exists an offline BST data structure that executes it optimally. We denote the number of unit-cost operations performed by this data structure by $\mathrm{OPT}(X)$. An online BST data structure is c-competitive if it executes all sequences X of length $\Omega(n)$ in time at most $c \cdot \mathrm{OPT}(X)$, where n is the number of nodes in the tree. An online BST that is $\mathcal{O}(1)$-competitive is called *dynamically optimal.*

Because a BST data structure is a Multifinger-BST data structure with one finger, the following definitions apply to BST data structures as well.

Definition 2. *Given a Multifinger-BST data structure A and an initial tree T, let $\mathcal{T}(A,T,X) = \sum_{i=1}^{|X|} \tau(A,T,X,i) + f(n)$ be an upper bound on the total running time of A on any access sequence X starting from tree T, where $\tau(A,T,X,i)$ denotes an amortized upper bound on the running time of A on the ith access of access sequence X, and $f(n)$ denotes the overhead. Define $\mathcal{T}_{X'}(A,T,X) = \sum_{i=1}^{|X'|} \tau(A,T,X,\pi(i))$ where X' is a contiguous subsequence of access sequence X and $\pi(i)$ is the index of the ith access of X' in X. The bound τ is **well-behaved** with overhead $f(n)$ if there exists constants C_0 and C_1 such that the cost of executing any single access x_i is at most $C_1 \cdot f(n)$, and for any given tree T, access sequence X, and any contiguous subsequence X' of X, $\mathcal{T}(A,T,X') \leq C_0 \cdot \mathcal{T}_{X'}(A,T,X) + C_1 \cdot f(n)$.*

1.2 Our Results

Given k online BST data structures $\mathcal{A}_1, \ldots, \mathcal{A}_k$, where k is a constant, our main result is the design of an online BST data structure which takes as input an online access sequence $(x_1, \ldots, x_m)$, along with an initial tree T; and executes, for all j, access sequence $(x_1, \ldots, x_j)$ in time $\mathcal{O}\left(\min_{i \in \{1,\ldots,k\}} \mathcal{T}(\mathcal{A}_i, T, (x_1, \ldots, x_j))\right)$ where $\mathcal{T}(\mathcal{A}_i, T, X)$ is a well-behaved bound on the running time of $\mathcal{A}_i$. To simplify the presentation, we let $k = 2$. By combining k BSTs two at a time, in a balanced binary tree, we achieve an $O(k)$ (constant) overhead.

Theorem 3. *Given two online BST data structures $\mathcal{A}_0$ and $\mathcal{A}_1$, let $\mathcal{T}_{X'}(\mathcal{A}_0,T,X)$ and $\mathcal{T}_{X'}(\mathcal{A}_1,T,X)$ be well-behaved amortized upper bounds with overhead $f(n) \geq n$ on the running time of $\mathcal{A}_0$ and $\mathcal{A}_1$, respectively, on a contiguous subsequence X' of any online access sequence X from an initial tree T. Then there exists an online BST data structure,* Combo-BST $=$ Combo-BST$(\mathcal{A}_0, \mathcal{A}_1, f(n))$ *such that*

$$\mathcal{T}_{X'}(\textsf{Combo-BST}, T, X) = \mathcal{O}(\min(\mathcal{T}_{X'}(\mathcal{A}_0,T,X), \mathcal{T}_{X'}(\mathcal{A}_1,T,X)) + f(n)).$$

If $\mathcal{A}_0$ and $\mathcal{A}_1$ are real-world BST data structures, so is Combo-BST.

Corollary 4. *There exists a BST data structure that is $\mathcal{O}(\log\log n)$-competitive, satisfies the working set bound (and thus satisfies the static finger bound and the static optimality bound), satisfies the dynamic finger bound, satisfies the unified bound*[1] *with an additive $\mathcal{O}(\log\log n)$, all with additive overhead $\mathcal{O}(n \log n)$, and performs each access in worst-case $\mathcal{O}(\log n)$ time.*

[1] The Cache-splay tree [11] was claimed to achieve the unified bound. However, this claim has been rescinded by one of the authors at the 5th Bertinoro Workshop on Algorithms and Data Structures.

Proof. We apply Theorem 3 to combine the bounds of the splay tree, the multi-splay tree [17], and the layered working set tree [4]. The multi-splay tree is $\mathcal{O}(\log \log n)$-competitive. Observe that $\mathrm{OPT}(X)$ is a well-behaved bound with overhead $\mathcal{O}(n)$ because any tree can be transformed to any other tree in $\mathcal{O}(n)$ time [16]. Therefore, $\mathcal{O}(\log \log n)$-competitiveness of multi-splay trees is a well-behaved bound with overhead $\mathcal{O}(n \log \log n)$. On the other hand, the multi-splay tree also satisfies the working set bound. The working set bound is a well-behaved bound with $\mathcal{O}(n \log n)$ overhead because only the first instance of each item in a subsequence of an access sequence has a different working set number with respect to that subsequence and the log of each such difference is upper bounded by $\log n$. The working set bound implies the static finger and static optimality bounds with overhead $O(n \log n)$ [13]. The splay tree satisfies the the dynamic finger bound [6, 7], which is a well-behaved bound with $\mathcal{O}(n)$ overhead because the additive term in the dynamic finger bound is linear and only the first access in a subsequence may have an increase in the amortized bound which is at most $\log n$. The layered working set tree [4] satisfies the unified bound with an additive $\mathcal{O}(\log \log n)$. Similar to the working set bound, the unified bound is a well-behaved bound with $\mathcal{O}(n \log n)$ overhead because only the first instance of each item in a subsequence of an access sequence has a different unified bound value with respect to that subsequence and each such difference is at most $\log n$. Therefore, because the $\mathcal{O}(\log \log n)$ term is additive and is dominated by $\mathcal{O}(\log n)$, the unified bound with an additive $\mathcal{O}(\log \log n)$ is a well-behaved bound with $\mathcal{O}(n \log n)$ overhead. Lastly, because the multi-splay tree performs each access in $\mathcal{O}(\log n)$ worst-case time and because $\mathcal{O}(\log n)$ is a well-behaved bound with no overhead, we can apply the transformation of Bose et al. [3] to our BST data structure to satisfy all of our bounds while performing each access in $\mathcal{O}(\log n)$ worst-case time. □

To achieve these results, we present OneFinger-BST, which can simulate any Multifinger-BST data structure in the BST model in constant amortized time per operation. We will present our Combo-BST data structure as a Multifinger-BST data structure in Section 4 and use OneFinger-BST to transform it into a BST data structure.

Theorem 5. *Given any Multifinger-BST data structure A, where $op_A[j]$ is the jth operation performed by A,* OneFinger-BST(A) *is a BST data structure such that, for any k, given k operations $(op_A[1], \ldots, op_A[k])$ online,* OneFinger-BST(A) *simulates them in $C_2 \cdot k$ total time for some constant C_2 that depends on the number of fingers used by A. If A is a real-world BST data structure, then so is* OneFinger-BST(A).

Organization and Roadmap. The rest of the paper is organized as follows. We present a method to transform a Multifinger-BST data structure to a BST data structure (Theorem 5) in Section 2. We show how to load and save the state of the tree in $\mathcal{O}(n)$ time in the Multifinger-BST model using multiple fingers in Section 3. We present our BST data structure, the Combo-BST, in Section 4. We analyze Combo-BST and prove our Theorem 3 in Section 5.

2 Simulating Multiple Fingers

In this section, we present OneFinger-BST, which transforms any given Multifinger-BST data structure T into a BST data structure OneFinger-BST(T).

Structural Terminology. First we present some structural terminology defined by [10], adapted here to the BST model; refer to Figure 1.

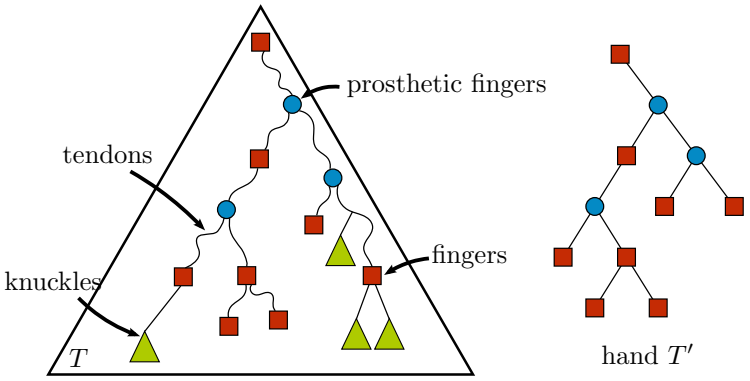


Fig. 1. A tree T with a set of fingers, and the corresponding hand structure

Given any Multifinger-BST T with a set F of fingers $f_1, \ldots, f_{|F|}$, where $|F| = \mathcal{O}(1)$, let $S(T, F)$ be the be the Steiner tree with terminals f_i, that is, the union of shortest paths in T between all pairs of fingers[2] F. We define *prosthetic fingers*, denoted by $P'(T, F)$, to be the set of nodes with degree 3 in $S(T, F)$ that are not in F. Then, we define the set of *pseudofingers*, denoted by $P(T, F)$, to be $P(T, F) = F \cup P'(T, F)$. Note that $|P(T, F)| \leq 2|F| = \mathcal{O}(1)$. The *hand* $H(T, F)$ is the compressed Steiner tree obtained from the Steiner tree $S(T, F)$ by contracting every vertex not in $P(T, F)$ (each of degree 2). A *tendon* $\tau_{x,y}$ is the shortest path in $S(T, F)$ connecting two pseudofingers x and y (excluding nodes x and y), where x is an ancestor of y and x and y are adjacent in $H(T, F)$. We refer to x as the top of $\tau_{x,y}$ and y as the bottom of $\tau_{x,y}$. A *knuckle* is a connected component of T after removing all of its pseudofingers and tendons.

To avoid confusion, we use $\mathsf{parent}_T(x)$, $\mathsf{left\text{-}child}_T(x)$, and $\mathsf{right\text{-}child}_T(x)$ to denote the pointers of a node x in the Multifinger-BST T, and use $\mathsf{parent}(x)$, $\mathsf{left\text{-}child}(x)$, and $\mathsf{right\text{-}child}(x)$ to denote the pointers of a node x in OneFinger-BST(T).

Our Approach. To simulate a Multifinger-BST data structure, OneFinger-BST needs to handle the movement and rotation of multiple fingers. To accomplish this, OneFinger-BST maintains the hand structure. We discuss how this is done at a high level in Section 2.2. However, a crucial part of maintaining the hand

[2] For convenience, we also define the root of the tree T to be a finger.

is an efficient implementation of tendons in the BST model where the distance between any two fingers connected by a tendon is at most a constant. We will implement a tendon as a pair of double-ended queues (deques). The next lemma lets us do this by showing that a tendon consists of an increasing and a decreasing subsequence. See Figure 2a.

Lemma 6. *A tendon can be partitioned into two subsets of nodes $T_>$ and $T_<$ such that the level-order key values of nodes in $T_>$ are increasing and the level-order key values of nodes in $T_<$ are decreasing, where the maximum key value in $T_>$ is smaller than the minimum key value in $T_<$.*

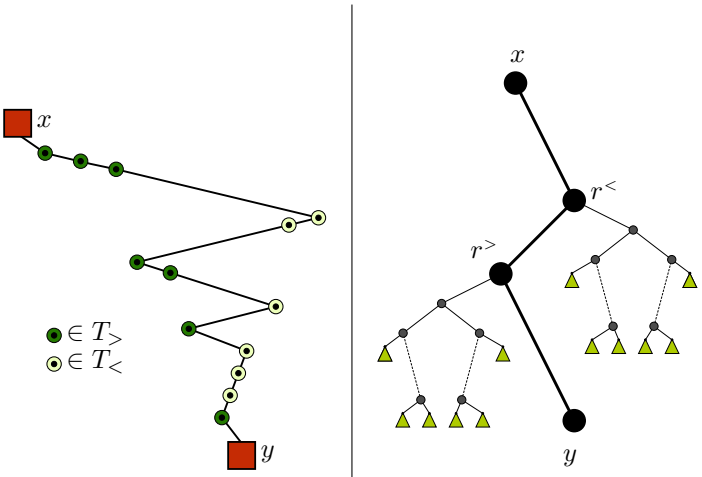
Proof. Letting $T_>$ to be the set of all nodes in tendon $\tau_{x,y}$ whose left-child is also in the tendon, and $T_<$ to be the set of all nodes in tendon $\tau_{x,y}$ whose right-child is also in the tendon yields the statement of the lemma. □

Deque-BST. It is straightforward to implement double ended queues (deques) in the BST model. We implement the deques for storing $T_>$ and $T_<$ symmetrically. LeftDeque-BST is a BST data structure storing the set of nodes in $T_>$ and RightDeque-BST is a BST data structure symmetric to LeftDeque-BST storing the set of nodes in $T_<$. We denote the roots of LeftDeque-BST and RightDeque-BST by $r^>$ and $r^<$ respectively. Because they are symmetric structures, we only describe LeftDeque-BST. Any node x to be pushed into a LeftDeque-BST is given as the parent of the $r^>$. Similarly, whenever a node is popped from LeftDeque-BST, it becomes the parent of $r^>$. LeftDeque-BST supports the following operations in constant amortized time: push-min$(T_>, x)$, push-max$(T_>, x)$, pop-min$(T_>)$, pop-max$(T_>)$.

2.1 Tendon-BST

We now present a BST data structure, Tendon-BST, which supports the following operations on a given tendon $\tau_{x,y}$, where $x' = \mathsf{parent}_T(x)$ and $y' = \mathsf{parent}_T(y)$, $\tau_{x',y} \leftarrow \mathsf{AddTop}(\tau_{x,y})$, $\tau_{x,y} \leftarrow \mathsf{AddBottom}(\tau_{x,y'}, y)$, $\tau_{x,y} \leftarrow \mathsf{RemoveTop}(\tau_{x',y})$, $\tau_{x,y'} \leftarrow \mathsf{RemoveBottom}(\tau_{x,y})$.

Implementation. We implement the Tendon-BST operations using LeftDeque-BST and RightDeque-BST. See Figure 2b. Nodes x, $r^>$, $r^<$, and y form a path in Tendon-BST where node x is an ancestor of nodes $r^>$, $r^<$, and y; and node y is a descedant of nodes x, $r^>$, and $r^<$. There are four such possible paths and the particular one formed depends on the key values of x and y, and the relationship between node y and its parent in T. These invariants imply that the distance between x and y is 3. When we need to insert a node into the tendon, we perform a constant number of rotations to preserve the invariants and position the node appropriately as the parent of $r^>$ or $r^<$. We then call the appropriate LeftDeque-BST or RightDeque-BST operation. Removing a node from the tendon is symmetric. Because deques (LeftDeque-BST, RightDeque-BST) can be implemented in constant amortized time per operation, and Tendon-BST performs a constant number of unit-cost operations in addition to one deque operation, it supports all of its operations in constant amortized time.



(a) $\tau_{x,y}$ and sets $T_>$ and $T_<$ (b) Tendon-BST storing $\tau_{x,y}$.

Fig. 2. A tendon in Multifinger-BST T and how its stored using Tendon-BST

2.2 OneFinger-BST

At a high level, the OneFinger-BST data structure maintains the hand $H(T, F)$, which is of constant size, where each node corresponds to a pseudofinger in $P(T, F)$. For each pseudofinger x the parent pointer, $\mathsf{parent}(x)$, either points to another pseudofinger or the bottom of a tendon, and the child pointers, $\mathsf{left\text{-}child}(x)$, $\mathsf{right\text{-}child}(x)$, each point to either another pseudofinger, or the root of a knuckle, or the top of a tendon.

Intuitively, the Tendon-BST structure allows us to *compress* a tendon down to constant depth. Whenever an operation is to be done at a finger, OneFinger-BST *uncompresses* the 3 surrounding tendons until the elements at distance up to 3 from the finger are as in the original tree, then performs the operation. OneFinger-BST then reconstructs the hand structure locally, possibly changing the set of pseudofingers if needed, and *recompresses* all tendons using Tendon-BST. This is all done in amortized $\mathcal{O}(1)$ time because Tendon-BST operations used for decompression and recompression take amortized $\mathcal{O}(1)$ time and reconfiguring any constant size local subtree into any shape takes $\mathcal{O}(1)$ time in the worst case.

ADT. OneFinger-BST is a BST data structure supporting the following operations on a Multifinger-BST T with a set F of fingers $f_1, \ldots, f_{|F|}$, where $|F| = \mathcal{O}(1)$:

- $\mathsf{MoveToParent}(f_i)$: Move finger f_i to its parent in T, $\mathsf{parent}_T(f_i)$.
- $\mathsf{MoveToLeftChild}(f_i)$: Move finger f_i to its left-child in T, $\mathsf{left\text{-}child}_T(f_i)$.
- $\mathsf{MoveToRightChild}(f_i)$: Move finger f_i to its right-child in T, $\mathsf{right\text{-}child}_T(f_i)$.
- $\mathsf{RotateAt}(f_i)$: Rotate finger f_i with its parent in T, $\mathsf{parent}_T(f_i)$.

Implementation. We augment each node with a $\mathcal{O}(|F|)$ bit field to store the type of the node and the fingers currently on the node.

All the OneFinger-BST operations take as input the finger they are to be performed on. We first do a brute force search using the augmented bits mentioned above to find the node pointed to by the input finger. Note that all such fingers will be within a $\mathcal{O}(1)$ distance from the root . We then perform the operation as well as the relevant updates to the tree to reflect the changes in $H(T, F)$. Specifically, in order to perform the operation, we extract the relevant nodes from the surrounding tendons of the finger by calling the appropriate Tendon-BST functions. We perform the operation and update the nodes to reflect the structural changes to the hand structure. Then we insert the tendon nodes back into their corresponding tendons using the appropriate Tendon-BST functions.

Theorem 7. *Given any Multifinger-BST data structure A, where $op_A[j]$ is the jth operation performed by A,* OneFinger-BST(A) *is a BST data structure such that, for any k, given k operations $(op_A[1], \ldots, op_A[k])$ online,* OneFinger-BST(A) *simulates them in $C_2 \cdot k$ total time for some constant C_2 that depends on the number of fingers used by A. If A is a real-world BST data structure, then so is* OneFinger-BST(A).

Proof. Note that before $op_M[1]$, all the fingers are initialized to the root of the tree and therefore the potentials associated with the deques of the tendons is zero. All finger movements and rotations are performed using OneFinger-BST operations. Each operation requires one finger movement or rotation and at most a constant number of Tendon-BST operations, as well as the time it takes to traverse between fingers. Because the time spent traversing between a constant number of fingers is at most a constant, this implies that OneFinger-BST(M) simulates operations $(op_M[1], \ldots, op_M[k])$ in $C_2 \cdot k$ time for any k. □

3 Multifinger-BST with Buffers

In our model, each node in the tree is allowed to store $\mathcal{O}(\log n)$ bits of augmented data. In this section, we show how to use this $\mathcal{O}(n \log n)$ collective bits of data to implement a traversable "buffer" data structure. More precisely, we show how to augment any Multifinger-BST data structure into a structure called Buf-MFBST supporting buffer operations.

Definition 8. *A* buffer *is a sequence $b_1, b_2, \ldots, b_n$ of cells, where each cell can store $\mathcal{O}(\log n)$ bits of data. The buffer can be traversed by a constant number of buffer-fingers, each initially on cell b_1, and each movable forwards or backwards one cell at a time.*

ADT. In addition to Multifinger-BST operations, Buf-MFBST supports the following operations on any buffer-finger bf of a buffer: bf.PreviousCell(), bf.NextCell(), bf.ReadCell(), bf.WriteCell(d).

Implementation. We store the jth buffer cell in the jth node in the in-order traversal of the tree. PreviousCell() and NextCell() are performed by traversing to the previous or next node respectively in the in-order traversal of the tree.

Lemma 9. *Given a tree T, traversing it in-order (or symmetrically in reverse-in-order) with a finger, interleaved with r rotation operations performed by other fingers, takes $\mathcal{O}(n + r)$ time.*

Proof. The cost of traversing the tree in-order is at most $2n$. Each rotation performed in between the in-order traversal operations can increase the total length of any path corresponding to a subsequence of the in-order traversal by at most one. Thus, the cost of traversing T in-order, interleaved with r rotation operations, is $\mathcal{O}(n + r)$. □

Tree state. We present an augmentation in the Multifinger-BST model, which we refer to as TSB, such that given any Multifinger-BST data structure M, TSB augments M with a tree state buffer. The following operations are supported on a tree state buffer: SaveState(): save the current state of the tree on the tree state buffer, LoadState(): transform the current tree to the state stored in the tree state buffer. Let the encoding of the tree state be a sequence of operations performed by a linear time algorithm LeftifyTree(T) that transforms the tree into a left path. There are numerous folklore linear time implementations of such an algorithm. We can save the state of T by calling LeftifyTree(T) and recording the performed operations in the tree state buffer. To load a tree state in the tree state buffer, we call LeftifyTree(T) then undo all the operations in the tree state buffer. Note that TSB maintains up to n cells but we may need to store more data. We can either pack more data into each cell or use multiple copies of TSB. We can also apply TSB to itself to allow for multiple buffers.

Lemma 10. *Given any Multifinger-BST data structure M with k fingers, TSB(M) is a Multifinger-BST data structure with $\mathcal{O}(k)$ fingers such that the number of operations performed to execute* SaveState() *or* LoadState() *is $\mathcal{O}(n)$.*

Proof. Because LeftifyTree(T) runs in linear time, there can be only $\mathcal{O}(n)$ rotations, and by Lemma 9 the running time of both operations is $\mathcal{O}(n)$. □

4 Combo-MFBST and Combo-BST

Given two online BST data structures $\mathcal{A}_0$ and $\mathcal{A}_1$, let $\mathcal{T}_{X'}(\mathcal{A}_0, T, X)$ be any well-behaved upper bound with overhead $f(n) \geq n$ on the running time of $\mathcal{A}_0$, and let $\mathcal{T}_{X'}(\mathcal{A}_1, T, X)$ be any well-behaved upper bound with overhead $f(n) \geq n$ on the running time of $\mathcal{A}_1$ on a contiguous subsequence X' of X, for any online access sequence X and initial tree T. Then Combo-MFBST = Combo-MFBST($\mathcal{A}_0, \mathcal{A}_1, f(n)$) is defined as follows. It uses a tree state buffer ST_* implemented as a TSB. It stores the initial tree state T in ST_* by calling SaveState() before executing any accesses. Then, Combo-MFBST executes any online access sequence in rounds by alternating between emulating $\mathcal{A}_0$ and $\mathcal{A}_1$. Specifically, each round consists of $C_3 \cdot f(n)$ operations that execute the access

sequence using BST data structure $\mathcal{A}_\mu$, for $\mu \in \{0, 1\}$, always starting from the initial tree T. When the operation limit $C_3 \cdot f(n)$ gets reached, say in the middle of executing access x_i, Combo-MFBST transforms the tree back to its initial state T by calling LoadState() on ST_*; and toggles the active BST data structure by setting μ to $1 - \mu$. The next round re-runs access x_i, this time on the opposite BST data structure. By picking a suitably large C_3, we ensure that every round completes at least one access, and thus no access gets executed by the same BST data structure in more than one round.

Lemma 11. *Given two BST data structures, $\mathcal{A}_0$ and $\mathcal{A}_1$,* Combo-MFBST $=$ Combo-MFBST$(\mathcal{A}_0, \mathcal{A}_1, f(n))$ *for any $f(n)$ is a Multifinger-BST data structure with $\mathcal{O}(1)$ fingers. Furthermore, if $\mathcal{A}_0$ and $\mathcal{A}_1$ are real-world BST data structures, then so is* Combo-MFBST.

Proof. Combo-MFBST$(\mathcal{A}_0, \mathcal{A}_1, f(n))$ has one tree state buffer which has $\mathcal{O}(1)$ fingers by Lemma 10. Because TSB augments each node with at most $\mathcal{O}(\log n)$ bits and Combo-MFBST uses only a constant number of registers each of size $\mathcal{O}(\log n)$ bits, the lemma follows. □

Definition 12. *Given two online BST data structures $\mathcal{A}_0$ and $\mathcal{A}_1$,* Combo-BST$(\mathcal{A}_0, \mathcal{A}_1, f(n))$ = OneFinger-BST(Combo-MFBST$(\mathcal{A}_0, \mathcal{A}_1, f(n))$).

5 Analysis

Theorem 13. *Given two online BST data structures $\mathcal{A}_0$ and $\mathcal{A}_1$, let $\mathcal{T}_{X'}(\mathcal{A}_0, T, X)$ and $\mathcal{T}_{X'}(\mathcal{A}_1, T, X)$ be well-behaved amortized upper bounds with overhead $f(n) \geq n$ on the running time of $\mathcal{A}_0$ and $\mathcal{A}_1$, respectively, on a contiguous subsequence X' of X for any online access sequence X and initial tree T Then there exists an online BST data structure,* Combo-BST $=$ Combo-BST$(\mathcal{A}_0, \mathcal{A}_1, f(n))$ *such that*

$$\mathcal{T}_{X'}(\text{Combo-BST}, T, X) = \mathcal{O}(\min(\mathcal{T}_{X'}(\mathcal{A}_0, T, X), \mathcal{T}_{X'}(\mathcal{A}_1, T, X)) + f(n)).$$

If $\mathcal{A}_0$ and $\mathcal{A}_1$ are real-world BST data structures, so is Combo-BST.

Proof. Let $\mathcal{A}_{\min} = \mathcal{A}_0$ if $\mathcal{T}_{X'}(\mathcal{A}_0, T, X) \leq \mathcal{T}_{X'}(\mathcal{A}_1, T, X)$, and $\mathcal{A}_{\min} = \mathcal{A}_1$ otherwise. Let $X'' = X'_1 \cdot \ldots \cdot X'_k$ be the subsequence of X' executed by $\mathcal{A}_{\min}$. If the Combo-MFBST terminates after at most $2k+1$ rounds (k of them performed by $\mathcal{A}_{\min}$), then taking into account the TSB traversal at every round which takes $\mathcal{O}(n) = \mathcal{O}(f(n))$ time by Lemma 10 we have

$$\mathcal{T}_{X'}(\text{Combo-MFBST}, T, X) = \mathcal{O}(k \cdot f(n) + k \cdot n). \tag{1}$$

Now we need to bound k. Each round but the last one runs for $C_3 \cdot f(n)$ steps exactly, and in particular, $\mathcal{T}(\mathcal{A}_{\min}, T, X'_j) \geq C_3 \cdot f(n)$ for all $j < k$, that is, it might need more steps to complete the last access of X'_j. Summing over all j, we get $C_3(k-1)f(n) \leq \sum_{j=1}^{k-1} \mathcal{T}(\mathcal{A}_{\min}, T, X'_j) \leq C_0 \sum_{j=1}^{k-1} \mathcal{T}_{X'_j}(\mathcal{A}_{\min}, T, X) + C_1(k-1)f(n) \leq C_0 \mathcal{T}_{X'}(\mathcal{A}_{\min}, T, X) + C_1(k-1)f(n)$ by well-behavedness, the definition of $\mathcal{T}$ and the fact that X'_js are disjoint subsets of X'. Therefore, setting $C_3 > C_1$

yields $k-1 \leq \mathcal{O}(\mathcal{T}_{X'}(\mathcal{A}_{\min}, T, X)/f(n))$. Combining with Equation 1, we obtain the desired bound for Combo-MFBST. By Lemma 11, Combo-MFBST is a real-world Multifinger-BST data structure. Applying OneFinger-BST (Theorem 5) yields our result. □

References

1. Allen, B., Ian Munro, J.: Self-organizing binary search trees. Journal of the ACM 25(4), 526–535 (1978)
2. Bayer, R.: Symmetric binary B-Trees: Data structure and maintenance algorithms. Acta Informatica 1, 290–306 (1972)
3. Bose, P., Collette, S., Fagerberg, R., Langerman, S.: De-amortizing binary search trees. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 121–132. Springer, Heidelberg (2012)
4. Bose, P., Douïeb, K., Dujmovic, V., Howat, J.: Layered working-set trees. Algorithmica 63(1-2), 476–489 (2012)
5. Bose, P., Douïeb, K., Iacono, J., Langerman, S.: The power and limitations of static binary search trees with lazy finger. arXiv:1304.6897 (2013)
6. Cole, R.: On the dynamic finger conjecture for splay trees. Part II: The proof. SIAM Journal on Computing 30(1), 44–85 (2000)
7. Cole, R., Mishra, B., Schmidt, J.P., Siegel, A.: On the dynamic finger conjecture for splay trees. Part I: Splay sorting log n-block sequences. SIAM Journal on Computing 30(1), 1–43 (2000)
8. Demaine, E.D., Harmon, D., Iacono, J., Pătraşcu, M.: Dynamic optimality — almost. SIAM Journal on Computing 37(1), 240–251 (2007)
9. Demaine, E.D., Iacono, J., Langerman, S., Özkan, Ö.: Combining binary search trees. arXiv:1304.7604 (2013)
10. Demaine, E.D., Langerman, S., Price, E.: Confluently persistent tries for efficient version control. Algorithmica 57(3), 462–483 (2010)
11. Derryberry, J.C.: Adaptive Binary Search Tree. PhD thesis, CMU (2009)
12. Derryberry, J.C., Sleator, D.D.: Skip-splay: Toward achieving the unified bound in the BST model. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 194–205. Springer, Heidelberg (2009)
13. Iacono, J.: Improved upper bounds for pairing heaps. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 32–45. Springer, Heidelberg (2000)
14. Iacono, J.: Alternatives to splay trees with $O(\log n)$ worst-case access times. In: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 516–522 (2001)
15. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM 32(3), 652–686 (1985)
16. Sleator, D.D., Tarjan, R.E., Thurston, W.P.: Rotation distance, triangulations, and hyperbolic geometry. In: Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC), pp. 122–135 (1986)
17. Wang, C.C., Derryberry, J., Sleator, D.D.: $O(\log \log n)$-competitive dynamic binary search trees. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 374–383 (2006)

The Two-Handed Tile Assembly Model Is Not Intrinsically Universal*

Erik D. Demaine[1], Matthew J. Patitz[2,**], Trent A. Rogers[3], Robert T. Schweller[4,**], Scott M. Summers[5], and Damien Woods[6,***]

[1] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA
edemaine@mit.edu

[2] Department of Computer Science and Computer Engineering, University of Arkansas
patitz@uark.edu

[3] Department of Mathematical Sciences, University of Arkansas
tar003@email.uark.edu

[4] Department of Computer Science, University of Texas–Pan American, Edinburg, TX, 78539, USA
rtschweller@utpa.edu

[5] Department of Computer Science and Software Engineering, University of Wisconsin–Platteville, Platteville, WI 53818, USA
summerss@uwplatt.edu

[6] Computer Science, California Institute of Technology
woods@caltech.edu

Abstract. In this paper, we study the intrinsic universality of the well-studied Two-Handed Tile Assembly Model (2HAM), in which two "supertile" assemblies, each consisting of one or more unit-square tiles, can fuse together (self-assemble) whenever their total attachment strength is at least the global temperature τ. Our main result is that for all $\tau' < \tau$, each temperature-τ' 2HAM tile system cannot simulate at least one temperature-τ 2HAM tile system. This impossibility result proves that the 2HAM is not intrinsically universal, in stark contrast to the simpler abstract Tile Assembly Model which was shown to be intrinsically universal (*The tile assembly model is intrinsically universal*, FOCS 2012). On the positive side, we prove that, for every fixed temperature $\tau \geq 2$, temperature-τ 2HAM tile systems are intrinsically universal: for each τ there is a single universal 2HAM tile set U that, when appropriately initialized, is capable of simulating the behavior of any temperature τ 2HAM tile system. As a corollary of these results we find an infinite set of infinite hierarchies of 2HAM systems with strictly increasing power within each hierarchy. Finally, we show how to construct, for

* A full version of this paper will appear on the arXiv.

** This author's research was supported in part by National Science Foundation Grant CCF-1117672.

*** This author's research was supported by National Science Foundation grants 0832824 (The Molecular Programming Project), CCF-1219274, and CCF-1162589.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 400–412, 2013.

each τ, a temperature-τ 2HAM system that simultaneously simulates all temperature-τ 2HAM systems.

1 Introduction

Self-assembly is the process through which unorganized, simple, components automatically coalesce according to simple local rules to form some kind of target structure. It sounds simple, but the end result can be extraordinary. For example, researchers have been able to self-assemble a wide variety of structures experimentally at the nanoscale, such as regular arrays [28], fractal structures [13, 24], smiling faces [23], DNA tweezers [29], logic circuits [21], neural networks [22], and molecular robots [18]. These examples are fundamental because they demonstrate that self-assembly can, in principle, be used to manufacture specialized geometrical, mechanical and computational objects at the nanoscale. Potential future applications of nanoscale self-assembly include the production of smaller, more efficient microprocessors and medical technologies that are capable of diagnosing and even treating disease at the cellular level.

Controlling nanoscale self-assembly for the purposes of manufacturing atomically precise components will require a bottom-up, hands-off strategy. In other words, the self-assembling units themselves will have to be "programmed" to direct themselves to do the right thing–efficiently and correctly. Thus, it is necessary to study the extent to which the process of self-assembly can be controlled in an algorithmic sense.

In 1998, Erik Winfree [27] introduced the abstract Tile Assembly Model (aTAM), an over-simplified discrete mathematical model of algorithmic DNA nanoscale self-assembly pioneered by Seeman [25]. The aTAM essentially augments classical Wang tiling [26] with a mechanism for sequential "growth" of a tiling (in Wang tiling, only the existence of a valid, mismatch-free tiling is considered and not the order of tile placement). In the aTAM, the fundamental components are un-rotatable, but translatable square "tile types" whose sides are labeled with (alpha-numeric) glue "colors" and (integer) "strengths". Two tiles that are placed next to each other *interact* if the glue colors on their abutting sides match, and they *bind* if the strengths on their abutting sides match and sum to at least a certain (integer) "temperature". Self-assembly starts from a "seed" tile type and proceeds nondeterministically and asynchronously as tiles bind to the seed-containing-assembly. Despite its deliberate over-simplification, the aTAM is a computationally expressive model. For example, Winfree [27] proved that it is Turing universal, which implies that self-assembly can be directed by a computer program.

In this paper, we work in a generalization of the aTAM, called the *two-handed* [3] (a.k.a., hierarchical [5], q-tile [6], polyomino [17]) abstract Tile Assembly Model (2HAM). A central feature of the 2HAM is that, unlike the aTAM, it allows two "supertile" assemblies, each consisting of one or more tiles, to fuse together. For two such assemblies to bind, they should not "sterically hinder" each other, and they should have a sufficient number of matching glues distributed along the interface

where they meet. Hence the model includes notions of local interactions (individual glues) and non-local interactions (large assemblies coming together). In the 2HAM, an assembly of tiles is producible if it is either a single tile, or if it results from the stable combination of two other producible assemblies.

We study the *intrinsic universality* in the 2HAM. Intrinsic universality uses a special notion of simulation, where the simulator preserves the dynamics of the simulated system. In the field of cellular automata, the topic of intrinsic universality has given rise to a rich theory [2, 4, 7, 8, 12, 19, 20] and indeed has also been studied in Wang tiling [14–16] and tile self-assembly [10,11]. The aTAM has been shown to be intrinsically universal [10], meaning that there is a single set of tiles U that works at temperature 2, and when appropriately initialized, is capable of simulating the behavior of an arbitrary aTAM tile assembly system. Modulo rescaling, this single tile set U represents the full power and expressivity of the entire aTAM model, at any temperature. Here, we ask whether there such a universal tile set for the 2HAM.

The theoretical power of non-local interaction in the 2HAM has been the subject of recent research. For example, Doty and Chen [5] proved that, surprisingly, $N \times N$ squares do not self-assemble any faster in so-called *partial order* 2HAM systems than they do in the aTAM, despite being able to exploit massive parallelism. More recently, Cannon, et al. [3], while comparing the abilities of the 2HAM and the aTAM, proved three main results, which seem to suggest that the 2HAM is at least as powerful as the aTAM: (1) non-local binding in the 2HAM can dramatically reduce the tile complexity (i.e., minimum number of unique tile types required to self-assemble a shape) for certain classes of shapes; (2) the 2HAM can simulate the aTAM in the following sense: for any aTAM tile system $\mathcal{T}$, there is a corresponding 2HAM tile system $\mathcal{S}$, which simulates the exact behavior—modulo connectivity—of $\mathcal{T}$, at scale factor 5; (3) the problem of verifying whether a 2HAM system uniquely produces a given assembly is coNP-complete (for the aTAM this problem is decidable in polynomial time [1]).

Main results. In this paper, we ask if the 2HAM is *intrinsically universal*: does there exist a "universal" 2HAM tile set U that, when appropriately initialized, is capable of simulating the behavior of an arbitrary 2HAM tile system? A positive answer would imply that such a tile set U has the ability to model the capabilities of all 2HAM systems.[1] Our first main result, Theorem 1, says that the 2HAM is *not* intrinsically universal, which means that the 2HAM is incapable of simulating itself. This statement stands in stark contrast to the case of the aTAM, which was recently shown to be intrinsically universal by Doty, Lutz, Patitz, Schweller, Summers and Woods [10]. Specifically, we show that for any temperature τ, there is a temperature τ 2HAM system that cannot be simulated by any temperature $\tau' < \tau$ 2HAM system. It is worthy of note that, in order to prove this result, we use a simple, yet novel combinatorial argument, which as

[1] Note that the above simulation result of Cannon et al. does not imply that the 2HAM is intrinsically universal because (a) it is for 2HAM simulating aTAM, and (b) it is an example of a "for all, there exists..." statement, whereas intrinsic universality is a "there exists, for all..." statement.

far as we are aware of, is the first lower bound proof in the 2HAM that does not use an information-theoretic argument. In our proof of Theorem 1 we show that the 2HAM cannot simulate massively cooperative binding, where the number of cooperative bindings is larger than the temperature of the simulator).

Our second main result, Theorem 3, is positive: we show, via constructions, that the 2HAM *is* intrinsically universal for fixed temperature, that is, the temperature τ 2HAM can simulate the temperature τ 2HAM. So although our impossibility result tells us that the 2HAM can not simulate "too much" cooperative binding, our positive result tells us it can indeed simulate *some* cooperative binding: an amount exactly equal to the temperature of the simulator.

As an immediate corollary of these results, we get a separation between classes of 2HAM tile systems based on their temperatures. That is, we exhibit an infinite hierarchy of 2HAM systems, of strictly-increasing temperature, that cannot be simulated by lesser temperature systems but can downward simulate lower temperature systems. Indeed, we exhibit an infinite number of such hierarchies in Theorem 4. Thus, as was suggested as future work in [10], and as has been shown in the theory of cellular automata [8], we use the notion of intrinsic universality to classify, and separate, 2HAM systems via their simulation ability.

As noted above, we show that temperature τ 2HAM systems are intrinsically universal. We actually show this for two different, seemingly natural, notions of simulation (called *simulation* and *strong simulation*), showing trade-offs between, and even within, these notions of simulation. For both notions of simulation, we show tradeoffs between scale factor, number of tile types, and complexity of the initial configuration. Finally, we show how to construct, for each τ, a temperature-τ 2HAM system that simultaneously simulates all temperature-τ 2HAM systems. We finish with a conjecture:

Conjecture 1. There exists $c \in \mathbb{N}$, such that for each $\tau \geq c$, temperature τ 2HAM systems do not strongly simulate Temperature $\tau - 1$ 2HAM systems.

2 Definitions

2.1 Informal Definition of 2HAM

The 2HAM [6, 9] is a generalization of the aTAM in that it allows for two assemblies, both possibly consisting of more than one tile, to attach to each other. Since we must allow that the assemblies might require translation before they can bind, we define a *supertile* to be the set of all translations of a τ-stable assembly, and speak of the attachment of supertiles to each other, modeling that the assemblies attach, if possible, after appropriate translation. We now give a brief, informal, sketch of the 2HAM.

A *tile type* is a unit square with four sides, each having a *glue* consisting of a *label* (a finite string) and *strength* (a non-negative integer). We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile*. A *supertile* is (the set of all translations of) a positioning of tiles on the integer lattice $\mathbb{Z}^2$. Two adjacent tiles in a supertile

interact if the glues on their abutting sides are equal and have positive strength. Each supertile induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The supertile is τ*-stable* if every cut of its binding graph has strength at least τ, where the weight of an edge is the strength of the glue it represents. That is, the supertile is stable if at least energy τ is required to separate the supertile into two parts. A 2HAM *tile assembly system* (TAS) is a pair $\mathcal{T} = (T, \tau)$, where T is a finite tile set and τ is the *temperature*, usually 1 or 2. Given a TAS $\mathcal{T} = (T, \tau)$, a supertile is *producible*, written as $\alpha \in \mathcal{A}[\mathcal{T}]$ if either it is a single tile from T, or it is the τ-stable result of translating two producible assemblies without overlap. A supertile α is *terminal*, written as $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$ if for every producible supertile β, α and β cannot be τ-stably attached. A TAS is *directed* if it has only one terminal, producible supertile.[2]

2.2 Definitions for Simulation

In this subsection, we formally define what it means for one 2HAM TAS to "simulate" another 2HAM TAS. For a tileset T, let A^T and $\tilde{A}^T$ denote the set of all assemblies over T and all supertiles over T respectively. Let $A^T_{<\infty}$ and $\tilde{A}^T_{<\infty}$ denote the set of all finite assemblies over T and all finite supertiles over T respectively.

In what follows, let U be a tile set. An *m-block assembly*, or *macrotile*, over tile set U is a partial function $\gamma : \mathbb{Z}_m \times \mathbb{Z}_m \dashrightarrow U$, where $\mathbb{Z}_m = \{0, 1, \ldots m-1\}$. Let B^U_m be the set of all m-block assemblies over U. The m-block with no domain is said to be *empty*. For an arbitrary assembly $\alpha \in A^U$ define $\alpha^m_{x,y}$ to be the m-block defined by $\alpha^m_{x,y}(i,j) = \alpha(mx+i, my+j)$ for $0 \leq i, j < m$.

For a partial function $R : B^U_m \dashrightarrow T$, define the *assembly representation function* $R^* : A^U \dashrightarrow A^T$ such that $R^*(\alpha) = \beta$ if and only if $\beta(x,y) = R(\alpha^m_{x,y})$ for all $x, y \in \mathbb{Z}^2$. Further, α is said to map *cleanly* to β under R^* if either (1) for all non empty blocks $\alpha^m_{x,y}$, $(x+u, y+v) \in \text{dom}\ \beta$ for some $u, v \in \{-1, 0, 1\}$ such that $u^2 + v^2 < 2$, or (2) α has at most one non-empty m-block $\alpha^m_{x,y}$. In other words, we allow for the existence of simulator "fuzz" directly north, south, east or west of a simulator macrotile, but we exclude the possibility of diagonal fuzz.

For a given *assembly representation function* R^*, define the *supertile representation function* $\tilde{R} : \tilde{A}^U \dashrightarrow \mathcal{P}(A^T)$ such that $\tilde{R}(\tilde{\alpha}) = \{R^*(\alpha) | \alpha \in \tilde{\alpha}\}$. $\tilde{\alpha}$ is said to *map cleanly* to $\tilde{R}(\tilde{\alpha})$ if $\tilde{R}(\tilde{\alpha}) \in \tilde{A}^T$ and α maps cleanly to $R^*(\alpha)$ for all $\alpha \in \tilde{\alpha}$.

In the following definitions, let $\mathcal{T} = (T, S, \tau)$ be a 2HAM TAS and, for some initial configuration $S_{\mathcal{T}}$, that depends on $\mathcal{T}$, let $\mathcal{U} = (U, S_{\mathcal{T}}, \tau')$ be a 2HAM TAS, and let R be an m-block representation function $R : B^U_m \dashrightarrow T$.

Definition 1. *We say that* $\mathcal{U}$ *and* $\mathcal{T}$ *have* equivalent productions *(at scale factor* m*), and we write* $\mathcal{U} \Leftrightarrow_R \mathcal{T}$ *if the following conditions hold:*

1. $\left\{\tilde{R}(\tilde{\alpha}) | \tilde{\alpha} \in \mathcal{A}[\mathcal{U}]\right\} = \mathcal{A}[\mathcal{T}]$.

[2] We do not use this definition in this paper but have included it for the sake of completeness.

2. *For all* $\tilde{\alpha} \in \mathcal{A}[\mathcal{U}]$, $\tilde{\alpha}$ *maps cleanly to* $\tilde{R}(\tilde{\alpha})$

Definition 2. *We say that* $\mathcal{T}$ follows $\mathcal{U}$ *(at scale factor* m*), and we write* $\mathcal{T} \dashv_R \mathcal{U}$ *if, for any* $\tilde{\alpha}, \tilde{\beta} \in \mathcal{A}[\mathcal{U}]$ *such that* $\tilde{\alpha} \to_{\mathcal{U}}^{1} \tilde{\beta}$, $\tilde{R}(\tilde{\alpha}) \to_{\mathcal{T}}^{\leq 1} \tilde{R}\left(\tilde{\beta}\right)$.

Definition 3. *We say that* $\mathcal{U}$ weakly models $\mathcal{T}$ *(at scale factor* m*), and we write* $\mathcal{U} \models_R^- \mathcal{T}$ *if, for any* $\tilde{\alpha}, \tilde{\beta} \in \mathcal{A}[\mathcal{T}]$ *such that* $\tilde{\alpha} \to_{\mathcal{T}}^{1} \tilde{\beta}$*, for all* $\tilde{\alpha}' \in \mathcal{A}[\mathcal{U}]$ *such that* $\tilde{R}(\tilde{\alpha}') = \tilde{\alpha}$*, there exists an* $\tilde{\alpha}'' \in \mathcal{A}[\mathcal{U}]$ *such that* $\tilde{R}(\tilde{\alpha}'') = \tilde{\beta}$, $\tilde{\alpha}' \to_{\mathcal{U}} \tilde{\alpha}''$*, and* $\tilde{\alpha}'' \to_{\mathcal{U}}^{1} \tilde{\beta}'$ *for some* $\tilde{\beta}' \in \mathcal{A}[\mathcal{U}]$ *with* $\tilde{R}\left(\tilde{\beta}'\right) = \tilde{\beta}$.

Definition 4. *We say that* $\mathcal{U}$ strongly models $\mathcal{T}$ *(at scale factor* m*), and we write* $\mathcal{U} \models_R^+ \mathcal{T}$ *if for any* $\tilde{\alpha}$, $\tilde{\beta} \in \mathcal{A}[\mathcal{T}]$ *such that* $\tilde{\gamma} \in C^{\tau}_{\tilde{\alpha},\tilde{\beta}}$*, then for all* $\tilde{\alpha}', \tilde{\beta}' \in \mathcal{A}[\mathcal{U}]$ *such that* $\tilde{R}(\tilde{\alpha}') = \tilde{\alpha}$ *and* $\tilde{R}\left(\tilde{\beta}'\right) = \tilde{\beta}$*, it must be that there exist* $\tilde{\alpha}'', \tilde{\beta}'', \tilde{\gamma}' \in \mathcal{A}[\mathcal{U}]$*, such that* $\tilde{\alpha}' \to_{\mathcal{U}} \tilde{\alpha}''$, $\tilde{\beta}' \to_{\mathcal{U}} \tilde{\beta}''$, $\tilde{R}(\tilde{\alpha}'') = \tilde{\alpha}$, $\tilde{R}\left(\tilde{\beta}''\right) = \tilde{\beta}$, $\tilde{R}(\tilde{\gamma}') = \tilde{\gamma}$*, and* $\tilde{\gamma}' \in C^{\tau'}_{\tilde{\alpha}'',\tilde{\beta}''}$.

Definition 5. *Let* $\mathcal{U} \Leftrightarrow_R \mathcal{T}$ *and* $\mathcal{T} \dashv_R \mathcal{U}$.

1. $\mathcal{U}$ simulates $\mathcal{T}$ *(at scale factor* m*) if* $\mathcal{U} \models_R^- \mathcal{T}$.
2. $\mathcal{U}$ strongly simulates $\mathcal{T}$ *(at scale factor* m*) if* $\mathcal{U} \models_R^+ \mathcal{T}$.

For simulation, we require that when a simulated supertile $\tilde{\alpha}$ may grow, via one combination attachment, into a second supertile $\tilde{\beta}$, then any simulator supertile that maps to $\tilde{\alpha}$ must also grow into a simulator supertile that maps to $\tilde{\beta}$. The converse should also be true.

For strong simulation, in addition to requiring that all supertiles mapping to $\tilde{\alpha}$ must be capable of growing into a supertile mapping to $\tilde{\beta}$ when $\tilde{\alpha}$ can grow into $\tilde{\beta}$ in the simulated system, we further require that this growth can take place by the attachment of *any* supertile mapping to $\tilde{\gamma}$, where $\tilde{\gamma}$ is the supertile that attaches to $\tilde{\alpha}$ to get $\tilde{\beta}$.

2.3 Intrinsic Universality

Let REPR denote the set of all m-block (or macrotile) representation functions. Let $\mathfrak{C}$ be a class of tile assembly systems, and let U be a tile set. We say U is *intrinsically universal* for $\mathfrak{C}$ if there are computable functions $\mathcal{R} : \mathfrak{C} \to \mathsf{REPR}$ and $\mathcal{S} : \mathfrak{C} \to \left(A^U_{<\infty} \to \mathbb{N} \cup \{\infty\}\right)$, and a $\tau' \in \mathbb{Z}^+$ such that, for each $\mathcal{T} = (T, S, \tau) \in \mathfrak{C}$, there is a constant $m \in \mathbb{N}$ such that, letting $R = \mathcal{R}(\mathcal{T})$, $S_{\mathcal{T}} = \mathcal{S}(\mathcal{T})$, and $\mathcal{U}_{\mathcal{T}} = (U, S_{\mathcal{T}}, \tau')$, $\mathcal{U}_{\mathcal{T}}$ simulates $\mathcal{T}$ at scale m and using macrotile representation function R. That is, $\mathcal{R}(\mathcal{T})$ gives a representation function R that interprets macrotiles (or m-blocks) of $\mathcal{U}_{\mathcal{T}}$ as assemblies of $\mathcal{T}$, and $\mathcal{S}(\mathcal{T})$ gives the initial state used to create the necessary macrotiles from U to represent $\mathcal{T}$ subject to the constraint that no macrotile in $S_{\mathcal{T}}$ can be larger than a single $m \times m$ square.

3 The 2HAM Is Not Intrinsically Universal

In this section, we prove the main result of this paper: there is no universal 2HAM tile set that, when appropriately initialized, is capable of simulating an arbitrary 2HAM system. That is, we prove that the 2HAM, unlike the aTAM, is not intrinsically universal.

Theorem 1. *The 2HAM is not intrinsically universal.*

In order to prove Theorem 1, we prove Theorem 2, which says that, for any claimed 2HAM simulator $\mathcal{U}$, that runs at temperature τ', there exists a 2HAM system, with temperature $\tau > \tau'$, that cannot be simulated by $\mathcal{U}$.

Theorem 2. *Let $\tau \in \mathbb{N}, \tau \geq 2$. For every tile set U, there exists a 2HAM TAS $\mathcal{T} = (T, S, \tau)$ such that for any initial configuration $S_{\mathcal{T}}$ and $\tau' \leq \tau - 1$, the 2HAM TAS $\mathcal{U} = (U, S_{\mathcal{T}}, \tau')$ does not simulate $\mathcal{T}$.*

The basic idea of the proof of Theorem 2 is to use Definitions 3 and 1 in order to exhibit two producible supertiles in $\mathcal{T}$, that do not combine in $\mathcal{T}$ because of a lack of total binding strength, and show that the supertiles that simulate them in $\mathcal{U}$ do combine in the (lower temperature) simulator $\mathcal{U}$. Then we argue that Definition 2 says that, because the simulating supertiles can combine in the simulator $\mathcal{U}$, then so too can the supertiles being simulated in the simulated system $\mathcal{T}$, which contradicts the fact that the two originally chosen supertiles from $\mathcal{T}$ do not combine in $\mathcal{T}$.

Proof. Our proof is by contradiction. Therefore, suppose, for the sake of obtaining a contradiction, that there exists a universal tile set U such that, for any 2HAM TAS $\mathcal{T} = (T, S, \tau)$, there exists an initial configuration $S_{\mathcal{T}}$ and $\tau' \leq \tau - 1$, such that $\mathcal{U} = (U, S_{\mathcal{T}}, \tau')$ simulates $\mathcal{T}$. Define $\mathcal{T} = (T, \tau)$ where T is the tile set defined in Figure 1, the default initial state is used, and $\tau > 1$. Let $\mathcal{U} = (U, S_{\mathcal{T}}, \tau')$ be the temperature $\tau' \leq \tau - 1$ 2HAM system, which uses tile set U and initial configuration $S_{\mathcal{T}}$ (depending on $\mathcal{T}$) to simulate $\mathcal{T}$ at scale factor m. Let $\tilde{R}$ denote the assembly replacement function that testifies to the fact that $\mathcal{U}$ simulates $\mathcal{T}$.

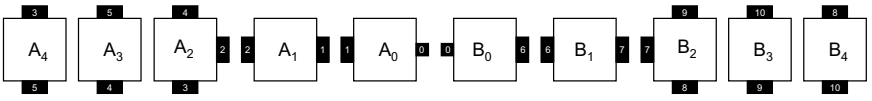


Fig. 1. The tile set for the proof of Theorem 2. Black rectangles represent strength-τ glues (labeled 1-8), and black squares represent the strength-1 glue (labeled 0).

We say that a supertile $\tilde{l} \in \mathcal{A}[\mathcal{T}]$ is a *left half-ladder* of height $h \in \mathbb{N}$ if it contains h tiles of the type A2 and $h-1$ tiles of type A3, arranged in a vertical column, plus τ tiles of each of the types A1 and A0. (An example of a left half-ladder is shown on the left in Figure 2. The dotted lines show positions at which

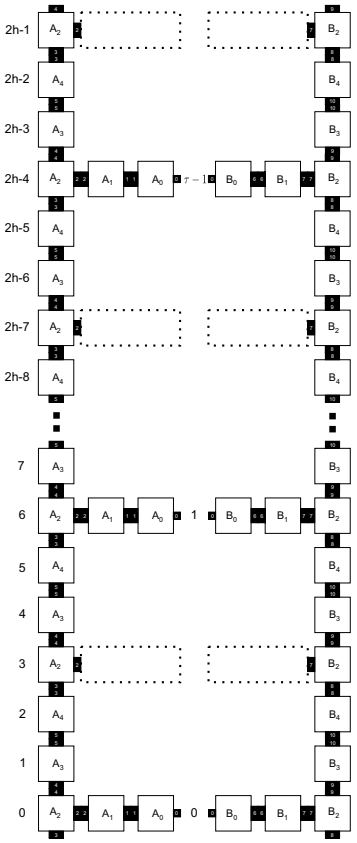


Fig. 2. Example half-ladders with τ rungs

tiles of type A1 and A0 could potentially attach, but since a half-ladder has exactly τ of each, only τ such locations have tiles.) Essentially, a left half-ladder consists of a single-tile-wide vertical column of height $2h-1$ with an A2 tile at the bottom and top, and those in between alternating between A3 and A2 tiles. To the east of exactly τ of the A2 tiles, an A1 tile is attached and to the east of each A1 tile, an A0 tile type is attached. These A1-A0 pairs, collectively, form the τ *rungs* of the left half-ladder. We can define *right* half-ladders similarly. A *right half-ladder* of height h is defined exactly the same way but using the tile types B3, B2, B1, and B0 and with rungs growing to the left of the vertical column. The east glue of A0 is a strength-1 glue matching the west glue of B0.

Let $LEFT \subseteq \mathcal{A}[\mathcal{T}]$ and $RIGHT \subseteq \mathcal{A}[\mathcal{T}]$ be the set of all left and right half-ladders of height h, respectively. Note that there are $\binom{h}{\tau}$ half-ladders of height h in $LEFT$ ($RIGHT$). Define, for each $\tilde{l} \in LEFT$, the *mirror image* of $\tilde{l}$ as the supertile $\bar{\tilde{l}} \in RIGHT$ such that $\bar{\tilde{l}}$ has rungs at the same positions as $\tilde{l}$.

For some $\tilde{l} \in LEFT$, we say that $\hat{\tilde{l}} \in \mathcal{A}[\mathcal{U}]$ is a *simulator* left half-ladder of height h if $\tilde{R}\left(\hat{\tilde{l}}\right) = \tilde{l}$. Note that $\hat{\tilde{l}}$ need not be unique. (One could even imagine

$\tilde{\tilde{l}}$ and $\tilde{\tilde{l}}'$ satisfying $\tilde{R}\left(\tilde{\tilde{l}}\right) = \tilde{l}$ and $\tilde{R}\left(\tilde{\tilde{l}}'\right) = \tilde{l}$ but $\tilde{\tilde{l}}$ and $\tilde{\tilde{l}}'$ only differ by a single tile!) The notation $C^{\tau}_{\tilde{\alpha},\tilde{\beta}}$ is defined as the set of all supertiles that result in the τ-stable combination of the supertiles $\tilde{\alpha}$ and $\tilde{\beta}$.

For some $\tilde{\tilde{r}} \in \mathcal{A}[\mathcal{U}]$, we say that $\tilde{\tilde{r}}$ is a *mate* of $\tilde{\tilde{l}}$ if $\tilde{R}\left(\tilde{\tilde{r}}\right) = \tilde{r} \in RIGHT$, where $\tilde{r} = \bar{\tilde{l}}$, $C^{\tau}_{\tilde{l},\tilde{r}} \neq \emptyset$ (they combine in $\mathcal{T}$), and $C^{\tau-1}_{\tilde{\tilde{l}},\tilde{\tilde{r}}} \neq \emptyset$ (they combine in $\mathcal{U}$). For a simulator left half-ladder $\tilde{\tilde{l}}$, we say that $\tilde{\tilde{l}}$ is *combinable* if $\tilde{\tilde{l}}$ has a mate. Part 1 of Definition 5 guarantees the existence of at least one combinable simulator left half-ladder for each left half-ladder. It is easy to see from Part 1 of Definition 5 that an arbitrary simulator left half-ladder need not be combinable, since by Definition 3, it may be a half-ladder $\tilde{\tilde{l}} \in \mathcal{A}[\mathcal{U}]$, which must first "grow into" a combinable left half-ladder $\tilde{\tilde{l}}'$ (analogous to $\tilde{\alpha}' \rightarrow_{\mathcal{U}} \tilde{\alpha}''$ in Definition 3).

Denote as $LEFT'$ some set that contains exactly one combinable simulator left half-ladder for each $\tilde{l} \in LEFT$. Note that, by Definitions 1 and 3, there must be at least one combinable simulator left half-ladder $\tilde{\tilde{l}}$ for each $\tilde{l}$, but that there also may be more than one, so the set $LEFT'$, while certainly not empty, need not be unique. By the definition of $LEFT'$, it is easy to see that $|LEFT'| = \binom{h}{\tau}$. We know that each combinable simulator left half-ladder $\tilde{\tilde{l}}$ has exactly τ rungs, and furthermore, since glue strengths in the 2HAM cannot be fractional, it is the case that τ' of these rungs bind to (the corresponding rungs of) a mate with a combined total strength of at least τ'. (Note that some, but not all, of these τ' rungs may be redundant in the sense that they do not interact with positive strength.)

There are $\binom{h}{\tau'}$ ways to position/choose τ' rungs on a (simulator) half-ladder of height h. (Note that a rung on a simulator half-ladder need not be a $m \times m$ block of tiles but merely a collection of rung-like blocks that map to rungs in the input system $\mathcal{T}$ via $\tilde{R}$.) Now consider the size $\binom{h}{\tau'}$ set of all possible rung positions, each denoted by a subset $X \subset \{0, 1, \ldots, h-1\}$, and the size $\binom{h}{\tau}$ set $LEFT'$. For each simulated half-ladder $\tilde{\tilde{l}} \in LEFT'$, there must exist a set of τ' rungs X such that $\tilde{\tilde{l}}$ binds to a mate via the rungs specified by X, with total strength at least τ'. As there are $\binom{h}{\tau}$ elements of $LEFT'$ and only $\binom{h}{\tau'}$ choices for X, the Generalized Pigeonhole Principle implies that there must be some set $LEFT'' \subset LEFT'$ with $|LEFT''| \geq \binom{h}{\tau}/\binom{h}{\tau'}$ such that every simulator left half-ladder in $LEFT''$ binds to a mate via the τ' rungs specified by a single choice of X, with total strength at least τ'. In the case that $h \geq 2\tau$, we have that $|LEFT''| \geq \binom{h}{\tau}/\binom{h}{\tau'} \geq \binom{h}{\tau}/\binom{h}{\tau-1} = \frac{h-\tau+1}{\tau}$.

Let $k = |U|^{4m^2}$, which is the number of ways to tile a neighborhood of four $m \times m$ squares from a set of $|U|$ distinct tile types. If $h = \tau\left(k^{\tau-1} + \tau\right)$, then $|LEFT''| \geq k^{\tau-1} + 1$. There are $k^{\tau'} \leq k^{\tau-1}$ ways to tile τ' neighborhoods that map to tiles of type A0 (plus any additional simulator fuzz that connects to simulated A0 tiles), under $\tilde{R}$, at the ends of the τ' rungs of a simulator left half-ladder. This tells us that there are at least two (combinable) simulator left

half-ladders $\tilde{\tilde{l}}_1, \tilde{\tilde{l}}_2 \in LEFT''$ such that $\tilde{\tilde{l}}_1$ binds to a mate via the rungs specified by X, with total strength at least τ', $\tilde{\tilde{l}}_2$ binds to a mate via the rungs specified by X, with total strength at least τ' and the rungs (along with any surrounding fuzz) specified by X of $\tilde{\tilde{l}}_1$ are tiled exactly the same as the rungs specified by X of $\tilde{\tilde{l}}_2$ are tiled. Thus, we can conclude that $\tilde{\tilde{r}}$, a mate of $\tilde{\tilde{l}}_1$, is a mate of $\tilde{\tilde{l}}_2$. We can conclude this because, while $\tilde{\tilde{l}}_1$ and $\tilde{\tilde{l}}_2$ agree exactly along τ' of their rungs, they also each have one rung in a unique position and since consecutive rungs in $\mathcal{T}$ have at least two empty spaces between then, the offset simulator rungs (and even their fuzz) cannot prevent $\tilde{\tilde{l}}_2$ from matching up with the mate of $\tilde{\tilde{l}}_1$.

However, $\tilde{R}\left(\tilde{\tilde{r}}\right) = \tilde{r} \in R$, $\tilde{R}\left(\tilde{\tilde{l}}_2\right) = \tilde{l}_2 \in L$ but $C^{\tau}_{\tilde{r},\tilde{l}_2} = \emptyset$ because $\tilde{r}$ and $\tilde{l}_2$ differ from each other in one rung location and therefore interact in $\mathcal{T}$ with total strength at most $\tau - 1$. This is a contradiction to Definition 2, which implies $C^{\tau}_{\tilde{r},\tilde{l}_2} \neq \emptyset$. □

Corollary 1. *There is no universal tile set U for the 2HAM, i.e., there is no U such that, for all 2HAM tile assembly systems $\mathcal{T} = (T, S, \tau)$, there exists an initial configuration $S_{\mathcal{T}}$ and temperature τ' such that $\mathcal{U} = (U, S_{\mathcal{T}}, \tau')$ simulates $\mathcal{T}$.*

Proof. Our proof is by contradiction, so assume that U is a universal tile set. Denote as g the strength of the strongest glue on any tile type in U. Let $\mathcal{T}' = (T', 4g + 1)$ be a modified version of the TAS $\mathcal{T} = (T, \tau)$ from the proof of Theorem 2 with each τ-strength glue in T converted to a strength $4g + 1$ glue in T' (all other glues and labels are unmodified). For any initial configuration $S_{\mathcal{T}}$, we know that $\mathcal{U} = (U, S_{\mathcal{T}}, \tau')$ does not simulate $\mathcal{T}$ for any $\tau' < 4g + 1$. If $\tau' \geq 4g + 1$, then the size of the largest supertile in $\mathcal{A}[\mathcal{U}]$ is 1, whence U is not a universal tile set. □

4 The Temperature-τ 2HAM Is Intrinsically Universal

In this section we state our second main result, which states that for fixed temperature $\tau \geq 2$ the class of 2HAM systems at temperature τ is intrinsically universal. In other words, for such τ there is a tile set that, when appropriately initialized, simulates any temperature τ 2HAM system. Denote as 2HAM(k) the set of all 2HAM systems at temperature k.

Theorem 3. *For all $\tau \geq 2$, 2HAM(τ) is intrinsically universal.*

In the full version of this paper we prove this theorem for two different, but seemingly natural notions of simulation. The first, simply called *simulation*, is where we require that when a simulated supertile $\tilde{\alpha}$ may grow, via one attachment, into a second supertile $\tilde{\beta}$, then any simulator supertile that maps to $\tilde{\alpha}$ must also grow into a simulator supertile that maps to $\tilde{\beta}$. The converse should also be true. The second notion, called *strong simulation*, is a stricter definition where in addition to requiring that all supertiles mapping to $\tilde{\alpha}$ must be capable of growing into a supertile mapping to $\tilde{\beta}$ when $\tilde{\alpha}$ can grow into $\tilde{\beta}$ in the simulated

system, we further require that this growth can take place by the attachment of *any* supertile mapping to $\tilde{\gamma}$, where $\tilde{\gamma}$ is the supertile that attaches to $\tilde{\alpha}$ to get $\tilde{\beta}$. Theorem 3 is proven for both notions of simulation. For each notion we provide three results, and in all cases we provide lower scale factor for simulation relative to strong simulation.

When we combine our negative and positive results, we get a separation between classes of 2HAM tile systems based on their temperatures.

Theorem 4. *There exists an infinite number of infinite hierarchies of 2HAM systems with strictly-increasing power (and temperature) that can simulate downward within their own hierarchy.*

Proof. Our first main result (Theorem 2) tells us that the temperature τ 2HAM cannot be simulated by any temperature $\tau' < \tau$ 2HAM. Hence we have, for all $i > 0, c \geq 4$, 2HAM $\left(c^i\right) \succ$ 2HAM $\left(c^{i-1}\right)$, where $\succ$ is the relation "cannot be simulated by". Moreover, Theorem 3 tells us that temperature τ 2HAM is intrinsically universal for fixed temperature τ. Suppose that $\tau' < \tau$ such that $\tau/\tau' \in \mathbb{N}$. Then temperature τ 2HAM can simulate temperature τ' (by simulating strength $g \leq \tau'$ attachments in the temperature τ' system with strength $g\,(\tau/\tau')$ attachments in the temperature τ system). Thus, for all $0 < i' \leq i$, 2HAM $\left(c^i\right)$ can simulate, via Theorem 3, 2HAM $\left(c^{i'}\right)$. The theorem follows by noting that our choice of c was arbitrary. □

We have shown that for each $\tau \geq 2$ there exists a single set of tile types U_τ, and a set of input supertiles over U_τ, such that the 2HAM system strongly simulates any 2HAM TAS $\mathcal{T}$. A related question is: does there exist a tile set that can simulate, or strongly simulate, all temperature τ 2HAM TASs simultaneously? Surprisingly, the answer is yes!

Theorem 5. *For each $\tau > 1$, there exists a 2HAM system $\mathcal{S} = (U_\tau, \tau)$ which simultaneously strongly simulates all 2HAM systems $\mathcal{T} = (T, \tau)$.*

References

1. Adleman, L.M., Cheng, Q., Goel, A., Huang, M.-D.A., Kempe, D., de Espanés, P.M., Rothemund, P.W.K.: Combinatorial optimization problems in self-assembly. In: Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, pp. 23–32 (2002)
2. Arrighi, P., Schabanel, N., Theyssier, G.: Intrinsic simulations between stochastic cellular automata. arXiv preprint arXiv:1208.2763 (2012)
3. Cannon, S., Demaine, E.D., Demaine, M.L., Eisenstat, S., Patitz, M.J., Schweller, R., Summers, S.M., Winslow, A.: Two hands are better than one (up to constant factors). In: Proceedings of the Thirtieth International Symposium on Theoretical Aspects of Computer Science (to appear, 2013)
4. Chacc, E.G., Meunier, P.-E., Rapaport, I., Theyssier, G.: Communication complexity and intrinsic universality in cellular automata. Theor. Comput. Sci. 412(1-2), 2–21 (2011)

5. Chen, H.-L., Doty, D.: Parallelism and time in hierarchical self-assembly. In: SODA 2012: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1163–1182. SIAM (2012)
6. Cheng, Q., Aggarwal, G., Goldwasser, M.H., Kao, M.-Y., Schweller, R.T., de Espanés, P.M.: Complexities for generalized models of self-assembly. SIAM Journal on Computing 34, 1493–1515 (2005)
7. Delorme, M., Mazoyer, J., Ollinger, N., Theyssier, G.: Bulking I: an abstract theory of bulking. Theoretical Computer Science 412(30), 3866–3880 (2011)
8. Delorme, M., Mazoyer, J., Ollinger, N., Theyssier, G.: Bulking II: Classifications of cellular automata. Theor. Comput. Sci. 412(30), 3881–3905 (2011)
9. Demaine, E.D., Demaine, M.L., Fekete, S.P., Ishaque, M., Rafalin, E., Schweller, R.T., Souvaine, D.L.: Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. Natural Computing 7(3), 347–370 (2008)
10. Doty, D., Lutz, J.H., Patitz, M.J., Schweller, R.T., Summers, S.M., Woods, D.: The tile assembly model is intrinsically universal. In: Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, pp. 439–446 (October 2012)
11. Doty, D., Lutz, J.H., Patitz, M.J., Summers, S.M., Woods, D.: Intrinsic universality in self-assembly. In: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, pp. 275–286 (2009)
12. Durand, B., Róka, Z.: The game of life: universality revisited. In: Delorme, M., Mazoyer, J. (eds.) Cellular Automata. Kluwer (1999)
13. Fujibayashi, K., Hariadi, R., Park, S.H., Winfree, E., Murata, S.: Toward reliable algorithmic self-assembly of DNA tiles: A fixed-width cellular automaton pattern. Nano Letters 8(7), 1791–1797 (2007)
14. Lafitte, G., Weiss, M.: Universal tilings. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 367–380. Springer, Heidelberg (2007)
15. Lafitte, G., Weiss, M.: Simulations between tilings. In: Conference on Computability in Europe (CiE 2008), Local Proceedings, pp. 264–273 (2008)
16. Lafitte, G., Weiss, M.: An almost totally universal tile set. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 271–280. Springer, Heidelberg (2009)
17. Luhrs, C.: Polyomino-safe DNA self-assembly via block replacement. In: Goel, A., Simmel, F.C., Sosík, P. (eds.) DNA 14. LNCS, vol. 5347, pp. 112–126. Springer, Heidelberg (2009)
18. Lund, K., Manzo, A.T., Dabby, N., Micholotti, N., Johnson-Buck, A., Nangreave, J., Taylor, S., Pei, R., Stojanovic, M.N., Walter, N.G., Winfree, E., Yan, H.: Molecular robots guided by prescriptive landscapes. Nature 465, 206–210 (2010)
19. Ollinger, N.: Intrinsically universal cellular automata. In: The Complexity of Simple Programs, in Electronic Proceedings in Theoretical Computer Science, vol. 1, pp. 199–204 (2008)
20. Ollinger, N., Richard, G.: Four states are enough? Theoretical Computer Science 412(1), 22–32 (2011)
21. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science 332(6034), 1196 (2011)
22. Qian, L., Winfree, E., Bruck, J.: Neural network computation with DNA strand displacement cascades. Nature 475(7356), 368–372 (2011)
23. Rothemund, P.: Folding DNA to create nanoscale shapes and patterns. Nature 440(7082), 297–302 (2006)
24. Rothemund, P.W., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology 2(12), 2041–2053 (2004)

25. Seeman, N.C.: Nucleic-acid junctions and lattices. Journal of Theoretical Biology 99, 237–247 (1982)
26. Wang, H.: Proving theorems by pattern recognition – II. The Bell System Technical Journal XL(1), 1–41 (1961)
27. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Institute of Technology (June 1998)
28. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature 394(6693), 539–544 (1998)
29. Yurke, B., Turberfield, A., Mills Jr., A., Simmel, F., Neumann, J.: A DNA-fuelled molecular machine made of DNA. Nature 406(6796), 605–608 (2000)

Clustering in the Boolean Hypercube in a List Decoding Regime

Irit Dinur[1,2,⋆] and Elazar Goldenberg[1]

[1] Weizmann Institute of Science
[2] Radcliffe Institute for Advanced Study

Abstract. We consider the following *clustering with outliers* problem: Given a set of points $X \subset \{-1,1\}^n$, such that there is some point $z \in \{-1,1\}^n$ for which $\Pr_{x \in X}[\langle x,z\rangle \geq \varepsilon] \geq \delta$, find z. We call such a point z a (δ,ε)-center of X.

In this work we give lower and upper bounds for the task of finding a (δ,ε)-center. We first show that for $\delta = 1-\nu$ close to 1, i.e. in the "unique decoding regime", given a $(1-\nu,\varepsilon)$-centered set our algorithm can find a $(1-(1+o(1))\nu,(1-o(1))\varepsilon)$-center. More interestingly, we study the "list decoding regime", i.e. when δ is close to 0. Our main upper bound shows that for values of ε and δ that are larger than $1/\mathrm{poly}\log(n)$, there exists a polynomial time algorithm that finds a $(\delta-o(1),\varepsilon-o(1))$-center. Moreover, our algorithm outputs a list of centers explaining all of the clusters in the input.

Our main lower bound shows that given a set for which there exists a (δ,ε)-center, it is hard to find even a $(\delta/n^c,\varepsilon)$-center for some constant c and $\varepsilon = 1/\mathrm{poly}(n), \delta = 1/\mathrm{poly}(n)$.

Keywords: Clustering, list decoding, approximation algorithms.

1 Introduction

Suppose we are given access to a set of points $X \subset \{-1,1\}^n$ such that at least δ fraction of these points are ε-correlated with some unknown "center" $z \in \{-1,1\}^n$. We wish to recover (an approximation of) z even if the remaining $1-\delta$ fraction of the points in X are arranged in an adversarial manner. Formally, a (δ,ε)-center is defined as follows,

Definition 1. *Given a set $X \subset \{-1,1\}^n$, the point $z \in \{-1,1\}^n$ is called a (δ,ε)-center if there exists $X' \subset X$, $|X| \geq \delta X$, such that:*

$$\forall x \in X' \, \langle x,z\rangle \geq \varepsilon.$$

We denote by $C_\varepsilon(z)$ the set of all points $x \in X$ satisfying $\langle z,x\rangle \geq \varepsilon$.

We call $C_\varepsilon(z)$ the *cluster* of z in X.

⋆ Research supported in part by the Israel Science Foundation grant no. 1179/09 and by the Binational Science Foundation grant no. 2008293 and by an ERC grant no. 239985. Part of the research done while visiting Microsoft Research in New England.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 413–424, 2013.

Clustering is a vastly studied topic, but usually the the focus is on inputs that are drawn from some unknown (parameterized) distribution, or on deterministic data with a *small* amount of adversarial noise. Here we consider the problem in the "list decoding" regime, where the fraction of corrupted data points approaches 1. In this case, there are potentially more than a single cluster, so the algorithm needs to output a *list* of clusters.

Formally, we study the following (δ, ε)-clustering problem: Given a set $X \subset \{-1,1\}^n$ that contains a (δ, ε)-center, find all such centers.

Ideally, we are seeking an algorithm that lists all possible centers. Of course, list decoding is feasible when there is some way to bound the list size. In the case of error correcting codes, the distance of the code may facilitate such a bound. In our case, there is no underlying code, so we instead rely on an approximate representation. The idea is simply to output a short list of centers such that every cluster is 'represented' by some center in the list. Pinning down the best notion of 'representation' turns out to be tricky, and we view this as part of the contribution of this paper, on which we elaborate more in Section 3.2.

In this work, we study upper and lower bounds for the problem of finding (δ, ε)-center. The complexity of this problem depends on the choice of the parameters ε and δ: We show that when ε or δ are close to 1, then the task of finding a (δ, ε)-center is relatively easy. For ε and δ that are larger than $1/\mathrm{poly}\log(n)$ we present an algorithm that finds a $((1-o(1))\delta, (1-o(1))\varepsilon)$-center. We complement our results by showing hardness results when δ and ε are much smaller. We elaborate on these results next.

1.1 Upper Bounds

In this part we present several approximation results for the (δ, ε)-clustering problem. The approximation version of the (δ, ε)-center problem allows the output to be a center whose cluster has a smaller margin value $\varepsilon' \leq \varepsilon$, and that contains a smaller $\delta' \leq \delta$ fraction of the points. In other words, under the promise of existence of a (δ, ε) center, the approximation algorithm will find a (δ', ε')-center.

Definition 2 (Approximate-Cluster Problem). *An instance of the problem is a (δ, ε)-centered set of points $X = \{x_1, \ldots, x_N\} \subset \{-1,1\}^n$. The goal is to find a (δ', ε') center with parameters as close as possible to ε, δ.*

We first give an approximation algorithm for the easier "unique-decoding" parameter regime, i.e. where δ is close to 1.

Theorem 1. *Let $0 < \varepsilon, \delta < 1$, and let $X \subset \{-1,1\}^n$. There is a polynomial-time algorithm for solving the following problems,*

1. *If X is $(1, \varepsilon)$-centered find a $(1, \varepsilon - O(\frac{\log N}{\sqrt{n}}))$-center.*
2. *If X is $(1-\nu, \varepsilon)$-centered find a $(1 - 1/a, \varepsilon - a\nu(1+\varepsilon))$-center, for any $a > 1$. In particular, for a parameter $\tau > 0$ if $\nu < \tau^2\varepsilon/(1+\varepsilon)$ then this gives a $(1-\tau, (1-\tau)\varepsilon)$-center.*

This algorithm is simply based on linear programming.

We next turn to the more challenging setting that is when both ε and δ are small. Our main result is a polynomial time algorithm that approximates the (δ, ε)-center for values of ε and δ that are larger than $1/\text{poly}\log(n)$. As explained earlier we are in a "list decoding" setting, that allows for more than one cluster to exist simultaneously. Moreover, we want our algorithm to output a list that "exhaustively" explains all of the clusters in the data.

Here, the goal for the algorithm is to output an exhaustive list of centers, namely a list for which:

- Each member in the list is a cluster in the data.
- Each cluster in the data is approximately equal to one of the clusters in the list.

The reason for asking only for an approximate equality to members in the list is clear: there can be an exponential number of different clusters, and approximation seems like a natural way to get a manageable list size. However, it turns out that even when allowing approximate centers, there still might be an exponential number of them (more details in Section 3.2). We show that this can only occur if the exponentially-many clusters are contained in one bigger cluster. In light of this example, the new goal for the algorithm becomes to output a list of centers that "cover" all of the clusters in the set. We state below an informal version of our main theorem, for a formal version please see Section 3.2.

Theorem 2 (Main result, informal). *Let $\varepsilon, \delta > 0$ be parameters, and let $X \subset \{-1,1\}^n$ be (δ, ε)-centered, with $|X| = N$. There exists an algorithm that runs in time polynomial in $n, N, \exp(\frac{1}{\varepsilon^2 \log 1/\varepsilon\delta})$, and outputs a list L of points each in $\{-1,1\}^n$, such that with probability $1 - 2^{-n}$ the following holds:*

- *Each $y \in L$ is a $((1-o(1))\cdot\delta, (1-o(1))\cdot\varepsilon)$ center for X.*
- *For every $z \in \{-1,1\}^n$ which is a (δ,ε)-center and , there exists $y \in L$ such that,*

$$C_{(1-o(1))\varepsilon}(z) \subseteq_{o(1)} C_{(1-o(1))\varepsilon}(y),$$

  where for sets A, B and $0 < \tau < 1$ we define $A \subseteq_\tau B$ if $|A \setminus B| < \tau |A|$.
- *Moreover, if $z \in \{-1,1\}^n$ is a (δ,ε)-center that is approximately maximal[1], there exists $y \in L$ such that,*

$$\left|C_\varepsilon(z) \triangle C_{(1-o(1))\varepsilon}(y)\right| < o(1)\left|C_\varepsilon(z)\right|.$$

We note that, while a priory the number of covering centers could be exponentially large, the correctness of our algorithm is a proof that it is polynomially bounded.

Let us briefly sketch our proof of this theorem. We first randomly restrict the given set of points into a small poly-logarithmic subset of coordinates. We show that a (δ, ε)-center for X is still a center in the restricted space (if δ and ε are

[1] Essentially, a cluster is approximately maximal if it is not approximately contained in any larger cluster, see Definition 5.

large enough). Therefore, we can enumerate over all possible centers and find a solution in the restricted space. Then we show how to extend the solution from the local space into a global solution.

Our last algorithmic result deals with smaller values of ε, $\varepsilon \geq \log n/\sqrt{n}$. When δ and ε are this small the above algorithm runs in super polynomial time. For such a choice of parameters, we prove that there is always a *data point* $x \in X$ that is itself a $(2\delta\varepsilon, \varepsilon^2)$-center. This leads to the following algorithm:

Theorem 3. *Let $X \subset \{-1,1\}^n$ be an (δ,ε)-centered set of N elements and let $\varepsilon \gg \log n/\sqrt{n}$. There is an algorithm that runs in time* $\mathrm{poly}(N)$ *and outputs a list L such that:*

- *Each $z \in L$ is a $(2\delta\varepsilon, \varepsilon^2)$-center.*
- *For every center z, there exists a center $z' \in L$, such that at least 2ε fraction of the points $x \in X$ satisfying $\langle x, z\rangle > \varepsilon$ are also satisfying $\langle x, z'\rangle > \varepsilon^2$.*

Observe that this result is incomparable to the one attained in Theorem 2. While in Theorem 2 we are able to find almost the whole cluster of each maximal center, the algorithm proposed by Theorem 3 finds only a non-trivial subset of each cluster. On the other hand, Theorem 3 is stronger in the sense that it has a guarantee for each center, and not just for a subset of them as in Theorem 2.

1.2 Lower Bounds

We next turn to lower bounds. It is not hard to see that given a $(1, \mathrm{poly}(1/n))$-centered set it is **NP**-hard to find such a center, by reduction from, say, 3SAT. Moreover, we describe stronger reductions that show the hardness of the approximation problem. That is, we show that for some choices of the parameters $(\delta, \varepsilon, \delta', \varepsilon')$ the approximate center problem is infeasible unless $\mathcal{BPP} \supseteq \mathbf{NP}$. Formally, we consider the following gap-clustering problem:

Definition 3. *The gap-clustering problem with parameters $(\delta, \varepsilon, \delta', \varepsilon')$:*

The input of the problem is a set of point $X \subset \{-1,1\}^n$. The goal is to distinguish between the following cases:

- *There exists a (δ,ε)-center in X.*
- *There is no (δ',ε')-center in X.*

There are four parameters involved so it is complicated to understand the tradeoffs between the settings of the parameters. There are two key points to address: First we would like to get as large as possible a gap between δ and δ' and ε and ε'. The second is the location of the gap: find the largest ε and δ for which the problem is still hard.

Since a large gap between δ and δ' might lead to a small gap between ε and ε', and vice versa, we separate this optimization question into two: find largest δ-gap and find the largest ε-gap.

Our first hardness result focuses on the gap between ε and ε'. It shows that it is hard to distinguish between the case that there exists a (δ,ε)-center, and the case that there is no $(\delta/c, \varepsilon/2)$-center for some constant $c > 1$, ε which is an arbitrarily large polynomial in $1/n$, and δ that is a constant, formally:

Theorem 4. *Unless $\mathcal{BPP} \supseteq \mathbf{NP}$, there exist constants $\delta, c > 1$, such that for every constant $\alpha > 2$, it is infeasible to solve the gap-clustering problem with parameters: $\delta, \delta' = \delta/c$, $\varepsilon = \frac{2}{n^{1/\alpha}} - o(\frac{1}{n^{1/\alpha}})$ and $\varepsilon' = \frac{1}{n^{1/\alpha}} + \omega(\frac{1}{n^{1/\alpha}})$.*

Our next result focuses on amplifying the gap between δ and δ'. It shows that it is **NP**-hard to distinguish between the case that there exists a (δ, ε)-center, and the case that there is no $(\frac{\delta}{n^c}, \varepsilon')$-center, for some constant c, and for ε, δ which are poly$(1/n)$, and $\varepsilon' = (1 - o(1))\varepsilon$. Formally:

Theorem 5. *Unless $\mathcal{BPP} \supseteq \mathbf{NP}$, there exist constants $c_1 > 0, c_2 > 0$ such that it is infeasible to solve the gap-clustering problem with parameters: $\delta = n^{-c_1}, \delta' = \frac{\delta}{n^{c_2}}$ and $\varepsilon > \varepsilon' = \Theta(n^{-1/3})$.*

There is a gap between our algorithmic results and the aforementioned lower bounds, two particular open questions are:

- Theorem 3 states that given a (δ, ε)-centered set, there is a polynomial time algorithm that finds a $(\delta\varepsilon, \varepsilon^2)$-center. A natural question that arises is how hard is the task of finding a better center - that is finding a (δ', ε')-center for $\varepsilon' \gg \varepsilon^2$ and δ' being non trivial.
- Both our hardness results deal with sub-constant values of ε, and it is not clear whether we can strengthen our hardness result to deal with larger values of ε. In particular, given a $(\delta = 1/\text{poly}(n), \varepsilon = \Omega(1))$-centered set is it hard to find a $(\delta', \varepsilon/2)$-center for any nontrivial δ' ? Note that if we take δ to be larger than $1/\text{poly}(\log(n))$, then by Theorem 2 we can find an approximate solution in polynomial time.

1.3 Related Work

Upper Bounds. The most related work on clustering with outliers, as far as we know, is the work of [BHPI02]. This work considers several clustering problems, one of which is the clustering with outliers problem. The main difference is that we consider a set of data points in the Boolean hypercube $\{-1, 1\}^n$, whereas they consider a set of points in $\mathbb{R}^n$, and their algorithm outputs centers that are in $\mathbb{R}^n$ as well. We provide a more detailed comparison between our work and theirs in the full version of the paper.

We are not aware of works that looked at the "list-decoding" version of the clustering problem, where the algorithm needs to output a list explaining all of the clusters in the data. In other settings in theoretical computer science list decoding has been, of course, extremely successful. The seminal work of Goldreich and Levin [GL89] has a similar feel: a string is recovered from its noisy parities. In our setting, we get not a parity of the bits, but an ε-correlated version of them, and this only on a δ fraction of the inputs. The most similar works are those of [IJK06, IJKW10] on list decoding direct products. In these works the decoding algorithm is given an access to k-tuples of bits of the hidden string such that only a δ fraction of the k-tuples are correct and the rest are adversarial noise. Our setting is even harder in that even the δ fraction of "good"

inputs are only guaranteed to be ε-correlated with the hidden string, rather than completely equal to it on k known bits. Extending our clustering algorithm to a direct product decoding result has been one of our main motivations, and is still work in progress.

Lower Bounds. Our lower bounds are closely related to the works of [FGKP09, GR06] on the MaxLin-$\mathbb{Q}$ problem, defined as follows: Given a system of equations over the rationales, and we are expected to "almost" satisfy as many equations as possible. Formally a MaxLin-$\mathbb{Q}$ with parameters $(N, n, \delta, \varepsilon)$ consists of a system of N equations over n variables $x_1, \ldots, x_n$ with rational coefficients,

$$\{a_{i0} + \sum_{j=1}^{N} a_{ij} x_j = 0\}_{j=1,\ldots,N}$$

and the goal is to distinguish between the following cases:

- At least $(1-\delta)N$ of the equations can be satisfied.
- In any assignment:

$$\left| a_{i0} + \sum_{j=1}^{N} a_{ij} x_j \right| < \varepsilon$$

  is true for at most δN equations.

In [FGKP09] this problem is shown to be **NP**-hard for any constant value of $\delta > 0$.

The gap-clustering problem and MaxLin-$\mathbb{Q}$ are similar in the following sense: In the completeness case, there exits an assignment (center) that satisfies (correlates) much more equations (points) compared to the soundness case. Furthermore, the quality of the solution considered in the completeness is much better compared to the soundness case. However, there are several hurdles that prevent us from reducing MaxLin-$\mathbb{Q}$ into gap-clustering. First, the coefficients of the linear-equations can take values outside $\{-1, 1\}$ unlike in gap-clustering. Second, in MaxLin-$\mathbb{Q}$ we are trying to satisfy equalities, and not inequalities as in gap-clustering. Third, note that it is hard to solve MaxLin-$\mathbb{Q}$ even when there is an assignment that satisfies $1 - \varepsilon$ fraction of equalities. In comparison, the problem of finding a $(1-\delta, \varepsilon)$-center is easy, see Theorem 1.

Although we could not directly reduce MaxLin-$\mathbb{Q}$ into gap-clustering, we were able to apply similar ideas to those presented in [FGKP09] and [GR06] to derive our hardness results.

Organization of the Paper: Section 2 contains standard tools we use later. Section 3 studies the upper bounds for our clustering problem and contains the proof of Theorem Theorem 2. In Section 4 we study lower bounds for the gap-clustering problem.

2 Preliminaries

We state the Johnson bound as appears in the book [AB08]. It asserts that, for an error correcting code with distance $1/2 - \varepsilon^2$, and for every word x, a ball of radius $1/2 - \varepsilon$ around x cannot contain too many codewords.

Lemma 1 (Johnson Bound [Joh62], Theorem 19.23 in [AB08]). *Let $0 < \varepsilon < 1$, for every $x \in \{0,1\}^n$, there exist at most $1/(2\varepsilon)$ vectors $y_1, \ldots, y_\ell \in \{0,1\}^n$ such that $\Delta(x, y_i) \leq 1/2 - \varepsilon$ for every $i \in [\ell]$, and $\Delta(y_i, y_{i'}) \geq 1/2 - \varepsilon^2$ for every $i \neq i' \in [\ell]$.*

We also state here the standard Chernoff bound:

Lemma 2 (Chernoff Bound). *Let $X_1, \ldots, X_t$ be random independent variables taking values in the interval $[0,1]$, with expectations $\mu_1, \ldots, \mu_t$, respectively. Let $X = \frac{1}{t}\sum_{i\in[t]} X_i$, and let $\mu = \frac{1}{t}\sum_{i\in[t]} \mu_i$ be the expectation of X. For any $0 < \gamma \leq 1$, we have the following:*

$$Pr[|X - \mu| \geq \gamma] \leq \exp^{-\gamma^2 n/3}.$$

Notation. For two sets $A, B \subseteq \{-1,1\}^n$ we denote their symmetric difference by $A \triangle B$. For a vector $z \in \{-1,1\}^n$ and a subset $K \subseteq [n]$, we denote by z_K its restriction to the coordinates in K.

3 Upper Bounds: Algorithms for Clustering

In this section we describe algorithms for clustering first in the "unique decoding" regime, where a small fraction of the data points are corrupted, and then in the "list decoding" regime, where a very large fraction of the data is corrupted.

We first observe that if X has a (δ, ε)-center, for $\varepsilon = 1 - \tau$ for small τ, then finding an approximate center is relatively easy: Any point $x \in X$ that belongs to the centered cluster is itself a $(\delta, 1 - 2\tau)$-center for that cluster, by the triangle inequality. By enumerating over all elements in X and checking for each $x \in X$ how many $y \in X$ are within the specified radius, we can recover a $(\delta, 1 - 2\tau)$-center.

The more interesting case is, therefore, when $\varepsilon < 1/2$ approaches 0.

3.1 Clustering with Few Outliers (Proof of Theorem 1)

We begin by addressing the easier "unique decoding" regime, where only a relatively small fraction of the data points are corrupt. More accurately, we give an algorithm that addresses the situation where δ is close to 1.

Theorem 1. *Let $0 < \varepsilon, \delta < 1$, and let $X \subset \{-1,1\}^n$. There is a polynomial-time algorithm for solving the following problems,*

1. *If X is $(1,\varepsilon)$-centered find a $(1,\varepsilon - O(\frac{\log N}{\sqrt{n}}))$-center.*
2. *If X is $(1-\nu,\varepsilon)$-centered find a $(1-1/a,\varepsilon - a\nu(1+\varepsilon))$-center, for any $a>1$. In particular, for a parameter $\tau>0$ if $\nu<\tau^2\varepsilon/(1+\varepsilon)$ then this gives a $(1-\tau,(1-\tau)\varepsilon)$-center.*

Proof. 1. Given a $(1,\varepsilon)$-centered set $X\subseteq\{-1,1\}^n$, write a linear program in variables $z_1,\ldots,z_n\in\mathbb{R}$ with the following equations

$$\forall x\in X,\quad \sum_i x_i z_i \geq \varepsilon n; \qquad \forall i\in[n],\quad -1\leq z_i\leq 1$$

The solution will be some $z\in[-1,1]^n$, and output $\tilde{z}$ the randomized rounding of z, i.e. $\tilde{z}_i=1$ with probability $(1+z_i)/2$.
A standard Chernoff bound will show that $|\langle\tilde{z},x\rangle-\langle z,x\rangle|<\sqrt{2\log|X|/n}$ for all $x\in X$ with high probability.

2. Given a $(1-\nu,\varepsilon)$-centered set X we write a similar linear program, except we add 'violation' variables v_x per each x as follows

$$\forall i\in[n]\quad -1\leq z_i\leq 1$$

$$\forall x\in X\quad 0\leq v_x\leq 1+\varepsilon$$

$$\forall x\in X\quad \frac{1}{n}\sum_i x_i z_i + v_x \geq \varepsilon$$

and then we find a solution minimizing $val=\frac{1}{|X|}\sum_x v_x$. Again, the final output of the algorithm is a randomized rounding $\tilde{z}$ of the solution z.
It is easy to see that the solution $z=\bar{0}$ with $\forall x,\ v_x=\varepsilon$ is a feasible solution whose value is $val=\varepsilon$. A more interesting solution is where z is the promised $(1-\nu,\varepsilon)$-center, and for every equation violated by x outside this ball, we set $v_x=1+\varepsilon$. This solution has value $val=0\cdot(1-\nu)+(1+\varepsilon)\cdot\nu=(1+\varepsilon)\nu$. These two solutions show that the solution to the linear program gives information only as long as $(1+\varepsilon)\nu<\varepsilon$. Suppose $z,\{v_x\}_{x\in X}$ is the solution for this system, with value $v=\mathbb{E}_x[v_x]\leq(1+\varepsilon)\nu<\varepsilon$. By Markov's inequality, at most $1/a$ fraction of the x's have $v_x>av$. The remaining $1-1/a$ equations are satisfied to within av, as claimed. The last conclusion follows by setting $a=1/\tau$.

3.2 Clustering with Few Outliers: The List Decoding Regime

We now turn to the clustering question when the input consists of mostly noise. In other words, where the data set X is guaranteed to be (δ,ε)-centered, for values of δ,ε as small as $1/\mathrm{poly}\log(n)$.

Clearly, X might have several distinct (δ,ε)-centers, and ideally we would like an algorithm that outputs a list of all of them. To control the length of the list, we must settle for a list of centers that 'represent' all the possible centers in X. One natural way to define 'represent' is by saying that z represents z' if the

symmetric difference between $C_\varepsilon(z)$ and $C_\varepsilon(z')$ is small (compared to their size). However, this notion turns out to be insufficient. It is easy to describe a set X of points that are highly correlated to a single center, and yet could be explained by an exponential number of other centers, whose pairwise symmetric difference is large, see full version for details. This example shows that the best we can hope for is an algorithm that outputs a list of clusters that "approximately cover" every cluster, in the sense that every (δ, ε) cluster is guaranteed to be approximately contained in a cluster from the list.

Definition 4. *We say that $A \subseteq_\tau B$ if $|A \setminus B| \leq \tau |A|$.*

Definition 5 (τ-maximal center). *For a set X, and parameters $\delta, \varepsilon, \tau > 0$ we say that a (δ, ε)-center z is τ-maximal if the following holds: For every $z' \in \{-1,1\}^n$, if $C_\varepsilon(z) \subseteq_{2\tau} C_{(1-\tau)\varepsilon}(z')$, then*

$$\left|C_{(1-\tau)\varepsilon}(z')\right| < (1+\rho)\left|C_\varepsilon(z)\right|, \text{ for } \rho = \tau^2\delta\varepsilon/8.$$

The main point of this definition is that if there is some z' whose cluster approximately contains the cluster of a maximal z, then the cluster of z' is not much larger. With this definition, we can now state the formal version of Theorem 2:

Theorem 6 (Formal Version of Theorem 2). *Let $\varepsilon, \delta, \tau > 0$ be parameters. Let $X \subset \{-1,1\}^n$ be an N-element set that is (δ, ε)-centered. There exists an algorithm that runs in time polynomial in $n, N, 2^{O(\frac{1}{\varepsilon^2\tau^2}\log 1/\tau^2\delta\varepsilon)}$, and outputs a list L of points each in $\{-1,1\}^n$, such that with probability $1-2^{-n}$ the following holds:*

1. *Each $y \in L$ is a $((1-\tau)\cdot\delta, (1-2\tau)\cdot\varepsilon)$ center for X.*
2. *For each $z \in \{-1,1\}^n$ which is a (δ,ε)-center there exists $y \in L$ such that,*

$$C_{(1-\tau)\varepsilon}(z) \subseteq_{2\tau} C_{(1-2\tau)\varepsilon}(y).$$

3. *Moreover, if z is τ-maximal, then*

$$\left|C_\varepsilon(z) \triangle C_{(1-\tau)\varepsilon}(y)\right| < O(\tau)\left|C_{(1-\tau)\varepsilon}(z)\right|.$$

The following lemma is the main technical tool in the proof:

Lemma 3. *Let $\varepsilon, \delta, \tau > 0$ be parameters. Let $X \subset \{-1,1\}^n$ be an N-element set that is (δ,ε)-centered. There exists an algorithm that runs in time* $\mathrm{poly}(n, N, 2^k)$, *where $k = O(\frac{1}{\varepsilon^2\tau^2}\log 1/\tau^2\delta\varepsilon)$, and outputs a list L of at most 2^k points each in $\{-1,1\}^n$, such that:*

- *Each $y \in L$ is a $((1-\tau)\cdot\delta, (1-2\tau)\cdot\varepsilon)$ center for X.*
- *For each $z \in \{-1,1\}^n$ which is a (δ,ε)-center with probability $> 1/2$ there exists $y \in L$ such that,*

$$C_{(1-\tau)\varepsilon}(z) \subseteq_{2\tau} C_{(1-2\tau)\varepsilon}(y).$$

Due to space limitations we omit the full proof of Theorem 6 and give below only the proof of the main technical lemma above, Lemma 3.

Proof (Proof of Lemma 3). The proof of this lemma follows by randomly restricting the points to a smaller dimensional space and then enumerating to find a good approximation for the cluster. The approximate center is then found by applying Theorem 1 on the approximate cluster. This algorithm is relatively efficient when $\varepsilon, \delta = \text{poly}(1/\log n)$. The algorithm is as follows.

Algorithm 1. Randomly Restrict and Enumerate

Input: A (δ, ε)-centered set X.
Parameters: $k = \frac{C}{\varepsilon^2 \tau^2} \log 1/\tau^2 \delta\varepsilon$ for some large enough C to be determined later, and $\tau > 0$.

1. Choose at random a multiset $K \subseteq [n]$ by selecting a random $i \in [n]$ into K repeatedly k times with replacement.
2. For each $y \in \{-1, 1\}^k$ let $X(y) = \{x \in X \mid \langle x_K, y\rangle \geq (1 - \tau/2) \cdot \varepsilon\}$, and compute the center $z(y)$ of $X(y)$ using the linear programming algorithm from Item 3 of Theorem 1 with correlation parameter $(1-\tau)\varepsilon$. If $z(y)$ is a $((1-\tau)\cdot\delta, (1-2\tau)\cdot\varepsilon)$ center for X then output it.

Clearly, each center produced by the list is a $((1-\tau)\cdot\delta, (1-2\tau)\cdot\varepsilon)$ center for X. Moreover, the list size is bounded by 2^k. It is left to prove the second and the third item of the lemma. Let z^* be a (δ, ε)-center, and consider the set

$$X^* := \{x \in X \mid \langle x_K, z^*_K\rangle > \varepsilon(1 - \tau/2)\}.$$

We will prove that $1-\gamma$ fraction of the elements of X^* also belong to $C_{(1-\tau)\varepsilon}(z^*)$, which means that X^* is $(1-\gamma, (1-\tau)\varepsilon)$-centered. This implies that at step 2 our algorithm will output some center z' of X^* (because $X^* = X(y)$ for $y = z^*_K$). We will then prove that $C_{1-\tau}\varepsilon(z^*) \subset_{2\tau} C_{(1-2\tau)\varepsilon}(z')$ which means that z^* is covered by our list.

We first claim that the sampling is good enough.

Claim. We say that $x \in X$ is typical with respect to K if $|\langle x, z^*\rangle - \langle x_K, z^*_K\rangle| \leq \frac{\tau\varepsilon}{2}$. Then for at least half of the choices of K, the fraction of typical x's is at least $1 - \gamma$ for $\gamma := 2\exp(-\tau^2\varepsilon^2 k/12) < \tau^2\delta\varepsilon/8$.

Proof. We first show that $\Pr_{x\in X,K}[x \text{ is not typical}] \leq \gamma$. In fact, we show this for each fixed x separately. For a random i, $x_i z^*_i$ can be viewed as a random ± 1 variable whose expectation is $\langle x, z^*\rangle$. By a Chernoff bound the probability that $|\langle x, z^*\rangle - \langle x_K, z^*_K\rangle| > \tau\varepsilon/2$ is at most $\gamma/2$.

By an averaging argument this means that for at least half of the choices of K have no more than γ atypical x's, which gives the claim.

Suppose from now on that K is as in the claim, so there are at most γ atypical points $x \in X$. Clearly, every point in X^* for which $\langle x, z^* \rangle < (1-\tau)\varepsilon$ must be atypical,and every typical point in $C_\varepsilon(z)$ is also in X^*. So z^* is a $(1-\gamma/(\delta-\gamma), (1-\tau)\varepsilon)$-center for this set. These conditions allow step 2 of the algorithm to work, according to Theorem 1, and to output a point z' that is a $(1-\tau, (1-2\tau)\varepsilon)$-center for X^*. The conditions of Theorem 1 are simply that $\gamma/(\delta-\gamma) < 2\gamma/\delta < \tau^2 \frac{\varepsilon}{1-\varepsilon}$ which clearly holds by the choice of k.

In order to complete the proof, we show

$$\left|C_{(1-\tau)\varepsilon}(z^*) \setminus C_{(1-2\tau)\varepsilon}(z')\right| \leq \gamma |X| + \tau |X^*| \leq 3\tau \left|C_{(1-\tau)\varepsilon}(z^*)\right|$$

This is simply since the measure of points that are in $C_{(1-\tau)\varepsilon}(z^*)$ but not in X^* is at most $\gamma |X|$, and the measure of points in X^* but not in $C_{(1-2\tau)\varepsilon}(z')$ is at most $\tau |X^*|$ (by the guarantee of Theorem 1). For the last inequality we rely on the fact that $\left|C_{(1-\tau)\varepsilon}(z^*)\right| \geq \delta$ and $\gamma/\delta < \tau$.

Due to space limitations we omit the proof of Theorem 3, and it appears in the full version of this paper.

4 Hardness of Approximating the Gap-Clustering Problem

In this section we study the hardness of the task of finding a (δ, ε) over various choices of parameters. We first show that it is infeasible to solve the task of finding a (δ, ε)-center without approximation, even when $\delta = 1$.

Claim. Unless $\mathcal{BPP} \supseteq \mathbf{NP}$, given a $(1, \varepsilon)$-centered set it is infeasible to find a $(1, \varepsilon)$-center, for $\varepsilon = n^{-1/3}(1 - o(1))$.

Due to space limitation we omit the proof, and it can be foul in the full version of the paper.

Of course, the more interesting question is that of approximate hardness. We show that one cannot even find an approximate center when such exists. Recall the definition the problem (Definition 3), where the task is given a set of points X distinguish between the case that there exists a (δ, ε)-center, and the case where no (δ', ε')-center exists.

There are two key points which we like to address in the parameters settings: First we would like to get as large as possible a gap between δ and δ' and ε and ε'. The second is locating the gap- finding the largest ε and δ for which the problem is hard. It is easy to see the larger ε and δ are, the stronger is the hardness result.

There are four parameters involved so the gaps and tradeoffs between the parameters can become quite complicated. We separate the task of finding the largest δ-gap and the task of finding the largest ε-gap.

Our first hardness result shows a factor 2 gap between ε and ε'. It shows that it is hard to distinguish between the case that there exists a (δ, ε)-center, and the

case that there is no $(\delta/c, \varepsilon/2)$ for some constant $c > 1$, ε which is an arbitrarily large polynomial in $1/n$, and δ that is a constant:

Theorem 4. *Unless $\mathcal{BPP} \supseteq \mathbf{NP}$, there exist constants $\delta, c > 1$, such that for every constant $\alpha > 2$, it is infeasible to solve the gap-clustering problem with parameters: $\delta, \delta' = \delta/c$, $\varepsilon = \frac{2}{n^{1/\alpha}} - o(\frac{1}{n^{1/\alpha}})$ and $\varepsilon' = \frac{1}{n^{1/\alpha}} + \omega(\frac{1}{n^{1/\alpha}})$.*
Due to space limitations we omit the proof, and include it in the full version of the paper.

Our next result shows a polynomial gap between δ and δ'. It shows that it is hard to distinguish between the case that there exists a (δ, ε)-center, and the case that no $(\frac{\delta}{n^c}, \varepsilon')$-center exists, for c being some constant, ε, δ which are poly$(1/n)$, and $\varepsilon' = (1 - o(1))\varepsilon$:

Theorem 5. *Unless $\mathcal{BPP} \supseteq \mathbf{NP}$, there exist constants $c_1 > 0, c_2 > 0$ such that it is infeasible to solve the gap-clustering problem with parameters: $\delta = n^{-c_1}, \delta' = \frac{\delta}{n^{c_2}}$ and $\varepsilon > \varepsilon' = \Theta(n^{-1/3})$.*

Due to space limitations we omit the proof, and include it in the full version of the paper.

Acknowledgement. We would like to thank Sariel Har-Peled, Guy Kindler and Igor Shinkar for helpful discussions.

References

[AB08] Arora, S., Barak, B.: Complexity theory: A modern approach (2008)
[BHPI02] Badoiu, M., Har-Peled, S., Indyk, P.: Approximate clustering via core-sets. In: Proc. 34th Annu. ACM Sympos. Theory Comput., pp. 250–257 (2002)
[FGKP09] Feldman, V., Gopalan, P., Khot, S., Ponnuswami, A.K.: On agnostic learning of parities, monomials, and halfspaces. SIAM J. Comput. 39(2), 606–645 (2009)
[GL89] Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, STOC 1989, pp. 25–32. ACM, New York (1989)
[GR06] Guruswami, V., Raghavendra, P.: Hardness of learning halfspaces with noise. In: Proceedings of FOCS, pp. 543–552 (2006)
[IJK06] Impagliazzo, R., Jaiswal, R., Kabanets, V.: Approximately list-decoding direct product codes and uniform hardness amplification. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, pp. 187–196. IEEE Computer Society, Washington, DC (2006)
[IJKW10] Impagliazzo, R., Jaiswal, R., Kabanets, V., Wigderson, A.: Uniform direct product theorems: Simplified, optimized, and derandomized, vol. 39, pp. 1637–1665 (2010)
[Joh62] Johnson, S.M.: A new upper bound for error-correcting codes. IRE Transactions on Information Theory 8(2), 203–207 (1962)
[TSSW96] Trevisan, L., Sorkin, G.B., Sudan, M., Williamson, D.P.: Gadgets approximation, and linear programming. In: IEEE Annual Symposium on Foundations of Computer Science, p. 617 (1996)

A Combinatorial Polynomial Algorithm for the Linear Arrow-Debreu Market*

Ran Duan and Kurt Mehlhorn

Max-Planck-Institut für Informatik, Saarbrücken, Germany
{duanran,mehlhorn}@mpi-inf.mpg.de

Abstract. We present the first combinatorial polynomial time algorithm for computing the equilibrium of the Arrow-Debreu market model with linear utilities. Our algorithm views the allocation of money as flows and iteratively improves the balanced flow as in [Devanur et al. 2008] for Fisher's model. We develop new methods to carefully deal with the flows and surpluses during price adjustments. In our algorithm, we need $O(n^6 \log(nU))$ maximum flow computations, where n is the number of persons and U is the maximum integer utility, and the length of the numbers is at most $O(n \log(nU))$ to guarantee an exact solution. The previous polynomial time algorithms [Nenakov and Primak 1983, Jain 2007, Ye 2007] for this problem are based on solving convex programs using the ellipsoid algorithm or interior-point method.

1 Introduction

We provide the first combinatorial polynomial algorithm for computing the model of economic markets formulated by Walras in 1874 [15]. In this model, every person has an initial distribution of some goods and a utility function of all goods. The market clears at a set of prices if every person sells its initial goods and then uses its entire revenue to buy a bundle of goods with maximum utility. We want to find the market equilibrium in which every good is assigned a price so that the market clears. In 1954, two Nobel laureates, Arrow and Debreu [2], proved that the market equilibrium always exists if the utility functions are concave, which is why this model is usually called "Arrow-Debreu market". But, their proof is based on Kakutani's fixed point theorem and non-constructive. Since then, many algorithmic results studied the linear version of this model, that is, all utility functions are linear.

The first polynomial time algorithm for the linear Arrow-Debreu model is given by [13,11]; it is based on solving a convex program using the ellipsoid algorithm. There are non-combinatorial algorithms which are faster than our algorithm, e.g., the one with time bound of $O(n^4 \log U)$ given by Ye [16] using the interior-point method. However, our algorithm has the advantage of being quite simple (see Figure 1 for a complete listing) and combinatorial, and hence, gives additional insight in the nature of the problem. We obtain equilibrium

* A full version of this paper is available at http://arxiv.org/abs/1212.0979

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 425–436, 2013.

prices by a simple procedure that iteratively adjusts prices and allocations in a carefully chosen, but intuitive manner. Previous to our algorithms, combinatorial algorithms were only known for computing approximate equilibria for the Arrow-Debreu model. Devanur and Vazirani [8] gave an approximation scheme with running time $O(\frac{n^4}{\epsilon} \log \frac{n}{\epsilon})$, improving [12]. Recently, Ghiyasvand and Orlin [10] improved the running time to $O(\frac{n}{\epsilon}(m + n \log n))$.

Many combinatorial algorithms consider a simpler model proposed by Fisher (see [3]), in which every buyer possesses an initial amount of money instead of some goods. Eisenberg and Gale [9] reduced the problem of computing the Fisher market equilibrium to a concave cost maximization problem and thus gave the first polynomial algorithm for the Fisher market by ellipsoid algorithm. The first combinatorial polynomial algorithm for the linear Fisher market equilibrium is given by Devanur et al [7]. When the input data is integral, the running time of their algorithm is $O(n^8 \log U + n^7 \log e_{max})$, where n is the number of buyers, U the largest integer utility, and e_{max} the largest initial amount of money of a buyer. Recently, Orlin [14] improved the running time for computing the linear Fisher model to $O(n^4 \log U + n^3 \log e_{max})$ and also gave the first strongly polynomial algorithm with running time $O(n^4 \log n)$.

We first define the model we will use in this paper, then discuss our main contributions.

1.1 Model and Definitions

We make the following assumptions on the model as in Jain's paper [11]:

1. There are n persons in the system. Each person i has only one good, which is different from the goods other people have. The good person i has is denoted by good i.
2. Each person has only one unit of good. So, if the price of good i is p_i, person i will obtain p_i units of money when selling its good.
3. Each person i has a linear utility function $\sum_j u_{ij} z_{ij}$, where z_{ij} is the amount of good j consumed by i.
4. Each u_{ij} is an integer between 0 and U.
5. For all i, there is a j such that $u_{ij} > 0$. (Everybody likes some good.)
6. For all j, there is an i such that $u_{ij} > 0$. (Every good is liked by somebody.)
7. For every proper subset P of persons, there exist $i \in P$ and $j \notin P$ such that $u_{ij} > 0$.

All these assumptions, with the exception of the last, are without loss of generality. The last assumption implies that all the equilibrium prices are nonzero [11]. The discuss more about the last assumption will appear in the full version of this paper.

Let $p = (p_1, p_2, ..., p_n)$ denote the vector of prices of goods 1 to n, so they are also the budgets of persons 1 to n. In this paper, we denote the set of all buyers by $B = \{b_1, b_2, ..., b_n\}$ and the set of all goods by $C = \{c_1, c_2, ..., c_n\}$. So, if the price of goods c_i is p_i, buyer b_i will have p_i amount of money. For a subset

B' of persons or a subset C' of goods, we also use $p(B')$ or $p(C')$ to denote the total prices of the goods the persons in B' own or the goods in C'. For a vector $v = (v_1, v_2, ..., v_k)$, let:

- $|v| = |v_1| + |v_2| + ... + |v_k|$ be the l_1-norm of v.
- $\|v\| = \sqrt{v_1^2 + v_2^2 + ... + v_k^2}$ be the l_2-norm of v.

It can be shown that each person only buys their favorite goods to maximize their utilities, that is, the goods with the maximum ratio of utility and price. Define its *bang per buck* to be $\alpha_i = \max_j\{u_{ij}/p_j\}$. The classical Arrow-Debreu [2] theorem says that there is a non-zero market clearing price vector, in which all the goods are sold and every buyer spends all its money to obtain a bundle of goods maximizing its utility.

For the current price vector p, the "equality graph" is a flow network $G = (\{s,t\} \cup B \cup C, E_G)$, where s is the source node and t is the sink node, then $B = \{b_1, ..., b_n\}$ denotes the set of buyers and $C = \{c_1, ..., c_n\}$ denotes the set of goods. E_G consists of:

- Edges from s to every node b_i in B with capacity p_i.
- Edges from every node c_i in C to t with capacity p_i.
- Edges from b_i to c_j with infinite capacity if $u_{ij}/p_j = \alpha_i$. Call these edges "equality edges".

So, our aim is to find a price vector p such that there is a flow in which all edges from s and to t are saturated, i.e., $(s, B \cup C \cup t)$ and $(s \cup B \cup C, t)$ are both minimum cuts. When this is satisfied, all goods are sold and all of the money earned by every person is spent.

In a flow f, define the surplus $r(b_i)$ of a buyer i to be the residual capacity of the edge (s, b_i), and define the surplus $r(c_j)$ of a good j to be the residual capacity of the edge (c_j, t). That is, $r(b_i) = p_i - \sum_j f_{ij}$, and $r(c_j) = p_j - \sum_i f_{ij}$, where f_{ij} is the amount of flow in the edge (b_i, c_j). Define the surplus vector of buyers to be $r(B) = (r(b_1), r(b_2), ..., r(b_n))$. Also, define the total surplus to be $|r(B)| = \sum_i r(b_i)$, which is also $\sum_j r(c_j)$ since the total capacity from s and to t are both equal to $\sum_i p_i$. For convenience, we denote the surplus vector of flow f' by $r'(B)$. In the network corresponding to market clearing prices, the total surplus of a maximum flow is zero.

1.2 Overview of Our Algorithm

The overall structure of our algorithm is similar to the ones of Devanur et al. [7] and Orlin [14] for computing equilibrium prices in Fisher markets, however, the details are quite different. The algorithm works iteratively. It starts with all prices equal to one. In each iteration it adjusts prices and allocations. The adjustment is guided by the analysis of a maximum flow in the equality graph.

In each iteration we first compute a balanced maximum flow [7]. A balanced maximum flow[1] is a maximum flow that minimizes the l_2-norm of the surplus vector $r(B)$. We then order the buyers in order of decreasing surpluses: $b_1, \ldots, b_n$. We find the minimal i such that $r(b_i)$ is substantially larger (by a factor of $1 + 1/n$) than $r(b_{i+1})$; $i = n$ if there is no such i. Let $B' = \{b_1, \ldots, b_i\}$ and let $\Gamma(B')$ be the goods that are adjacent to a node in B' in the equality graph. Since the flow is balanced, there is no flow from the buyers in $B \setminus B'$ to the goods in $\Gamma(B')$. We raise the prices of and the flows[2] into the nodes of $\Gamma(B')$ by a common factor x. This affects the surpluses of the buyers, some go up and some go down. More precisely, there are four kind of buyers, depending on whether a buyer belongs to B' or not and on whether the good owned by the buyer belongs to $\Gamma(B')$ or not. We increase the prices until one of three events happens: (1) a new edge enters the equality graph[3] (2) the surplus of a buyer in B' and a buyer in $B \setminus B'$ becomes equal, or (3) x reaches a substantial value ($1 + 1/(n^3)$ in our algorithm).[4] This ends the description of an iteration.

In what sense are we making progress? The l_2-norm of the surplus vector does not decrease in every iteration.[5] In (3), the l_2-norm may increase. However, also at least one price increases significantly. Since we can independently upper bound the prices, we can bound the number of iterations in which event (3) occurs, and as a consequence, the total increase of the l_2-norm of the surplus vector. When event (1) or (2) occurs, the l_2-norm of the surplus vector decreases substantially, since surplus moves from a buyer in B' to a buyer in $B \setminus B'$ and buyers in these two groups have, by the choice of groups, substantially different surpluses.

We continue until the l_2-norm of the surplus vector is sufficiently small, so that a simple rounding procedure yields equilibrium prices.

1.3 Other Results

There are also algorithms considering Arrow-Debreu model with non-linear utilities [6,5]. The CES (constant elasticity of substitution) utility functions have drawn much attention, where the utility functions are of the form $u(x_1, , x_n) = (\sum_{j=1}^{n} c_j x_j^p)^{1/p}$ for $-\infty < p < 1$ and $p \neq 0$. [5] has shown that for $p > 0$ and $-1 \leq p < 0$, there are polynomial algorithms by convex program. However, Chen, Paparas and Yannakakis [4] have shown that it is PPAD-hard to solve market equilibrium of CES utilities for $p < -1$. They also define a new concept "Non-monotone utilities", and show the PPAD-hardness to solve the markets

[1] In contrast to [7] the balanced maximum flow is not unique.

[2] In [7,14] only prices are raised and flows stay the same. This works for Fisher's model because budgets are fixed. However, in the Arrow-Debreu model, an increase of prices of goods will also increase the budgets of their owners.

[3] The increase of prices of goods in $\Gamma(B')$ makes the goods in $C \setminus \Gamma(B')$ more attractive and hence an equality edge connecting a buyer in B' with a good in $C \setminus \Gamma(B')$ may come into existence. This event also exists in [7,14].

[4] Events (2) and (3) have no parallel in [7,14].

[5] In [7] the balance is strictly decreasing.

with non-monotone utilities. It remains open to find the exact border between tractable and intractable utility functions.

2 The Algorithm

As in [7], our algorithm finds a balanced flow and increases the prices in the "active subgraph". But, in the Arrow-Debreu model, when we increase the prices of some good i, the budget of buyer i will also increase. So, we need to find a careful way to prevent the total surplus from increasing.

2.1 Balanced Flow

Definition 1. *In the network G of current p, a* balanced flow *is a maximum flow that minimizes $\|r(B)\|$ over all choices of maximum flows.*

For flows f and f' and their surplus vectors $r(B)$ and $r'(B)$, respectively, if $\|r(B)\| < \|r'(B)\|$, then we say f is *more balanced* than f'. The next lemma shows why it is called "balanced".

Lemma 1. *[7] If $a \geq b_i \geq 0, i = 1, 2, ..., k$ and $\delta \geq \sum_{i=1}^{k} \delta_i$, where $\delta, \delta_i \geq 0$, $i = 1, 2, ..., k$, then:*

$$\|(a, b_1, b_2, ..., b_k)\|^2 \leq \|(a + \delta, b_1 - \delta_1, b_2 - \delta_2, ..., b_k - \delta_k)\|^2 - \delta^2. \tag{1}$$

Proof.

$$(a + \delta)^2 + \sum_{i=1}^{k} (b_i - \delta_i)^2 - a^2 - \sum_{i=1}^{k} b_i^2 \tag{2}$$

$$\geq 2a\delta + \delta^2 - 2\sum_{i=1}^{k} b_i \delta_i \geq \delta^2 + 2a(\delta - \sum_{i=1}^{k} \delta_i) \geq \delta^2. \tag{3}$$

Lemma 2. *[7] In the network G for a price vector p, given a maximum flow f, a balanced flow f' can be computed by at most n max-flow computations.*

Proof. In the residual graph G_f w.r.t. f, let $S \subseteq B \cup C$ be the set of nodes reachable from s, and let $T = (B \cup C) \setminus S$ be the remaining nodes. Then, there are no edges from $S \cap B$ to $T \cap C$ in the equality graph, and there is no flow from $T \cap B$ to $S \cap C$. The buyers in $T \cap B$ and the goods in $S \cap C$ have no surplus w.r.t. f, and this holds true for every maximum flow. Let G' be the network spanned by $s \cup S \cup t$, and let f' be the balanced maximum flow in G'. The f' can be computed by n max-flow computations. (Corollary 8.8 in [7] is applicable since $(s \cup S, t)$ is a min-cut in G'.) Finally, f' together with the restriction of f to $s \cup T \cup t$ is a balanced flow in G.

The surpluses of all goods in f' are the same as those in f since we only balance the surpluses of buyers.

2.2 Price Adjustment

We need to increase the prices of some goods to get more equality edges ([7,14]). For a subset of buyers B_1, define its neighborhood $\Gamma(B_1)$ in the current network to be: $\Gamma(B_1) = \{c_j \in C | \exists b_i \in B_1, s.t.\ (b_i, c_j) \in E_G\}$. Clearly, there is no edge in G from B_1 to $C \setminus \Gamma(B_1)$. In a balanced flow f, given a surplus bound $S > 0$, let $B(S)$ denote the subset of buyers with surpluses at least S, that is, $B(S) = \{b_i \in B | r(b_i) \geq S\}$. We can see the goods in $\Gamma(B(S))$ have no surplus.

Lemma 3. *In a balanced flow f, given a surplus bound S, there is no edge that carries flow from $B \setminus B(S)$ to $\Gamma(B(S))$.*

Proof. Suppose there is such an edge (b_i, c_j) that carries flow such that $b_i \notin B(S)$ and $c_j \in \Gamma(B(S))$. Then, in the residual graph, there are directed edges (b_k, c_j) and (c_j, b_i) with nonzero capacities in which $b_k \in B(S)$. However, $r(b_k) \geq S > r(b_i)$, so we can augment along this path and get a more balanced flow, contradicting that f is already a balanced flow.

From Lemma 3, we can increase the prices in $\Gamma(B(S))$ by the same factor x without inconsistency. There is no edge from $B(S)$ to $C \setminus \Gamma(B(S))$, and the edges from $B \setminus B(S)$ to $\Gamma(B(S))$ are not carrying flow, and hence, there will be no harm if they disappear from the equality graph. If there are edges (b_i, c_j) and (b_i, c_k) where $b_i \in B(S), c_j, c_k \in \Gamma(B(S))$, then $u_{ij}/p_j = u_{ik}/p_k$. Since the prices in $\Gamma(B(S))$ are multiplied by a common factor x, u_{ij}/p_j and u_{ik}/p_k remain equal after a price adjustment. However, the goods in $C \setminus \Gamma(B(S))$ will become more attractive, so there may be edges from $B(S)$ to $C \setminus \Gamma(B(S))$ entering the network, and the increase of prices needs to stop when this happens. Define such a factor to be $X(S)$, that is,

$$X(S) = \min\{\frac{u_{ij}}{p_j} \cdot \frac{p_k}{u_{ik}} | b_i \in B(S), (b_i, c_j) \in E_G, c_k \notin \Gamma(B(S))\}. \tag{4}$$

So, we need $O(n^2)$ multiplications/divisions to compute $X(S)$. When we increase the prices of the goods in $\Gamma(B(S))$ by a common factor $x \leq X(S)$, the equality edges in $B(S) \cup \Gamma(B(S))$ will remain in the network. We will also need the following theorem to prevent the total surplus from increasing.

Theorem 1. *Given a balanced flow f in the current network G and a surplus bound S, we can multiply the prices of goods in $\Gamma(B(S))$ with a parameter $x > 1$. When $x \leq \min_i\{p_i/(p_i - r(b_i)) | b_i \in B(S), c_i \notin \Gamma(B(S))\}$ and $x \leq X(S)$, we obtain a flow f' in the new network G' of adjusted prices with the same value of total surplus by:*

$$f'_{ij} = \begin{cases} x \cdot f_{ij} & \text{if } c_j \in \Gamma(B(S)); \\ f_{ij} & \text{if } c_j \notin \Gamma(B(S)). \end{cases}$$

Then, the surplus of each good remains unchanged, and the surpluses of the buyers become:

$$r'(b_i) = \begin{cases} x \cdot r(b_i) & \text{if } b_i \in B(S), c_i \in \Gamma(B(S)); \\ (1-x)p_i + x \cdot r(b_i) & \text{if } b_i \in B(S), c_i \notin \Gamma(B(S)); \\ (x-1)p_i + r(b_i) & \text{if } b_i \notin B(S), c_i \in \Gamma(B(S)); \\ r(b_i) & \text{if } b_i \notin B(S), c_i \notin \Gamma(B(S)). \end{cases}$$

We call these kinds of buyers type 1 to type 4 buyers, respectively.

Proof. Since the flows on all edges associated with goods in $\Gamma(B(S))$ are multiplied by x, the surplus of each good in $\Gamma(B(S))$ remains zero. Only the surplus of type 2 buyers decreases because the flows from a type 2 buyer b_i are multiplied by x, but its budget p_i is not changed. The flow after adjustment is $x(p_i - r(b_i))$. We need this to be at most p_i, so $x \leq p_i/(p_i - r(b_i))$ for all type 2 buyers b_i, and in f', the new surplus $r'(b_i) = (1-x)p_i + x \cdot r(b_i)$.

Since both money and flows are multiplied by x for a type 1 buyer, their surplus is also multiplied by x. For a type 3 buyer b_i, their flows are not changed, but their money is multiplied by x, so the new surplus is $xp_i - (p_i - r(b_i))$.

After each price adjustment, in the new network, we will find a maximum flow by augmentation on the adjusted flow f' and then find a balanced flow by Lemma 2. This guarantees that when the surplus of a good becomes zero, it will not change back to non-zero anymore. Thus, the prices of the goods with non-zero surpluses have not been adjusted.

Property 1. The prices of goods with non-zero surpluses remain unchanged in the algorithm.

2.3 Whole Procedure

The whole algorithm is shown in Figure 1, where K is a constant we will set later. In this section, one iteration denotes the execution of one entire iteration inside the loop. The rounding procedures and termination conditions are given by the following two lemmas, whose proofs will be discussed in the full version of this paper.

Lemma 4. *When the total surplus is at least* $\frac{1}{4n^4U^{3n}} = \epsilon$*, we can slightly adjust prices to rational numbers of length* $O(n \log(nU))$*, so that in its adjusted flow, the* l_2*-norm of the surplus vector only increases by a factor of* $1 + O(1/n^4)$*.*

Lemma 5. *When the total surplus is* $< \frac{1}{4n^4U^{3n}} = \epsilon$ *in a flow* f*, we can obtain an exact solution from the current equality graph in polynomial time.*

In the first iteration, we construct a balanced flow f in the network where all prices are equal to 1. In the equality graph, we have at least one edge incident to every buyer. The total surplus will be bounded by n, actually $n-1$ as at least one good will be sold completely. From Theorem 1, in the execution of the algorithm, the total surplus will never increase.

Initially set $p_i = 1$ for all goods i;
Repeat
  Construct the network G for the current p, and compute the balanced flow f in it;
  Sort all buyers by their surpluses in decreasing order: $b_1, b_2, ..., b_n$;
  Find the first i in which $\frac{r(b_i)}{r(b_{i+1})} > 1 + 1/n$, and $i = n$ when there is no such i;
  Let $S = r(b_i)$ and obtain $B(S), \Gamma(B(S)), X(S)$; $(B(S) = \{b_1, b_2, ..., b_i\})$
  Multiply the prices in $\Gamma(B(S))$ by a gradually increasing factor $x > 1$ until:
  (Let f' be the flow corresponding to x which is constructed according to Theorem 1.)
    New equality edges emerge (x reaches $X(S)$);
    OR the surplus of a buyer $\in B(S)$ and a buyer $\notin B(S)$ equals in f';
    OR x reaches $1 + \frac{1}{K \cdot n^3}$
  Round the prices in $\Gamma(B(S))$ according Lemma 4;
Until $|r(B)| < \epsilon$, where $\epsilon = \frac{1}{4n^4 U^{3n}}$;
Finally, round the prices according to Lemma 5 to get an exact solution.

Fig. 1. The whole algorithm

To ensure that the algorithm will terminate in a polynomial number of steps, we will require the following lemmas. From Property 1, the prices of goods with surplus stay one during the whole algorithm, so there is still a good with price one in the end. And, we need to bound the largest price:

Lemma 6. *The prices of goods are at most* $(nU)^{n-1}$ *throughout the algorithm.*

Proof. It is enough to show that during the entire algorithm, for any non-empty and proper subset $\hat{C}$ of goods, there are goods $c_i \in \hat{C}, c_j \notin \hat{C}$ such that $p_i/p_j \leq nU$. So, when we sort all the prices in decreasing order, the ratio of two adjacent prices is at most nU. Since there is always a good with price 1, the largest price is $\leq (nU)^{n-1}$.

If $\hat{C}$ contains goods with surpluses, then their price is 1. The claim follows.

Let $\hat{B} = \Gamma(\hat{C})$ be the set of buyers adjacent to goods in $\hat{C}$ in the equality graph. If there exist b_i, c_j s.t. $b_i \in \hat{B}, c_j \notin \hat{C}$ and $u_{ij} > 0$, let $c_k \in \hat{C}$ be one of the goods adjacent to b_i in the equality graph, and then $u_{ij}/p_j \leq u_{ik}/p_k$. So, $p_k/p_j \leq u_{ik}/u_{ij} \leq U$.

If there do not exist such b_i, c_j, then the buyers in $\hat{B}$ do not like any goods in $C \setminus \hat{C}$, and there is $b_k \notin \hat{B}$, but $c_k \in \hat{C}$. Otherwise the persons whose goods are in $\hat{C}$ will not like any goods not in $\hat{C}$, contradicting assumption (7). Let $B' = \{j | b_j \in \hat{B}, c_j \notin \hat{C}\}$ and $B'' = \{j | b_j \notin \hat{B}, c_j \in \hat{C}\}$. We have:

$$p_k \leq p(B'') = p(\hat{C}) - p(\{j | b_j \in \hat{B}, c_j \in \hat{C}\}) \tag{5}$$

$$\leq p(\hat{B}) - p(\{j | b_j \in \hat{B}, c_j \in \hat{C}\}) = p(B'). \tag{6}$$

The inequality of the second line holds since goods in $\hat{C}$ have surplus 0 and all of the flows from $\hat{B}$ go to $\hat{C}$. Thus, B' must be non-empty, and hence, there is a $j \in B'$ with $p_j \geq p(B')/n$. We conclude $p_k \leq np_j$.

By Lemma 5, we can round to the exact solution when the algorithm terminates. To analyze the correctness and running time, we need the following lemma:

Lemma 7. *After every price adjustment by x, the l_2-norm of the surplus vector $\|r(B)\|$ will either*

- *be multiplied by a factor of $1 + O(1/n^3)$ when $x = 1 + \frac{1}{Kn^3}$, or*
- *be divided by a factor of $1 + \Omega(1/n^3)$.*

Note that by Lemma 4, the rounding procedure can only increase $\|r(B)\|$ by a factor of $1 + O(1/n^4)$.

Theorem 2. *In total, we need to compute $O(n^6 \log(nU))$ maximum flows, and the length of numbers is bounded by $O(n \log(nU))$. Thus, if we use the common $O(n^3)$ max-flow algorithm (see [1]), the total running time is $O(n^{10} \log^2(nU))$.*

Proof. By Lemma 6, every price can be multiplied by $x = 1 + \frac{1}{Kn^3}$ for $O(\log_{1+1/(Kn^3)}(nU)^n) = O(n^4 \log(nU))$ times, so the total number of iterations of the first type in Lemma 7 is $O(n^5 \log(nU))$. The total factor multiplied to $\|r(B)\|$ by the first type iterations is $(1 + O(1/n^3))^{O(n^5 \log(nU))}$.

At the beginning, $\|r(B)\| \leq \sqrt{n}$. When the algorithm terminates, $\|r(B)\| < \epsilon = \frac{1}{4n^4U^{3n}}$, so the number of second type iterations is bounded by

$$\log_{1+\Omega(1/n^3)}(\frac{1}{\epsilon}\sqrt{n}(1 + O(1/n^3))^{O(n^5 \log(nU))}) = O(n^5 \log(nU)). \tag{7}$$

Thus, the total number of iterations performed is bounded by $O(n^5 \log(nU))$. Since we need to compute n max-flows for the balanced flow in every iteration by Lemma 2, we need $O(n^6 \log(nU))$ maximum flow computations in total. By Lemma 6 and Lemma 4, the length of the numbers to be handled is bounded by $O(n \log(nU))$. Note that max-flow computations only need additions and subtractions. We perform multiplications and divisions when we scale prices and when we set up the max-flow computation in the computation of balanced flow. The numbers of multiplications/divisons is by a factor n less than the numbers of additions/subtractions, and hence, it suffices to charge $O(n \log(nU))$ per arithmetic operation.

Next we will prove Lemma 7. When we sort all the buyers by their surpluses $b_1, b_2, ..., b_n$ in decreasing order, b_1 is at least $|r(B)|/n$ (where $|r(B)|$ is the total surplus). So, for the first i in which $\frac{r(b_i)}{r(b_{i+1})} > 1+1/n$, we can see $\frac{r(b_j)}{r(b_{j+1})} \leq 1+1/n$ for $j < i$, so $r(b_i) \geq r(b_1)(1 + 1/n)^{-n} > |r(B)|/(e \cdot n)$. When such an i does not exist, each $r(b_i)$ is larger than $|r(B)|/(e \cdot n)$, and all goods in $\Gamma(B)$ must have zero surplus because the flow is otherwise not maximum. Thus, there are goods that have no buyers, and hence, either new equality edges emerge, or x reaches $1 + \frac{1}{Kn^3}$.

From the algorithm, in every iteration, x satisfies the following conditions:

1. $x \leq 1 + \frac{1}{Kn^3}$.
2. In f', $r'(b) \geq r'(b')$ for all $b \in B(S), b' \notin B(S)$. Here, $r'(b)$ is the surplus of b w.r.t. f', the flow corresponding to x by Theorem 1.
3. If $x < 1 + \frac{1}{Kn^3}$, the following possibilities arise:

(a) There is a new equality edge (b_i, c_j) with $b_i \in B(S), c_j \notin \Gamma(B(S))$. By Lemma 8 below, we can obtain a flow f'' in which either $r''(b_i) = r'(b_i) - p_j$, or there is a $b_k \notin B(S)$ with $r''(b_i) = r''(b_k)$ (same as (b)).
(b) When x satisfies the second requirement in the algorithm, it satisfies: there exist $b \in B(S)$ and $b' \notin B(S)$ such that $r'(b) = r'(b')$ in f'.

Lemma 8. *If there is a new equality edge (b_i, c_j) with $b_i \in B(S), c_j \notin \Gamma(B(S))$, we can obtain a flow f'' from f' (without increasing the total surplus) in which either $r''(b_i) = r'(b_i) - p_j$, or there is a $b_k \notin B(S)$ with $r''(b_i) = r''(b_k)$.*

Proof. Let $B' \subseteq B \setminus B(S)$ be the set of buyers with flows to c_j in f', and let w be the largest surplus of a buyer in $B \setminus B(S)$. Run the following procedure (f'' denotes the current flow in the algorithm):

Augment along (b_i, c_j) gradually until:
  $r''(b_i) = w$ or $r''(c_j) = 0$;
If $r''(b_i) = w$ then Exit;
For all $b_k \in B'$ in any order
  Augment along (b_i, c_j, b_k) gradually until:
    $r''(b_i) = \max\{r''(b_k), w\}$ or $f''(b_k, c_j) = 0$;
  Set $w = \max\{r''(b_k), w\}$;
  If $r''(b_i) = w$ then Exit.

During the procedure, the surplus of b_i decreases but cannot become less than the surplus of a buyer in $B \setminus B(S)$, so condition (2) holds. In the end, if $r''(b_i) = w$, then there is a $b_k \in B \setminus B(S)$ s.t. $r''(b_i) = r''(b_k)$; otherwise, c_j has no surplus, and the flows to it all come from b_i, so $r''(b_i) = r'(b_i) - p_j$.

From Theorem 1, the surpluses in f' will increase for type 1 and 3 buyers, will decrease for type 2 buyers, and will stay unchanged for type 4 buyers. Note that the surplus of a type 1 or 2 buyer cannot be smaller than the surplus of any type 3 or 4 buyer. From Theorem 1 and Lemma 8, we infer that the total surplus will not increase, type 2 and 3 buyers will get more balanced, and $r'(b) = x \cdot r(b)$ for type 1 buyers b, so $\|r'(B)\| \le x\|r(B)\| = (1 + O(1/n^3))\|r(B)\|$.

In (3a), there is a new equality edge (b_i, c_j). After the procedure described in Lemma 8, if there is no $b_k \notin B(S)$ such that $r''(b_i) = r''(b_k)$, then $r''(b_i) = r'(b_i) - p_j$ ($p_j \ge 1$). For all $b_k \notin B(S)$, $r''(b_i) > r''(b_k)$, and $r''(b_k) = r'(b_k) + \delta_k$, where $\delta_k \ge 0$ and $\sum_{b_k \notin B(S)} \delta_k \le p_j$. Because $|r(B)| \le n$, $\|r(B)\|^2 \le n^2$. By Lemma 1,

$$\|r''(B)\|^2 \le \|r'(B)\|^2 - p_j^2 \tag{8}$$
$$\le x^2\|r(B)\|^2 - 1 \tag{9}$$
$$\le x^2\|r(B)\|^2 - \frac{1}{n^2}\|r(B)\|^2 \tag{10}$$
$$= (1 - \Theta(1/n^2))\|r(B)\|^2. \tag{11}$$

So, we have $\|r''(B)\| = (1 - \Omega(1/n^2))\|r(B)\|$.

In (3a), after the procedure described in Lemma 8, if there is $b_k \notin B(S)$ such that $r''(b_i) = r''(b_k)$, then we are in a similar situation as in (3b), possibly with an even smaller total surplus. So, we can prove this case by the proof of (3b).

In (3b), let $u_1, u_2, ..., u_k$ and $v_1, v_2, ..., v_{k'}$ be the list of original surpluses of type 2 and 3 buyers, respectively. Define $u = \min\{u_i\}, v = \max\{v_j\}$, so $u_i \geq u$ for all i, and $v_j \leq v$ for all j, and $u > (1 + 1/n)v$. After the price and flow adjustments in Theorem 1, the list of surpluses will be $u_1 - \delta_1, u_2 - \delta_2, ..., u_k - \delta_k$ and $v_1 + \delta'_1, v_2 + \delta'_2, ..., v_k + \delta'_{k'}$ (here $\delta_i, \delta'_j \geq 0$ for all i, j), and there exist I, J such that $u_I - \delta_I = v_J + \delta'_J$, where $u_I - \delta_I$ is the smallest among $u_i - \delta_i$, and $v_J + \delta'_J$ is the largest among $v_j + \delta'_j$ by condition (2). Since the surpluses of type 1 buyers also increase (and the total surplus may decrease), we have $\sum_i \delta_i \geq \sum_j \delta'_j$, $\delta_I \leq \sum_i \delta_i$, and $\delta'_J \leq \sum_j \delta'_j$. Compute:

$$\sum_i (u_i - \delta_i)^2 + \sum_j (v_j + \delta'_j)^2 - (\sum_i u_i^2 + \sum_j v_j^2) \tag{12}$$

$$= -2\sum_i u_i\delta_i + 2\sum_j v_j\delta'_j + \sum_i \delta_i^2 + \sum_j \delta'^2_j \tag{13}$$

$$\leq -u\sum_i \delta_i + v\sum_j \delta'_j - \sum_i \delta_i(u_i - \delta_i) + \sum_j \delta'_j(v_j + \delta'_j) \tag{14}$$

$$\leq -(u - v)\sum_i \delta_i - (u_I - \delta_I)\sum_i \delta_i + (v_J + \delta'_J)\sum_j \delta'_j \tag{15}$$

$$\leq -(u - v)\sum_i \delta_i \tag{16}$$

$$\leq -(u - v)\max\{\delta_I, \delta'_J\} \tag{17}$$

$$\leq -\frac{1}{2}(u - v)^2 \tag{18}$$

$$< -\frac{1}{2(n+1)^2}u^2. \tag{19}$$

Let $w_1, w_2, ...w_{k''}$ be the list of surpluses of type 1 buyers; all of them are $\leq e \cdot u$. After price adjustment, the surpluses will be $x \cdot w_1, x \cdot w_2, ...x \cdot w_{k''}$ from Theorem 1. Compute:

$$\sum_i (xw_i)^2 \leq (1 + \frac{1}{Kn^3})^2 \sum_i w_i^2 \tag{20}$$

$$\leq \sum_i w_i^2 + (\frac{2}{Kn^3} + \frac{1}{K^2n^6}) \cdot ne^2u^2 \tag{21}$$

$$= \sum_i w_i^2 + (\frac{2}{Kn^2} + \frac{1}{K^2n^5})e^2u^2. \tag{22}$$

Let $K = 32e^2$, then the change to the sum of squares of surpluses for type 2 and 3 buyers is less than $-\frac{1}{8n^2}u^2 = -\frac{4}{Kn^2}e^2u^2$. The total change to $\|r(B)\|^2$ is:

$$< (-\frac{2}{Kn^2} + \frac{1}{K^2n^5})e^2u^2. \tag{23}$$

Since $u \geq \frac{1}{e} r(b_i)$ for all buyers b_i, $nu^2 \geq \frac{1}{e^2}\|r(B)\|^2$. Since the change is negative:

$$\|r'(B)\|^2 < \|r(B)\|^2 + (-\frac{2}{Kn^2} + \frac{1}{K^2 n^5})\frac{1}{n}\|r(B)\|^2 \tag{24}$$

$$= \|r(B)\|^2 - \frac{2}{Kn^3}\|r(B)\|^2 + \frac{1}{K^2 n^6}\|r(B)\|^2 \tag{25}$$

$$= \|r(B)\|^2 (1 - \frac{1}{Kn^3})^2. \tag{26}$$

Thus, Lemma 7 is proved.

References

1. Ahuja, R.K., Magnati, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993)
2. Arrow, K.J., Debreu, G.: Existence of an equilibrium for a competitive economy. Econometrica 22, 265–290 (1954)
3. Brainard, W.C., Scarf, H.E.: How to compute equilibrium prices in 1891. Cowles Foundation Discussion Papers 1272, Cowles Foundation for Research in Economics, Yale University (August 2000)
4. Chen, X., Paparas, D., Yannakakis, M.: The complexity of non-monotone markets. CoRR, abs/1211.4918 (2012)
5. Codenotti, B., McCune, B., Penumatcha, S., Varadarajan, K.: Market equilibrium for ces exchange economies: existence, multiplicity, and computation. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 505–516. Springer, Heidelberg (2005)
6. Codenotti, B., Mccune, B., Varadarajan, K.: Market equilibrium via the excess demand function. In: Proceedings STOC 2005, pp. 74–83 (2005)
7. Devanur, N.R., Papadimitriou, C.H., Saberi, A., Vazirani, V.V.: Market equilibrium via a primal–dual algorithm for a convex program. J. ACM 55(5), 22:1–22:18 (2008)
8. Devanur, N.R., Vazirani, V.V.: An improved approximation scheme for computing Arrow-Debreu prices for the linear case. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 149–155. Springer, Heidelberg (2003)
9. Eisenberg, E., Gale, D.: Consensus of Subjective Probabilities: the Pari-mutuel Method. Defense Technical Information Center (1958)
10. Ghiyasvand, M., Orlin, J.B.: A simple approximation algorithm for computing Arrow-Debreu prices. To appear in Operations Research (2012)
11. Jain, K.: A polynomial time algorithm for computing an Arrow-Debreu market equilibrium for linear utilities. SIAM J. Comput. 37(1), 303–318 (2007)
12. Jain, K., Mahdian, M., Saberi, A.: Approximating market equilibria (2003)
13. Nenakov, E.I., Primak, M.E.: One algorithm for finding solutions of the arrow-debreu model. Kibernetica, 127–128 (1983)
14. Orlin, J.B.: Improved algorithms for computing Fisher's market clearing prices. In: Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, pp. 291–300. ACM, New York (2010)
15. Walras, L.: Elements of Pure Economics, or the theory of social wealth (1874)
16. Ye, Y.: A path to the Arrow-Debreu competitive market equilibrium. Math. Program. 111(1), 315–348 (2007)

Towards an Understanding of Polynomial Calculus: New Separations and Lower Bounds
(Extended Abstract)

Yuval Filmus[1], Massimo Lauria[2], Mladen Mikša[2], Jakob Nordström[2], and Marc Vinyals[2]

[1] University of Toronto, Toronto, Ontario M5S 2E4, Canada
[2] KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden

Abstract. During the last decade, an active line of research in proof complexity has been into the space complexity of proofs and how space is related to other measures. By now these aspects of resolution are fairly well understood, but many open problems remain for the related but stronger polynomial calculus (PC/PCR) proof system. For instance, the space complexity of many standard "benchmark formulas" is still open, as well as the relation of space to size and degree in PC/PCR.

We prove that if a formula requires large resolution width, then making XOR substitution yields a formula requiring large PCR space, providing some circumstantial evidence that degree might be a lower bound for space. More importantly, this immediately yields formulas that are very hard for space but very easy for size, exhibiting a size-space separation similar to what is known for resolution. Using related ideas, we show that if a graph has good expansion and in addition its edge set can be partitioned into short cycles, then the Tseitin formula over this graph requires large PCR space. In particular, Tseitin formulas over random 4-regular graphs almost surely require space at least $\Omega(\sqrt{n})$.

Our proofs use techniques recently introduced in [Bonacina-Galesi '13]. Our final contribution, however, is to show that these techniques provably cannot yield non-constant space lower bounds for the functional pigeonhole principle, delineating the limitations of this framework and suggesting that we are still far from characterizing PC/PCR space.

1 Introduction

Proof complexity studies how hard it is to provide succinct certificates for tautological formulas in propositional logic—i.e., proofs that formulas always evaluate to true under any truth value assignment, where these proofs are verifiable in time polynomial in their size. It is widely believed that there is no proof system where such efficiently verifiable proofs can always be found of size at most polynomial in the size of the formulas they prove. Showing this would establish $\mathsf{NP} \neq \mathsf{co\text{-}NP}$, and hence $\mathsf{P} \neq \mathsf{NP}$, and the study of proof complexity was initiated by Cook and Reckhow [16] as an approach towards this (still very distant) goal.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 437–448, 2013.

A second prominent motivation for proof complexity is the connection to applied SAT solving. By a standard transformation, any propositional logic formula F can be transformed to another formula F' in conjunctive normal form (CNF) such that F' has the same size up to constant factors and is unsatisfiable if and only if F is a tautology. Any algorithm for solving SAT defines a proof system in the sense that the execution trace of the algorithm constitutes a polynomial-time verifiable witness of unsatisfiability (such a witness is often referred to as a *refutation* rather than a *proof*, and we will use the two terms interchangeably in this paper). In the other direction, most modern SAT solvers can in fact be seen to search for proofs in systems studied in proof complexity, and upper and lower bounds for these proof systems hence give information about the potential and limitations of such SAT solvers.

In addition to running time, a major concern in SAT solving is memory consumption. In proof complexity, these two resources are modelled by *proof size/length* and *proof space*. It is thus interesting to understand these complexity measures and how they are related to each other, and such a study reveals intriguing connections that are also of intrinsic interest to proof complexity. In this context, it is natural to focus on proof systems at comparatively low levels in the proof complexity hierarchy that are, or could plausibly be, used as a basis for SAT solvers. Such proof systems include resolution and polynomial calculus. This paper takes as its starting point the former system but focuses on the latter.

Previous Work. The *resolution* proof system was introduced in [12], and is at the foundation of state-of-the-art SAT solvers based on so-called conflict-driven clause learning (CDCL) [4, 23]. In resolution, one derives new disjunctive clauses from the clauses of the original CNF formula until contradiction is reached. One of the early breakthroughs in proof complexity was the (sub)exponential lower bound on proof length (measured as the number of clauses in a proof) obtained by Haken [19]. Truly exponential lower bounds—i.e., bounds $\exp(\Omega(n))$ in the size n of the formula—were later established in [14, 25] and other papers.

Ben-Sasson and Wigderson [11] identified *width* as a crucial resource, where the width is the size of a largest clause in a resolution proof. They proved that strong lower bounds on width imply strong lower bounds on length, and used this to rederive essentially all known length lower bounds in terms of width.

The study of space in resolution was initiated by Esteban and Torán [17], measuring the space of a proof (informally) as the maximum number of clauses needed to be kept in memory during proof verification. Alekhnovich et al. [1] later extended the concept of space to a more general setting, including other proof systems. The (clause) space measure can be shown to be at most linear in the formula size, and matching lower bounds were proven in [1, 8, 17].

Atserias and Dalmau [3] proved that space is in fact lower-bounded by width, which allowed to rederive all hitherto known space lower bounds as corollaries of width lower bounds. A strong separation of the two measures was obtained in [9], exhibiting formulas with constant width complexity but almost linear space complexity. Also, dramatic space-width trade-offs have been shown in [7],

with formulas refutable in constant width and constant space where optimizing one of the measures causes essentially worst-case behaviour of the other.

Regarding the connections between length and space, it follows from [3] that formulas of low space complexity also have short proofs. For the subsystem of *tree-like resolution*, where each line in the proof can only be used once, [17] showed that length upper bounds also imply space upper bounds, but for general resolution [9] established that this is false in the strongest possible sense. Strong trade-offs between length and space were proven in [5, 10].

This paper focuses on the more powerful *polynomial calculus (PC)*[1] proof system [15], which is not at all as well understood. In a PC proof, clauses are interpreted as multilinear polynomials (expanded out to sums of monomials), and one derives contradiction by showing that these polynomials have no common root. Intriguingly, while proof complexity-theoretic results seem to hold out the promise that SAT solvers based on PC could be orders of magnitude faster than CDCL, such algebraic solvers have so far failed to be truly competitive.

Proof size[2] in PC is measured as the total number of monomials and the analogue of resolution space is the number of monomials needed in memory during verification of a proof. Resolution width translates into polynomial degree in PC. While length, space and width in resolution are fairly well understood, our understanding of the corresponding measures in PC is much more limited.

Impagliazzo et al. [21] showed that strong degree lower bounds imply strong size lower bounds. This is a parallel to the length-width relation in [11], and in fact this latter paper can be seen as a translation of [21] from PC to resolution. This size-degree relation has been used to prove exponential lower bounds on size in a number of papers, with [2] perhaps providing the most general setting.

The first lower bounds on space were reported in [1], but only sublinear bounds and only for formulas of unbounded width. The first space lower bounds for k-CNF formulas were presented in [18], and asymptotically optimal (linear) lower bounds were finally proven by Bonacina and Galesi [13]. However, there are several formula families with high resolution space complexity for which the PC space complexity has remained unknown, e.g., Tseitin formulas (encoding that the sum of all vertex degrees in an undirected graph must be even), ordering principle formulas, and functional pigeonhole principle (FPHP) formulas.

Regarding the relation between space and degree, it is open whether degree is a lower bound for space (the analogue of what holds in resolution) and also it has been unknown whether the two measures can be separated in the sense that there are formulas of low degree complexity requiring high space. However, [6] recently proved a space-degree trade-off analogous to the resolution space-width trade-off in [7] (in fact for the very same formulas). This could be interpreted as indicating

[1] Strictly speaking, to get a stronger proof system than resolution we need to look at the generalization *PCR* as defined in [1], but for simplicity we will be somewhat sloppy in this introduction in distinguishing between PC and PCR.

[2] The *length* of a proof is the number of lines, whereas *size* also considers the size of lines. In resolution the two measures are essentially equivalent. In PC size and length can be very different, however, and so size is the right measure to study.

that there should be a space-degree separation analogous to the space-width separation in resolution, and the authors of [13] suggest that their techniques might be a step towards understanding degree and proving that degree lower-bounds space, similar to how this was done for resolution width in [3].

As to size versus space in PC, essentially nothing has been known. It is open whether small space complexity implies small size complexity and/or the other way around. Some size-space trade-offs were recently reported in [6, 20], but these trade-offs are weaker than the corresponding results for resolution.

Our Results. We study the relation of size, space, and degree in PC (and the stronger system PCR) and present a number of new results as described below.

1. We prove that if the resolution width of refuting a CNF formula F is w, then by substituting each variable by an exclusive or of two new variables and expanding out we get a new CNF formula $F[\oplus]$ requiring PCR space $\Omega(w)$. In one sense, this is stronger than claiming that degree is a lower bound for space, since high width complexity is a necessary but not sufficient condition for high degree complexity. In another sense, however, this is (much) weaker in that XOR substitution can amplify the hardness of formulas substantially. Nevertheless, to the best of our knowledge this is the first result making any connection between width/degree and space for polynomial calculus.
2. More importantly, this result yields essentially optimal separations between length and degree on the one hand and space on the other. Namely, taking expander graphs and making double copies of all edges, we show that Tseitin formulas over such graphs have proofs in size $\mathrm{O}(n \log n)$ and degree $\mathrm{O}(1)$ in PC but require space $\Theta(n)$ in PCR. (Furthermore, since these small-size proofs are tree-like, this shows that there is no tight correlation between size and space in tree-like PC/PCR in contrast to resolution.)
3. Using related ideas, we also prove strong PCR space lower bounds for Tseitin formulas over (simple or multi-)graphs where the edge set can be partitioned into small cycles. (The two copies of every edge in the multi-graph above form such cycles, but this works in greater generality.) In particular, for Tseitin formulas over random d-regular graphs for $d \geq 4$ we establish that an $\Omega(\sqrt{n})$ PCR space lower bound holds asymptotically almost surely.
4. On the negative side, we show that the techniques in [13] cannot prove any non-constant PCR space lower bounds for functional pigeonhole principle (FPHP) formulas. That is, although these formulas require high degree and it seems plausible that they are hard also with respect to space, the machinery developed in [13] provably cannot establish such lower bounds. Unfortunately, this seems to indicate that we are further from characterizing degree in PC/PCR than previously hoped.

Organization of This Paper. The rest of this paper is organized as follows. We briefly review preliminaries in Section 2. In Section 3, we give a more detailed overview of our results and sketch some proofs. Section 4 contains some concluding remarks. Due to space constraints, most of the proofs are deferred to the full-length version of this paper.

2 Preliminaries

A *literal* over a Boolean variable x is either the variable x itself (a *positive literal*) or its negation $\neg x$ or $\overline{x}$ (a *negative literal*). It will also be convenient to use the alternative notation $x^0 = x$, $x^1 = \overline{x}$, where we identify 0 with true and 1 with false[3] (so that x^b is true if $x = b$). A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals. We denote the empty clause by $\bot$. A clause containing at most k literals is called a *k-clause*. A *CNF formula* $F = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses. A *k-CNF formula* is a CNF formula consisting of k-clauses.

Let $\mathbb{F}$ be a field and consider the polynomial ring $\mathbb{F}[x, \overline{x}, y, \overline{y}, \ldots]$ (where x and $\overline{x}$ are viewed as distinct formal variables). We write $[n] = \{1, \ldots, n\}$.

Definition 1 (Polynomial calculus resolution (PCR)). *A* PCR configuration $\mathbb{P}$ *is a set of polynomials in* $\mathbb{F}[x, \overline{x}, y, \overline{y}, \ldots]$. *A* PCR refutation *of a CNF formula F is a sequence of configurations* $\{\mathbb{P}_0, \ldots, \mathbb{P}_\tau\}$ *such that* $\mathbb{P}_0 = \emptyset$, $1 \in \mathbb{P}_\tau$, *and for* $t \in [\tau]$ *we obtain* $\mathbb{P}_t$ *from* $\mathbb{P}_{t-1}$ *by one of the following steps:*

Axiom download $\mathbb{P}_t = \mathbb{P}_{t-1} \cup \{p\}$, *where p is either a monomial* $m = \prod_i x_i^b$ *encoding a clause* $C = \bigvee_i x_i^b \in F$, *or a* Boolean axiom $x^2 - x$ *or* complementarity axiom $x + \overline{x} - 1$ *for any variable x (or $\overline{x}$).*

Inference $\mathbb{P}_t = \mathbb{P}_{t-1} \cup \{p\}$, *where p is inferred by* linear combination $\frac{q \quad r}{\alpha q + \beta r}$ *or* multiplication $\frac{q}{xq}$ *from polynomials* $q, r \in \mathbb{P}_{t-1}$ *for* $\alpha, \beta \in \mathbb{F}$ *and x a variable.*

Erasure $\mathbb{P}_t = \mathbb{P}_{t-1} \setminus \{p\}$, *where p is a polynomial in* $\mathbb{P}_{t-1}$.

If we drop complementarity axioms and encode each negative literal $\overline{x}$ *as the polynomial* $(1 - x)$, *the proof system is called* polynomial calculus (PC).

The size $S(\pi)$ *of a PC/PCR refutation* π *is the number of monomials (counted with repetitions) in all downloaded or derived polynomials in* π, *the* (monomial) space $Sp(\pi)$ *is the maximal number of monomials (counted with repetitions)*[4] *in any configuration in* π, *and the* degree $Deg(\pi)$ *is the maximal degree of any monomial appearing in* π. *Taking the minimum over all PCR refutations of a formula F, we define the size* $S_{\mathcal{PCR}}(F \vdash \bot)$, *space* $Sp_{\mathcal{PCR}}(F \vdash \bot)$, *and degree* $Deg_{\mathcal{PCR}}(F \vdash \bot)$ *of refuting F in PCR (and analogously for PC).*

We can also define *resolution* in this framework, where proof lines are always clauses (i.e., single monomials) and new clauses can be derived by the *resolution rule* inferring $C \vee D$ from $C \vee x$ and $D \vee \overline{x}$. The *length* of a resolution refutation π is the number of downloaded and derived clauses, the *space* is the maximal number of clauses in any configuration, and the *width* is the size of a largest clause appearing in π (or equivalently the degree of such a monomial). Taking the minimum over all refutations as above we get the measures $L_{\mathcal{R}}(F \vdash \bot)$, $Sp_{\mathcal{R}}(F \vdash \bot)$, and $W_{\mathcal{R}}(F \vdash \bot)$. It is not hard to show that PCR can simulate resolution efficiently with respect to all these measures.

[3] Note that this is the opposite of what is found in many other papers, but as we will see shortly it is the natural choice in the context of polynomial calculus.

[4] In [1], space is defined *without* repetitions. All our results hold in this setting as well.

We say that a refutation is *tree-like* if every line is used at most once as the premise of an inference rule before being erased (though it can possibly be rederived later). All measures discussed above can also be defined for restricted subsystems of resolution, PC and PCR admitting only tree-like refutations.

Let us now describe the formulas which will be the main focus of our study.

Definition 2 (Tseitin formula). *Let $G = (V, E)$ be an undirected graph and $\chi\colon V \to \{0,1\}$ be a function. Identify every edge $e \in E$ with a variable x_e and let $PARITY_{v,\chi}$ denote the CNF encoding of the constraint that the number of true edges x_e incident to a vertex $v \in V$ is equal to $\chi(v) \pmod 2$. Then the* Tseitin formula *over G with respect to f is $Ts(G,\chi) = \bigwedge_{v\in V} PARITY_{v,\chi}$.*

When the degree of G is bounded by d, $Ts(G,\chi)$ is a d-CNF formula with at most $2^{d-1}|V|$ clauses. We say that a vertex set U has *odd (even) charge* if $\sum_{u\in U}\chi(u)$ is odd (even). By a simple counting argument one sees that $Ts(G,\chi)$ is unsatisfiable if $V(G)$ has odd charge. Lower bounds on the hardness of refuting such unsatisfiable formulas $Ts(G,\chi)$ can be proven in terms of the expansion of G as defined next.

Definition 3 (Connectivity expansion [1]). *The* connectivity expansion *of $G = (V, E)$ is the largest c such that for every $E' \subseteq E$, $|E'| \leq c$, the graph $G' = (V, E \setminus E')$ has a connected component of size strictly greater than $|V|/2$.*

If F is a CNF formula and $f\colon \{0,1\}^d \to \{0,1\}$ is a Boolean function, then we can obtain a new CNF formula by substituting $f(x_1,\ldots,x_d)$ for every variable x and expanding out to conjunctive normal form. We write $F[f]$ to denote the resulting *substituted formula*, where we will be interested in substitutions with a particular kind of functions defined as follows.

Definition 4 (Non-authoritarian function [10]). *We say that a Boolean function $f(x_1,\ldots,x_d)$ is* non-authoritarian *if for every x_i and for every assignment α to x_i there exist α_0, α_1 extending α such that $f(\alpha_b) = b$ for $b \in \{0,1\}$.*

By way of example, exclusive or (XOR), denoted $\oplus$, is clearly non-authoritarian, since regardless of the value of one variable, the other one can be flipped to make the function true or false, but standard non-exclusive or $\vee$ is not.

Let us finally give a brief overview of the framework developed in [13], which we use to prove our PCR space lower bounds.[5] A *partial partition* $\mathcal{Q}$ of a variable set V is a collection of disjoint sets $Q_i \subseteq V$. We use the notation $\bigcup \mathcal{Q} = \bigcup_{Q_i\in\mathcal{Q}} Q_i$. For two sets of partial assignments H and H' to disjoint domains, we denote by $H \times H'$ the set of assignments $H \times H' = \{\alpha \cup \beta \mid \alpha \in H \text{ and } \beta \in H'\}$. A set of partial assignments H to the domain Q is *flippable* on Q if for each variable $x \in Q$ and $b \in \{0,1\}$ there exists an assignment $\alpha_b \in H$ such that $\alpha_b(x) = b$. We say that H *satisfies* a formula F if all $\alpha \in H$ satisfy F.

A *$\mathcal{Q}$-structured assignment set* is a pair $(\mathcal{Q}, \mathcal{H})$ consisting of a partial partition $\mathcal{Q} = \{Q_1,\ldots,Q_t\}$ of V and a set of partial assignments $\mathcal{H} = \prod_{i=1}^{t} H_i$, where

[5] The actual definitions that we use are slightly different but essentially equivalent.

each H_i assigns to and is flippable on Q_i. We write $(\mathcal{Q}, \mathcal{H}) \preccurlyeq (\mathcal{Q}', \mathcal{H}')$ if $\mathcal{Q} \subseteq \mathcal{Q}'$ and $\mathcal{H}'\restriction_{\mathcal{Q}} = \mathcal{H}$, where $\mathcal{H}'\restriction_{\mathcal{Q}} = \prod_{Q_i \in \mathcal{Q}} H_i'$. A structured assignment set $(\mathcal{Q}, \mathcal{H})$ *respects* a CNF formula F' if for every clause $C \in F'$ either $\mathit{Vars}(C) \cap \bigcup \mathcal{Q} = \emptyset$ or there is a set $Q \in \mathcal{Q}$ such that $\mathit{Vars}(C) \subseteq Q$ and $\mathcal{H}$ satisfies C.

Expressed in this language, the key technical definition in [13] is as follows.

Definition 5 (Extendible family). *A non-empty family $\mathcal{F}$ of structured assignment sets $(\mathcal{Q}, \mathcal{H})$ is* r-extendible *for a CNF formula F with respect to a satisfiable $F' \subseteq F$ if every $(\mathcal{Q}, \mathcal{H}) \in \mathcal{F}$ satisfies the following conditions.*

Size $|\mathcal{Q}| \leq r$.
Respectfulness $(\mathcal{Q}, \mathcal{H})$ *respects* F'.
Restrictability *For every $\mathcal{Q}' \subseteq \mathcal{Q}$ the restriction $(\mathcal{Q}', \mathcal{H}\restriction_{\mathcal{Q}'})$ is in $\mathcal{F}$.*
Extendibility *If $|\mathcal{Q}| < r$ then for every clause $C \in F \setminus F'$ there exists $(\mathcal{Q}', \mathcal{H}') \in \mathcal{F}$ such that 1. $(\mathcal{Q}, \mathcal{H}) \preccurlyeq (\mathcal{Q}', \mathcal{H}')$, 2. $\mathcal{H}'$ satisfies C, and 3. $|\mathcal{Q}'| \leq |\mathcal{Q}| + 1$.*

To prove PCR space lower bounds for a formula F, it is sufficient to find an extendible family for F. All space lower bounds presented in this paper are obtained in this manner, where in addition we always have $F' = \emptyset$.

Theorem 6 ([13]). *Suppose that F is a CNF formula which has an r-extendible family $\mathcal{F}$ with respect to some $F' \subseteq F$. Then $Sp_{\mathcal{PCR}}(F \vdash \bot) \geq r/4$.*

3 Overview of Results and Sketches of Some Proofs

In this section, we give a more detailed overview with formal statements of our results, and also provide some proof sketches in order to convey the main technical ideas. As a general rule, the upper bounds we state are for polynomial calculus (PC) whereas the lower bounds hold for the stronger system PCR.

Relating PCR Space and Resolution Width. The starting point of our work is the question of how space and degree are related in polynomial calculus, and in particular whether it is true that degree lower-bounds space. While this question remains wide open, we make partial progress by showing that if the resolution width of refuting a CNF formula F is large (which in particular must be the case if F requires high degree), then by making XOR substitution we obtain a formula $F[\oplus]$ that requires large PCR space. In fact, this works not only for exclusive or but for any non-authoritarian function (as defined in Definition 4). The formal statement is as follows.

Theorem 7. *Let F be a k-CNF formula and let f be any non-authoritarian function. Then $Sp_{\mathcal{PCR}}(F[f] \vdash \bot) \geq (W_{\mathcal{R}}(F \vdash \bot) - k + 1)/4$ holds over any field.*

Proof (sketch). In one sentence, the proof of Theorem 7 is by combining the concept of extendible families in Definition 5 with the combinatorial characterization of resolution width in [3]. We show that the properties of F implied by the width lower bound can be used to construct an extendible family for $F[f]$.

To make this description easier to parse, let us start by describing in somewhat more detail the width characterization in [3].

Consider the following game played on F by two players *Spoiler* and *Duplicator*. Spoiler asks about assignments to variables in F and Duplicator answers true or false. Spoiler can only remember ℓ assignments simultaneously, however, and has to forget some variable when this limit is reached. If Duplicator is later asked about some forgotten variable, the new assignment need not be consistent with the previous forgotten one. Spoiler wins the game by constructing a partial assignment that falsifies some clause in F, and the game is a Duplicator win if there is a strategy to keep playing forever without Spoiler ever reaching this goal. It was proven in [3] that this game exactly captures resolution width in the sense that Duplicator has a winning strategy if and only if $\ell \leq W_{\mathcal{R}}(F \vdash \bot)$.

Let us fix $r = W_{\mathcal{R}}(F \vdash \bot) - k + 1$ and use Duplicator's winning strategy for $\ell = W_{\mathcal{R}}(F \vdash \bot)$ to build an r-extendible family for $F[\oplus]$ (the proof for general non-authoritarian functions is very similar). Consider any assignment α reached during the game. We define a corresponding structured assignment set $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$ by adding a block $Q_x = \{x_1, x_2\}$ to $\mathcal{Q}_\alpha$ for every $x \in \mathrm{Dom}(\alpha)$, and let H_x contain all assignments α_x to $\{x_1, x_2\}$ such that $\alpha_x(x_1 \oplus x_2) = \alpha(x)$.

Given these structured assignment sets $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$, the family $\mathcal{F}$ is constructed inductively as follows. The base case is that $(\mathcal{Q}_\emptyset, \mathcal{H}_\emptyset) = (\emptyset, \emptyset)$ is in $\mathcal{F}$. To extend $(\mathcal{Q}_\alpha, \mathcal{H}_\alpha)$ to satisfy a clause in $C[\oplus]$, we simulate a Spoiler with memory α who asks about all variables in C. Since Duplicator does not falsify C, when all variables have been queried some literal in C must be satisfied by the assignment. Fix one such variable assignment $\{x = b\}$ and add $\left(\mathcal{Q}_{\alpha \cup \{x=b\}}, \mathcal{H}_{\alpha \cup \{x=b\}}\right)$ as defined above to $\mathcal{F}$. All that remains now is to verify that this yields an extendible family as described in Definition 5 and then apply Theorem 6.

Separation of Size and Degree from Space. It follows from Theorem 7 that there are formulas which have small PC refutations in constant degree but nevertheless require maximal space in PCR.

Theorem 8. *For any field $\mathbb{F}$ of characteristic p there is a family of k-CNF formulas F_n (where k depends on p) of size $\mathrm{O}(n)$ for which $Sp_{\mathcal{PCR}}(F_n \vdash \bot) = \Omega(n)$ over any field but which have tree-like PC refutations $\pi_n : F_n \vdash \bot$ over $\mathbb{F}$ of size $S(\pi_n) = \mathrm{O}(n \log n)$ and degree $Deg(\pi_n) = \mathrm{O}(1)$.*

Proof (sketch). Let us focus on $p = 2$. Consider a Tseitin formula $Ts(G, \chi)$ for any constant-degree graph G over n vertices with connectivity expansion $\Omega(n)$ and any odd-charge function χ.

From [11] we know that $W_{\mathcal{R}}(F \vdash \bot) = \Omega(n)$. It is not hard to see that XOR substitution yields another Tseitin formula $Ts(G', \chi)$ for the multi-graph G' obtained from G by adding double copies of all edges. This formula requires large PCR space (over any field) by Theorem 7. The upper bound follows by observing that the CNF encodes a linear system of equations, which is easily shown inconsistent in PC by summing up all equations in a tree-like fashion.

It follows from Theorem 8 that tree-like space in PC/PCR is not upper-bounded by tree-like size, in contrast to resolution. This is the only example we are aware of where the relations between size, degree, and space in PC/PCR differ from those between length, width, and space in resolution, so let us state this as a formal corollary.

Corollary 9. *It is* not true *in PC/PCR that tree-like space complexity is upper-bounded by the logarithm of tree-like size complexity.*

Space Complexity of Tseitin Formulas. A closer analysis of the proof of Theorem 8 reveals that it partitions the edge set of G' into small edge-disjoint cycles (namely, length-2 cycles corresponding to the two copies of each original edge) and uses partial assignments that all maintain the same parities of the vertices on a given cycle. It turns out that this approach can be made to work in greater generality as stated next.

Theorem 10. *Let $G = (V, E)$ be a connected graph of bounded degree d with connectivity expansion c such that E can be partitioned into cycles of length at most b. Then it holds over any field that $Sp_{\mathcal{PCR}}(Ts(G, \chi) \vdash \bot) \geq c/4b - d/8$.*

Proof (sketch). We build on the resolution space lower bound in [1, 17], where the proof works by inductively constructing an assignment α_t for each derived configuration $\mathbb{C}_t$ (which corresponds to removing edges from G and updating the vertex charges accordingly) such that (a) α_t satisfies $\mathbb{C}_t$, and (b) α_t does not create any odd-charge component in G of size less than $n/2$. The inductive update can be performed as long as the space is not too large, which shows that contradiction cannot be derived in small space (since $\mathbb{C}_t$ is satisfiable).

To lift this proof to PCR, however, we must maintain not just one but an exponential number of such good assignments, and in general we do not know how to do this. Nevertheless, some more thought reveals that the only important aspect of our assignments are the resulting vertex parities. And if the edge set is partitioned into cycles, we can always shift edge charges along the cycles so that for all the exponentially many assignments, these parities are all the same (meaning that we only have to maintain one good assignment after all).

Some graphs, such as rectangular grids, can be partitioned into cycles of size $\mathrm{O}(1)$, yielding tight bounds on space. A bit more surprisingly, random d-regular graphs for $d \geq 4$ turn out to (sort of) admit partitions into cycles of size $\mathrm{O}(\sqrt{n})$, which yields the following theorem.

Theorem 11. *Let G be a random d-regular graph on n vertices, where $d \geq 4$. Then over any field it holds almost surely that $Sp_{\mathcal{PCR}}(Ts(G, \chi) \vdash \bot) = \Omega\big(\sqrt{n}\big)$.*

Proof (sketch). As long as we are interested in properties holding asymptotically almost surely, we can replace random 4-regular graphs with unions of two random Hamiltonian cycles [22]. We show that a graph distributed according to the latter model almost surely decomposes into cycles of length $\mathrm{O}(\sqrt{n})$, along with εn additional edges (which are easily taken care of separately). Since random

graphs are also excellent expanders, we can apply Theorem 10. The argument easily extends to random d-regular graphs for any $d \geq 4$.

We believe that the true space bound should actually be $\Theta(n)$, just as for resolution, but such a result seems beyond the reach of our current techniques. Also, note that to make Theorem 10 go through we need graph expansion *plus* partitions into small cycles. It seems plausible that expansion alone should imply PCR space lower bounds, as for resolution, but again we cannot prove this.

Limitations of the PCR Space Lower Bound Technique. The framework in [13] can also be used to rederive all PCR space lower bounds shown previously in [1, 18], and in this sense [13] sums up what we know about PCR space lower bounds. There are also intriguing similarities between [13] and [3] (as partly hinted in the proof sketch for Theorem 7), which raises the question whether extendible families could perhaps be a step towards characterizing degree and showing that degree lower-bounds space in PC/PCR.

Even more intriguingly, however, there are CNF formulas for which it seems reasonable to expect that PCR space lower bounds should hold, but where extendible families seem very hard to construct. Such formulas include ordering principle formulas, functional pigeonhole principle (FPHP) formulas, and random 3-CNF formulas. In fact, no PCR space lower bounds are known for *any* 3-CNF formula—it is consistent with current knowledge that all 3-CNF formulas could have constant space complexity in PCR (!), though this seemingly absurd possibility can be ruled out for PC [18].

We show that the problems in applying [13] to the functional version of the pigeonhole principle are inherent, in that these techniques provably cannot establish *any* nontrivial space lower bound.

Theorem 12. *There is no r-extendible family for $FPHP_n^{n+1}$ for $r > 1$.*

Since by [24] these formulas require PC refutation degree $\Omega(n)$, one way of interpreting Theorem 12 is that the concept of r-extendible families is very far from providing the hoped-for characterization of degree.

One step towards proving PCR space lower bounds could be to obtain a weaker PC space lower bound—as noted above in the discussion of 3-CNF formulas, this can sometimes be easier. For $FPHP_n^{n+1}$, however, and for a slightly more general class of formulas described in the full-length version of this paper, it turns out that such PC space lower bounds would immediately imply also PCR space lower bounds.

Theorem 13. $Sp_{\mathcal{PCR}}(FPHP_n^{n+1} \vdash \bot) = \Theta(Sp_{\mathcal{PC}}(FPHP_n^{n+1} \vdash \bot))$.

4 Concluding Remarks

In this paper, following up on recent work in [6, 13, 18, 20], we report further progress on understanding space complexity in polynomial calculus and how the

space measure is related to size and degree. Specifically, we separate size and degree from space, and provide some circumstantial evidence for the conjecture that degree might be a lower bound on space in PC/PCR. We also prove space lower bounds for a large class of Tseitin formulas, a well-studied formula family for which nothing was previously known regarding PCR space.

We believe that our lower bounds for Tseitin formulas over random graphs are *not* optimal, however. And for the functional pigeonhole principle, we show that the technical tools developed in [13] cannot prove any non-constant PCR space lower bounds. Although we have not been able to prove this, we believe that similar impossibility results should hold also for ordering principle formulas and for the canonical 3-CNF version of the pigeonhole principle. Since all of these formulas require large degree in PCR and large space in resolution, it is natural to suspect that they should be hard for PCR space as well. The fact that arguments along the lines of [13] do not seem to be able to establish this suggests that we are still far from a combinatorial characterization of degree analogous to the characterization of resolution width in [3]. It thus remains a major open problem to understand the relation between degree and space in PC/PCR, and in particular whether degree (or even width) is a lower bound on space or not.

Also, our separations of size and degree on the one hand and space on the other depend on the characteristic of the underlying field. It would be satisfying to find formulas that provide such separations regardless of characteristic. Natural candidates are ordering principle formulas or onto function pigeon principle formulas, or, for potentially even stronger separations, pebbling formulas.

Acknowledgements. The authors wish to thank Ilario Bonacina and Nicola Galesi for numerous and very useful discussions.

The research of the first author has received funding from the European Union's Seventh Framework Programme (FP7/2007–2013) under grant agreement no. 238381. Part of the work of the first author was performed while visiting KTH Royal Institute of Technology. The other authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611. The fourth author was also supported by Swedish Research Council grants 621-2010-4797 and 621-2012-5645.

References

[1] Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Space complexity in propositional calculus. SIAM J. Comput. 31(4), 1184–1211 (2002)

[2] Alekhnovich, M., Razborov, A.A.: Lower bounds for polynomial calculus: Non-binomial case. Proc. Inst. Math. 242, 18–35 (2003)

[3] Atserias, A., Dalmau, V.: A combinatorial characterization of resolution width. J. Comput. Syst. Sci. 74(3), 323–334 (2008)

[4] Bayardo Jr., R.J., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Proc. 14th National Conference on Artificial Intelligence (AAAI 1997), pp. 203–208 (1997)

[5] Beame, P., Beck, C., Impagliazzo, R.: Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. In: Proc. 44th Symposium on Theory of Computing (STOC 2012), pp. 213–232 (2012)
[6] Beck, C., Nordström, J., Tang, B.: Some trade-off results for polynomial calculus. In: Proc. 45th Symposium on Theory of Computing, STOC 2013 (2013)
[7] Ben-Sasson, E.: Size space tradeoffs for resolution. SIAM J. Comput. 38(6), 2511–2525 (2009)
[8] Ben-Sasson, E., Galesi, N.: Space complexity of random formulae in resolution. Random Struct. Algorithms 23(1), 92–109 (2003)
[9] Ben-Sasson, E., Nordström, J.: Short proofs may be spacious: An optimal separation of space and length in resolution. In: Proc. 49th Symposium on Foundations of Computer Science (FOCS 2008), pp. 709–718 (2008)
[10] Ben-Sasson, E., Nordström, J.: Understanding space in proof complexity: Separations and trade-offs via substitutions. In: Proc. 2nd Symposium on Innovations in Computer Science (ICS 2011), pp. 401–416 (2011)
[11] Ben-Sasson, E., Wigderson, A.: Short proofs are narrow—resolution made simple. J. ACM 48(2), 149–169 (2001)
[12] Blake, A.: Canonical Expressions in Boolean Algebra. PhD thesis, University of Chicago (1937)
[13] Bonacina, I., Galesi, N.: Pseudo-partitions, transversality and locality: A combinatorial characterization for the space measure in algebraic proof systems. In: Proc. 4th Conference on Innovations in Theoretical Computer Science (ITCS 2013), pp. 455–472 (2013)
[14] Chvátal, V., Szemerédi, E.: Many hard examples for resolution. J. ACM 35(4), 759–768 (1988)
[15] Clegg, M., Edmonds, J., Impagliazzo, R.: Using the Groebner basis algorithm to find proofs of unsatisfiability. In: Proc. 28th Symposium on Theory of Computing (STOC 1996), pp. 174–183 (1996)
[16] Cook, S.A., Reckhow, R.: The relative efficiency of propositional proof systems. J. Symb. Log. 44(1), 36–50 (1979)
[17] Esteban, J.L., Torán, J.: Space bounds for resolution. Inf. Comput. 171(1), 84–97 (2001)
[18] Filmus, Y., Lauria, M., Nordström, J., Thapen, N., Ron-Zewi, N.: Space complexity in polynomial calculus. In: Proc. 27th Conference on Computational Complexity (CCC 2012), pp. 334–344 (2012)
[19] Haken, A.: The intractability of resolution. Theor. Comput. Sci. 39(2-3), 297–308 (1985)
[20] Huynh, T., Nordström, J.: On the virtue of succinct proofs: Amplifying communication complexity hardness to time-space trade-offs in proof complexity. In: Proc. 44th Symposium on Theory of Computing (STOC 2012), pp. 233–248 (2012)
[21] Impagliazzo, R., Pudlák, P., Sgall, J.: Lower bounds for the polynomial calculus and the Gröbner basis algorithm. Comput. Complex. 8(2), 127–144 (1999)
[22] Kim, J.H., Wormald, N.C.: Random matchings which induce Hamilton cycles, and hamiltonian decompositions of random regular graphs. J. Comb. Theory B 81, 20–44 (2001)
[23] Marques-Silva, J.P., Sakallah, K.A.: GRASP—a new search algorithm for satisfiability. In: Proc. International Conference on Computer-Aided Design (ICCAD 1996), pp. 220–227 (1996)
[24] Razborov, A.A.: Lower bounds for the polynomial calculus. Comput. Complex. 7(4), 291–324 (1998)
[25] Urquhart, A.: Hard examples for resolution. J. ACM 34(1), 209–219 (1987)

On the Power of Deterministic Mechanisms for Facility Location Games*

Dimitris Fotakis[1] and Christos Tzamos[2]

[1] School of Electrical and Computer Engineering,
National Technical University of Athens, 157 80 Athens, Greece
[2] Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institute of Technology, Cambridge, MA 02139
fotakis@cs.ntua.gr, tzamos@mit.edu

Abstract. We investigate the approximability of K-Facility Location by deterministic strategyproof mechanisms. Our main result is an elegant characterization of deterministic strategyproof mechanisms with a bounded approximation ratio for 2-Facility Location on the line. Specifically, we show that for instances with $n \geq 5$ agents, any such mechanism either admits a unique dictator, or always places the facilities at the two extremes. As a consequence, we obtain that the best approximation ratio achievable by deterministic strategyproof mechanisms for 2-Facility Location on the line is precisely $n - 2$. Employing a technical tool developed for the characterization, we show that for every $K \geq 3$, there do not exist any deterministic anonymous strategyproof mechanisms with a bounded approximation ratio for K-Facility Location on the line, even for simple instances with $K + 1$ agents. Moreover, building on the characterization for the line, we show that there do not exist any deterministic mechanisms with a bounded approximation ratio for 2-Facility Location in more general metric spaces, which is true even for simple instances with 3 agents located in a star.

1 Introduction

We study *K-Facility Location games*, where K facilities are placed in a continuous metric space based on the preferences of n strategic agents. Such problems are motivated by natural scenarios in Social Choice, where the government plans to build a fixed number of public facilities in an area (see e.g., [13]). Each *agent* reports her ideal location, and the government applies a *mechanism* mapping their preferences to K facility locations. The government seeks to minimize the *social cost*, namely the total distance of the agents' locations to the nearest facility. On the other hand, the agents seek to minimize their *connection cost*, namely the distance of their location to the nearest facility. In fact, an agent may even misreport her ideal location in an attempt of manipulating the mechanism. Therefore, the mechanism should be *strategyproof*, i.e., ensure that no

* This research was supported by the project Algorithmic Game Theory, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALES, investing in knowledge society through the European Social Fund. Full version at http://arxiv.org/abs/1207.0935

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 449–460, 2013.

agent can benefit from misreporting her location. Moreover, to compute a desirable outcome, the mechanism should achieve a good approximation to the optimal social cost.

Previous Work. In addition to strategyproofness, which is an essential property of any mechanism, Social Choice suggests a few efficiency-related properties, e.g., onto, non-dictatorship, and Pareto-efficiency, that accompany strategyproofness, and ensure that the mechanism's outcome is socially desirable. There are several characterization theorems which state that for a particular domain, the class of strategyproof mechanisms with some efficiency-related properties coincides with a rather restricted class of mechanisms (see e.g., [2]). A notable example of a problem admitting a rich class of strategyproof mechanisms is that of locating a single facility on the line, where the agents' preferences are single-peaked. Moulin [14] proved that a mechanism is strategyproof for 1-Facility Location on the line iff it is a generalized median voter scheme. Schummer and Vohra [17] extended this characterization to tree metrics. For non-tree metrics, they proved that any onto strategyproof mechanism must be a dictatorship. Recently, Dokow et al. [4] obtained similar characterizations for the class of onto strategyproof mechanisms for 1-Facility Location on the discrete line and on the discrete circle.

Adopting an algorithmic viewpoint, Procaccia and Tennenholtz [16] introduced the framework of *Approximate Mechanism Design without Money*. The idea is to consider game-theoretic versions of optimization problems, where a social objective function summarizes (or even strengthens) the efficiency-related properties. Any reasonable approximation to the optimal solution can be regarded as a socially desirable outcome, and we seek to determine the best approximation ratio achievable by strategyproof mechanisms. For example, the results of [14,17] imply that 1-Facility Location in tree metrics can be solved optimally by a strategyproof mechanism. On the other hand, the negative result of [17] implies that the best approximation ratio achievable by deterministic mechanisms for 1-Facility Location in general metrics is $n-1$.

Procaccia and Tennenholtz [16] considered location problems on the line, and obtained upper and lower bounds on the approximation ratio achievable by strategyproof mechanisms. For 2-Facility Location, they suggested the TWO-EXTREMES mechanism, that places the facilities at the leftmost and at the rightmost location, and achieves an approximation ratio of $n-2$. On the negative side, they proved a lower bound of $3/2$ on the approximation ratio of any deterministic mechanism. Lu et al. [12] strengthened the lower bound for deterministic mechanisms to 2 and established a lower bound of 1.045 for randomized mechanisms. Shortly afterwards, Lu et al. [11] significantly improved the lower bound for deterministic mechanisms to $(n-1)/2$. On the positive side, they proved that a natural randomized mechanism is strategyproof and achieves an approximation ratio of 4 for 2-Facility Location in general metrics. However, Lu et al. observed that this mechanism is not strategyproof for more than two facilities.

Motivation and Contribution. Facility Location games are among the central problems in the research agenda of Mechanism Design without Money, and have received considerable attention. Our work is motivated by the apparent difficulty of obtaining any strong(er) positive results on the approximability of K-Facility Location by deterministic mechanisms. In fact, among the main open problems of [11] were (i) to determine the best approximation ratio achievable by deterministic mechanisms for 2-Facility Location on the line, (ii) to investigate the existence of deterministic mechanisms with a

bounded approximation ratio for K-Facility Location with $K \geq 3$, and (iii) to investigate the existence of deterministic mechanisms with a bounded approximation ratio for 2-Facility Location in metric spaces other than the line and the circle. In this work, we resolve the first question, and obtain strong negative results for the second and the third.

Attacking these questions requires a complete understanding of the behavior of deterministic strategyproof mechanisms for K-Facility Location, similar to that offered by characterization theorems in Social Choice. Hence, we suggest an approach in the intersection of Social Choice and Mechanism Design without Money. Following the approach of the latter, we focus on *nice mechanisms*, namely deterministic strategyproof mechanisms with an approximation ratio bounded by a function of n and K. Following the approach of Social Choice, we embark on a complete characterization of nice mechanisms. Although for simplicity, we focus on the objective of social cost, the class of nice mechanisms is very general and essentially independent of the objective function. E.g., for any $p \geq 1$ or for $p = \infty$, a mechanism achieves a bounded approximation ratio for the objective of minimizing the L_p norm of the agents' connection cost iff it achieves a bounded approximation for the objective of social cost. Thus, to a very large extent, a characterization of nice mechanisms retains the generality of characterizations in Social Choice, since it captures all, but some socially intolerable, strategyproof mechanisms.

Focusing on nice mechanisms facilitates the characterization, since it excludes several socially intolerable mechanisms. Nevertheless, any characterization of nice mechanisms, even for two facilities, remains an intriguing task, because there is no apparent notion of monotonicity (as e.g., in [10]), and the combinatorial structure of the preferences is significantly more complicated than that for a single facility.

Our main result is an elegant characterization of nice mechanisms for 2-Facility Location on the line. We show that any nice mechanism for $n \geq 5$ agents either admits a unique dictator, or always places the facilities at the two extremes (Theorem 1). A corollary is that the best approximation ratio achievable by deterministic mechanisms for 2-Facility Location on the line is $n - 2$. Another rather surprising consequence is that TWO-EXTREMES is the only anonymous nice mechanism for this problem.

The proof of Theorem 1 proceeds by establishing the characterization at three different levels of generality: 3-agent, 3-location, and general instances. Along the way, we develop strong technical tools that fully describe the behavior of nice mechanisms. To exploit locality, we first focus on well-separated instances with 3 agents, where an isolated agent is served by one facility, and two nearby agents are served by the other facility. Interestingly, we identify two large classes of well-separated instances where any nice mechanism should keep allocating the latter facility to the same agent (Propositions 1 and 2). Building on this, we show that the location of the facility serving the nearby agents is determined by a generalized median voter scheme, as in [14], but with a threshold depending on the location and the identity of the isolated agent, and then extend this property to general instances with 3 agents (Fig. 1). The key step is to show that the threshold of each isolated agent can take only two extreme values: one corresponding to the existence of a partial dictator, and one corresponding to allocating the facility to the furthest agent. Then, considering all possible cases for the agents' thresholds, we show that any nice mechanism for 3 agents either places the facilities at the two extremes, or admits a partial dictator (Theorem 2).

Employing partial group strategyproofness [11, Sec. 3], and a new technical tool for moving agents between different coalitions without affecting the outcome (Lemma 1), and show that any nice mechanism applied to 3-location instances with $n \geq 5$ agents either admits a (full) dictator, or places the facilities at the two extremes (Theorem 3). Rather surprisingly, this implies that nice mechanisms for 3 agents are somewhat less restricted than nice mechanisms for $n \geq 5$ agents. Finally, in Section 3.3, we employ induction on the number of different locations, and conclude the proof of Theorem 1.

In addition to extending the ideas of [14] to 2-Facility Location games and to exploiting the notions of image sets and partial group strategyproofness, we introduce new ideas and technical tools, which provide new insights into the behavior of nice mechanisms for K-Facility Location games and may be of independent interest. Among them, we may single out the notion of well-separated instances and the idea of reducing K-Facility Location in well-separated instances to a single facility game between the two nearby agents, the ideas used to extend the facility allocation from well-separated to general instances, and the technical tool of moving agents between different coalitions.

For $K \geq 3$ facilities, we show that there do not exist any anonymous nice mechanisms even for well-separated instances with $K+1$ agents on the line (Theorem 4). For 2-Facility Location in metric spaces more general than the line and the circle, we show that there do not exist any nice mechanisms, which holds even for simple instances with 3 agents in a star (Theorem 5). Both results are based on the technical tools for well-separated instances developed in the proof of Theorem 1, thus indicating the generality and the potential applicability of our techniques.

Other Related Work. In Social Choice, the work on multiple facility location games mostly focuses on Pareto-efficient deterministic strategyproof mechanisms with some additional properties (see e.g., [3,9,13]). However, known results do not have any immediate implications for the approximability of the social cost.

Single Facility. Alon et al. [1] almost completely characterized the approximation ratios achievable by randomized and deterministic mechanisms for 1-Facility Location in general metrics and rings. Next, [6] proved that for the L_2 norm of the agents' distances, the best approximation ratio is 1.5 for randomized and 2 for deterministic mechanisms.

Multiple Facilities. Recently, we presented, in [7], the first randomized strategyproof mechanism with a bounded approximation ratio for K-Facility Location on the line. The mechanism works by equalizing the expected connection cost of all agents, and is n-approximate for the social cost and 2-approximate for the maximum cost of the agents. Moreover, in [7], we presented a randomized strategyproof mechanism that applies to instances with K facilities and $K+1$ agents on the line, and is 2-approximate for the social cost, thus improving on the approximation ratio of $(K+1)/2$ obtained by Escoffier et al. [5] for this special case. Notably, Theorem 4 shows that instances with only $K+1$ agents are hard for deterministic mechanisms.

Imposing Mechanisms. Nissim et al. [15] introduced the notion of *imposing mechanisms*, where the mechanism can restrict how the agents exploit its outcome. They obtained a randomized imposing mechanism for K-Facility Location on the line that approximates the average optimal social cost within an additive term of $o(n)$. Subsequently, we proved, in [8], that the imposing version of the randomized mechanism of [11] is $4K$-approximate and strategyproof for K-Facility Location in general metrics.

2 Notation, Definitions, and Preliminaries

Except for Section 5, we consider K-Facility Location on the real line. So, in this section, we introduce the notation and the basic notions only for instances on the real line.

Notation. For a tuple $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, $\min \boldsymbol{x}$, $\max \boldsymbol{x}$, and $\operatorname{med} \boldsymbol{x}$ denote the smallest, the largest, and the $\lceil n/2 \rceil$-smallest coordinate of $\boldsymbol{x}$, respectively. We let $\boldsymbol{x}_{-i}$ be the tuple $\boldsymbol{x}$ without x_i. We write $(\boldsymbol{x}_{-i}, a)$ to denote the tuple $\boldsymbol{x}$ with a in place of x_i, $(\boldsymbol{x}_{-\{i,j\}}, a, b)$ to denote $\boldsymbol{x}$ with a in place of x_i and b in place of x_j, and so on.

Instances. Let $N = \{1, \ldots, n\}$ be a set of $n \geq 3$ agents. Each agent $i \in N$ has a location $x_i \in \mathbb{R}$, which is i's private information. We usually refer to a locations profile $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ as an *instance*. For an instance $\boldsymbol{x}$, we say that the agents are arranged on the line according to a permutation π if π arranges them in increasing order of their locations in $\boldsymbol{x}$, i.e., $x_{\pi(1)} \leq x_{\pi(2)} \leq \cdots \leq x_{\pi(n)}$. In the following, we consider *3-agent instances*, where $n = 3$, and *3-location instances*, where there are three different locations x_1, x_2, x_3, and a partition of N into three coalitions N_1, N_2, N_3 such that all agents in coalition N_i occupy location x_i, $i \in \{1, 2, 3\}$. We usually denote such an instance as $(x_1{:}N_1, x_2{:}N_2, x_3{:}N_3)$. For a set N of agents, we let $\mathcal{I}(N)$ denote the set of all instances, and let $\mathcal{I}_3(N)$ denote the set of all 3-location instances.

Mechanisms. A mechanism F for K-Facility Location maps an instance $\boldsymbol{x}$ to a K-tuple $(y_1, \ldots, y_K) \in \mathbb{R}^K$, $y_1 \leq \cdots \leq y_K$, of facility locations. We let $F(\boldsymbol{x})$ denote the outcome of F for instance $\boldsymbol{x}$, and let $F_\ell(\boldsymbol{x})$ denote y_ℓ, i.e., the ℓ-th smallest coordinate in $F(\boldsymbol{x})$. For 2-Facility Location, $F_1(\boldsymbol{x})$ denotes the leftmost and $F_2(\boldsymbol{x})$ denotes the rightmost facility. We write $y \in F(\boldsymbol{x})$ to denote that $F(\boldsymbol{x})$ has a facility at y. A mechanism F is *anonymous* if for all $\boldsymbol{x}$ and all agent permutations π, $F(\boldsymbol{x}) = F(x_{\pi(1)}, \ldots, x_{\pi(n)})$. An agent i is a *dictator* of F if for all instances $\boldsymbol{x}$, $x_i \in F(\boldsymbol{x})$.

Social Cost. Given a mechanism F for K-Facility Location and an instance $\boldsymbol{x}$, the cost of agent i is $\operatorname{cost}[x_i, F(\boldsymbol{x})] = \min_{1 \leq \ell \leq K} \{|x_i - F_\ell(\boldsymbol{x})|\}$. The *social cost* of F for $\boldsymbol{x}$ is $\operatorname{cost}[F(\boldsymbol{x})] = \sum_{i=1}^{n} \operatorname{cost}[x_i, F(\boldsymbol{x})]$. The optimal cost for an instance $\boldsymbol{x}$ is $\min \sum_{i=1}^{n} \operatorname{cost}[x_i, (y_1, \ldots, y_K)]$, where the minimum is over all tuples $(y_1, \ldots, y_K)$.

A mechanism F has an approximation ratio of $\rho \geq 1$, if for any instance $\boldsymbol{x}$, the cost of $F(\boldsymbol{x})$ is at most ρ times the optimal cost for $\boldsymbol{x}$. We say that the approximation ratio ρ of F is *bounded* if ρ depends only on n and K. Since for any $p \geq 1$ (or for $p = \infty$), and for any non-negative n-tuple $\boldsymbol{c}$, $\|c\|_p \leq \sum_{i=1}^{n} c_i \leq n^{1-1/p} \|c\|_p$, a mechanism has a bounded approximation ratio for the L_p norm of the agent costs if and only if it has a bounded approximation ratio for the social cost.

Strategyproofness. A mechanism F is *strategyproof* if no agent can benefit from misreporting her location. Formally, for all instances $\boldsymbol{x}$, any agent i, and all locations y, $\operatorname{cost}[x_i, F(\boldsymbol{x})] \leq \operatorname{cost}[x_i, F(\boldsymbol{x}_{-i}, y)]$. A mechanism F is *group strategyproof* if for any coalition of agents misreporting their locations, at least one of them does not benefit. Formally, for all instances $\boldsymbol{x}$, any coalition S, and all subinstances $\boldsymbol{y}_S$, there exists some agent $i \in S$ such that $\operatorname{cost}[x_i, F(\boldsymbol{x})] \leq \operatorname{cost}[x_i, F(\boldsymbol{x}_{-S}, \boldsymbol{y}_S)]$. A mechanism F is *partially group strategyproof* if for any coalition of agents that occupy the same location, none of them can benefit if they misreport their location simultaneously. Formally, for all instances $\boldsymbol{x}$, any coalition of agents S, all occupying the same location x in $\boldsymbol{x}$, and all subinstances $\boldsymbol{y}_S$, $\operatorname{cost}[x, F(\boldsymbol{x})] \leq \operatorname{cost}[x, F(\boldsymbol{x}_{-S}, \boldsymbol{y}_S)]$.

Any group strategyproof mechanism is partially group strategyproof, and any partially group strategyproof mechanism is strategyproof. [11, Lemma 2.1] shows that any strategyproof mechanism for K-Facility Location is partially group strategyproof.

Image Sets. Given a mechanism F, the *image* (or option) *set* $I_i(\boldsymbol{x}_{-i})$ of an agent i with respect to an instance $\boldsymbol{x}_{-i}$ is the set of facility locations the agent i can obtain by varying her reported location, i.e., $I_i(\boldsymbol{x}_{-i}) = \{a \in \mathbb{R} : \exists y \in \mathbb{R} \text{ such that } a \in F(\boldsymbol{x}_{-i}, y)\}$. If F is strategyproof, any image set $I_i(\boldsymbol{x}_{-i})$ is a collection of closed intervals (see e.g., [18, p. 249]), and F places a facility at the location in $I_i(\boldsymbol{x}_{-i})$ closest to the location of agent i. Formally, for any agent i, all instances $\boldsymbol{x}$, and all locations y, $\mathrm{cost}[y, F(\boldsymbol{x}_{-i}, y)] = \inf_{a \in I_i(\boldsymbol{x}_{-i})}\{|y - a|\}$. In [11, Section 3.1], it is shown that using partial group strategyproofness, we can extend the notion of image sets and the properties above to coalitions of agents that occupy the same location in an instance $\boldsymbol{x}$.

Any (open) interval in the complement of an image set $I \equiv I_i(\boldsymbol{x}_{-i})$ is called a *hole* of I. Given a location $y \notin I$, we let $l_y = \sup_{a \in I}\{a < y\}$ and $r_y = \inf_{a \in I}\{a > y\}$ be the locations in I nearest to y on the left and on the right, respectively. Since I is a collection of closed intervals, l_y and r_y are well defined and satisfy $l_y < y < r_y$. For convenience, given a $y \notin I$, we refer to the interval (l_y, r_y) as a y-hole in I.

Nice Mechanisms. We refer to any mechanism F that is deterministic, strategyproof, and has a bounded approximation ratio as a *nice mechanism*. We usually refer to F without mentioning its approximation ratio, with the understanding that given F and the set N of agents, we can determine an upper bound ρ on F's approximation ratio.

Due to the bounded approximation ratio, a nice mechanism F for K-Facility Location is *unanimous*, i.e., for all $\boldsymbol{x}$ where the agents occupy only K locations $x_1, \ldots, x_K$, $F(\boldsymbol{x}) = (x_1, \ldots, x_K)$. Similarly, any hole in an image set $I_i(\boldsymbol{x}_{-i})$ of F is a bounded interval. Otherwise, we could move agent i sufficiently far away from the other agents, and obtain an instance for which F would have approximation ratio larger than ρ.

Well-Separated Instances. Given a nice mechanism F for K-Facility Location with approximation ratio ρ, a $(K+1)$-agent instance $\boldsymbol{x}$ is called $(i_1|\cdots|i_{K-1}|i_K, i_{K+1})$-*well-separated* if $x_{i_1} < \cdots < x_{i_{K+1}}$ and $\rho(x_{i_{K+1}} - x_{i_K}) < \min_{2 \le \ell \le K}\{x_{i_\ell} - x_{i_{\ell-1}}\}$. Namely, in a well-separated instance, there are two nearby agents whose distance to each other is less than $1/\rho$ times the optimal social cost. Therefore any mechanism with an approximation ratio of ρ serves the two nearby agents by the same facility, and serves the remaining "isolated" agents by a different facility each.

We can show that if there is an $(i_1|\cdots|i_{K-1}|i_K, i_{K+1})$-well-separated instance $\boldsymbol{x}$ with $F_K(\boldsymbol{x}) = x_{i_K}$ (resp. $F_K(\boldsymbol{x}) = x_{i_{K+1}}$), then as long as we "push" the locations of agents i_K and i_{K+1} to the right (resp. left), while keeping the instance well-separated, the rightmost facility of F stays with the location of i_K (resp. i_{K+1}). We note that the equivalent property holds if the two nearby agents are located elsewhere in the instance.

Proposition 1. *Let F be a nice mechanism and $\boldsymbol{x}$ be a $(i_1|\cdots|i_{K-1}|i_K, i_{K+1})$-well-separated instance with $F_K(\boldsymbol{x}) = x_{i_K}$. Then for every $(i_1|\cdots|i_{K-1}|i_K, i_{K+1})$-well-separated instance $\boldsymbol{x}' = (\boldsymbol{x}_{-\{i_K, i_{K+1}\}}, x'_{i_K}, x'_{i_{K+1}})$ with $x_{i_K} \le x'_{i_K}$, $F_K(\boldsymbol{x}') = x'_{i_K}$.*

Proposition 2. *Let F be a nice mechanism and $\boldsymbol{x}$ be a $(i_1|\cdots|i_{K-1}|i_K, i_{K+1})$-well-separated instance with $F_K(\boldsymbol{x}) = x_{i_{K+1}}$. For all $(i_1|\cdots|i_{K-1}|i_K, i_{K+1})$-well-separated instances $\boldsymbol{x}' = (\boldsymbol{x}_{-\{i_K, i_{K+1}\}}, x'_{i_K}, x'_{i_{K+1}})$ with $x'_{i_{K+1}} \le x_{i_{K+1}}$, $F_K(\boldsymbol{x}') = x'_{i_{K+1}}$.*

3 Strategyproof Mechanisms for 2-Facility Location

We start with discussing the key proof steps and the consequences of our main result:

Theorem 1. *Let F be a nice mechanism for 2-Facility Location with $n \geq 5$ agents. Then, either $F(\boldsymbol{x}) = (\min \boldsymbol{x}, \max \boldsymbol{x})$ for all instances $\boldsymbol{x}$, or there exists a unique dictator j such that for all $\boldsymbol{x}$, $x_j \in F(\boldsymbol{x})$.*

We are aware of only two nice mechanisms for 2-Facility Location with $n \geq 4$ agents, one for each case of Theorem 1. The DICTATORIAL mechanism is an adaptation of the mechanism of [11] for the circle. It chooses a dictator j, and for each instance $\boldsymbol{x}$, allocates a facility to x_j. Then, it considers the distance of the dictator to the leftmost and to the rightmost location, $d_l = |\min \boldsymbol{x} - x_j|$ and $d_r = |\max \boldsymbol{x} - x_j|$, respectively. The second facility is placed at $x_j - \max\{d_l, 2d_r\}$, if $d_l > d_r$, and to $x_j + \max\{d_r, 2d_l\}$, otherwise. As in [11, Section 5], it can be shown that DICTATORIAL is strategyproof and $(n-1)$-approximate for the line. The TWO-EXTREMES mechanism places the facilities at $(\min \boldsymbol{x}, \max \boldsymbol{x})$, for all instances $\boldsymbol{x}$, and is group strategyproof, anonymous, and $(n-2)$-approximate, as shown in [16]. By Theorem 1, TWO-EXTREMES is the only anonymous nice mechanism for 2-Facility Location with $n \geq 5$ agents and its approximation ratio is best possible. Using Theorem 1, we can show that:

Corollary 1. *Any deterministic strategyproof mechanism for 2-Facility Location with $n \geq 5$ agents has an approximation ratio of at least $n - 2$.*

The crux, and the most technically involved part, of the proof of Theorem 1 is to establish a characterization of nice mechanisms dealing with just 3 agents. In particular, we show that any nice mechanism for 2-Facility Location with 3 agents either places the facilities at the two extremes, or admits a *partial dictator*, namely an agent allocated a facility either for all agent permutations or for all agent permutations but one.

Theorem 2. *Let F be any nice mechanism for 2-Facility Location with $n = 3$ agents. Then, there exist at most two permutations π_1, π_2, with $\pi_1(2) = \pi_2(2)$, such that for all instances $\boldsymbol{x}$ where the agents are arranged on the line according to π_1 or π_2, $\operatorname{med} \boldsymbol{x} \in F(\boldsymbol{x})$. For any other permutation π and instance $\boldsymbol{x}$, where the agents are arranged on the line according to π, $F(\boldsymbol{x}) = (\min \boldsymbol{x}, \max \boldsymbol{x})$.*

We highlight that the notion of a partial dictator is essential. The COMBINED mechanism for 3 agents chooses a permutation (i, j, k) of the agents, and for each instance $\boldsymbol{x}$, places the facilities using TWO-EXTREMES, if $x_i < x_k$, and using DICTATORIAL with dictator j, otherwise. Thus, COMBINED admits a partial dictator, is strategyproof, and achieves an approximation ratio of 2.

Using the notion of partial group strategyproofness, we extend Theorem 2 to 3-location instances. The next step is to show that when applied to 3-location instances with $n \geq 5$ agents, nice mechanisms do not have the option of a partial dictator. More formally, in Section 3.2, we sketch the proof of the following:

Theorem 3. *Let N be a set of $n \geq 5$ agents, and let F be any nice mechanism for 2-Facility Location applied to instances in $\mathcal{I}_3(N)$. Then, either there exists a unique dictator $j \in N$ such that for all instances $\boldsymbol{x} \in \mathcal{I}_3(N)$, $x_j \in F(\boldsymbol{x})$, or for all instances $\boldsymbol{x} \in \mathcal{I}_3(N)$, $F(\boldsymbol{x}) = (\min \boldsymbol{x}, \max \boldsymbol{x})$.*

Finally, in Section 3.3, we employ induction on the number of agents, and extend Theorem 3 to general instances with $n \geq 5$ agents, thus concluding the proof of Theorem 1. In the next three subsections, we present the main ideas of the proof of Theorem 1. We usually omit any quantification of F, with the understanding that F denotes a nice mechanism for 2-Facility Location applied to the relevant class of instances.

3.1 Strategyproof Allocation of 2 Facilities to 3 Agents

In this section, we sketch the proof of Theorem 2. In the following, we let $\uparrow$ and $\downarrow$ denote the largest and the smallest element of the affinely extended real line. Hence, $\uparrow$ is greater than any real number and $\downarrow$ is less than any real number. We use the indices i, j, k to implicitly define a permutation of the agents. We mostly use the convention that i is the leftmost agent, j is the middle agent, and k is the rightmost agent.

We recall that given a nice mechanism F with approximation ratio ρ for 3 agents, a 3-agent instance $\boldsymbol{x}$ is $(i|j,k)$-*well-separated* if $x_i < x_j < x_k$ and $\rho(x_k - x_j) < x_j - x_i$. Similarly, $\boldsymbol{x}$ is $(i,j|k)$-*well-separated* if $x_i < x_j < x_k$ and $\rho(x_j - x_i) < x_k - x_j$. A 3-agent instance $\boldsymbol{x}$ is i-*left-well-separated* if $\boldsymbol{x}$ is either $(i|j,k)$ or $(i|k,j)$-well-separated, and is k-*right-well-separated* if it is either $(i,j|k)$ or $(j,i|k)$-well-separated. Moreover, a 3-agent instance $\boldsymbol{x}$ is i-*well-separated* if $\boldsymbol{x}$ is either i-left or i-right-well-separated.

At a high-level, the proof of Theorem 2 proceeds by gradually restricting the possible outcomes of a nice mechanism. There are three main steps, each establishing the desired conclusion for a different level of generality. As a first step, we consider the behavior of nice mechanisms for well-separated instances. Since the mechanism has a bounded approximation ratio, for any i-well-separated instance, one facility serves the isolated agent i, and the other facility is placed between the locations of the two nearby agents j, k. Thus, building on the characterization of [14], we show that for any i-well-separated instance, the facility serving agents j and k is allocated by a generalized median voter scheme (see e.g., [18, Definition 10.3]) whose characteristic threshold may depend on the identity i and the location a of the isolated agent. More specifically, we show that any agent-location pair (i,a) specifies a unique threshold $p \in [a, +\infty) \cup \{\uparrow\}$ and a preferred agent $j \neq i$, that fully determine the location of the rightmost facility for all i-left-well-separated instances $\boldsymbol{x}$ with $x_i = a$ (see Fig. 1.a; by symmetry, the same holds for i-right-well-separated instances and the leftmost facility, though possibly with different values of p and j). Moreover, the allocation of the rightmost facility becomes simple for the two extreme values of the threshold p: if $p = a$, the preferred agent j serves as a dictator imposed by i for all i-left-well-separated instances, while if $p = \uparrow$, the rightmost facility is placed at the rightmost location.

The key step is to show that the threshold p of any agent-location pair (i,a) can be either a or $\uparrow$ (resp. either a or $\downarrow$ if i is the rightmost agent). To this end, we first extend the allocation above to general instances with i as the leftmost (resp. rightmost) agent (see Fig. 1.b). In a nutshell, we show that for most such instances, the threshold p and the preferred agent j, that determine the location of a facility on the right (resp. left) part of the instance, are the same as those for i-left (resp. i-right) well-separated instances. Thus, if the preferred agent j is located on the right (resp. left) of the threshold p, she essentially serves as a partial dictator, imposed by the leftmost (resp. rightmost) agent, for the corresponding permutation of agents.

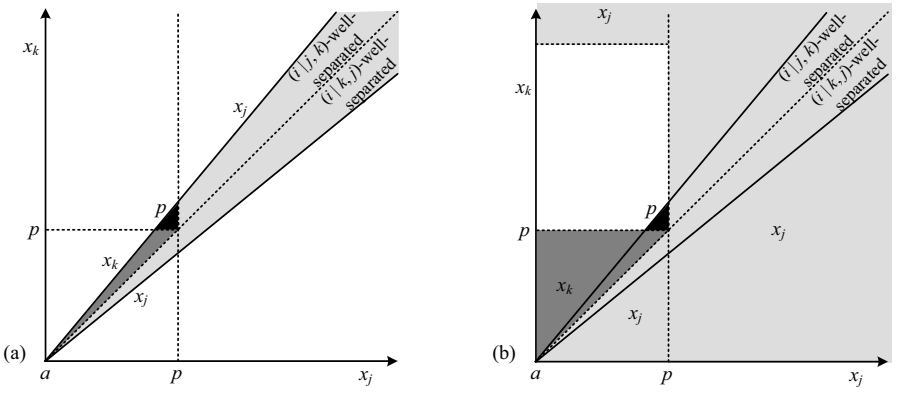


Fig. 1. (a) The location of $F_2(\boldsymbol{x})$ for all i-left-well-separated instances $\boldsymbol{x}$ with $x_i = a$. We let j be the preferred agent and p be the threshold of (i, a). The location of agent j (resp. k) is on the x-axis (resp. y-axis). The area around the line $x_j = x_k$ includes all i-left-well-separated instances. For instances in the dark grey area (where $x_j \leq x_k \leq p$), $F_2(\boldsymbol{x}) = x_k$, for instances in the black triangle (where $x_j \leq p \leq x_k$), $F_2(\boldsymbol{x}) = p$, and for instances in the light grey area (where either $x_j \geq p$ or $x_k \leq x_j \leq p$), $F_2(\boldsymbol{x}) = x_j$. (b) We consider instances $\boldsymbol{x}$ with $x_i = a < x_j, x_k$, which are not necessarily well-separated. The plot depicts for which instances a facility (not necessarily the rightmost one) is placed at either x_j, or x_k, or p.

As consequence, we obtain that the thresholds of the two allocation rules (one imposed by the leftmost agent and one imposed by the rightmost agent) always fall in the two extremes: either a or $\uparrow$ for the leftmost agent (resp. either a or $\downarrow$ for the rightmost agent). Otherwise, there exist instances with two different partial dictators, leading to an unbounded approximation ratio. Intuitively, the black triangle in Fig. 1.b does not exist, and either $p = a$, which corresponds to the existence of a partial dictator, or $p = \uparrow$ (resp. $p = \downarrow$), which corresponds to placing the facilities at the two extremes.

Building on this, we show that the thresholds of the two allocation rules, one for the leftmost and one for the rightmost agent, can only depend on their identity, and not on their location. Moreover, if an agent i imposes a partial dictator, the third agent agrees with i not only on the existence of a partial dictator, but also on the dictator's identity, and the partial dictator is unique. Therefore, every nice mechanism is essentially characterized by whether there are two agents that agree on imposing the third agent as a partial dictator or not. Examining all possible cases, we conclude that every nice mechanism F either always places the facilities at the two extremes, or admits a partial dictator j. In the latter case, the partial dictator j is identified by any instance $\boldsymbol{x}$, with $x_i < x_j < x_k$, such that $x_j \in F(\boldsymbol{x})$.

3.2 Strategyproof Allocation of 2 Facilities to 3 Locations

The proof of Theorem 3 is based on an extension of Theorem 2 to 3-location instances. To justify the extension, we can restate the whole proof of Theorem 2 with 3 coalitions of agents, instead of 3 agents, and use that any strategyproof mechanism is also partially group strategyproof [11, Lemma 2.1].

A central notion in the proof of Theorem 3 is that of a *dictator coalition*. A non-empty $C \subset N$, $|C| \leq |N| - 2$, is a dictator coalition for 3-location instances, if for all partitions N_1, N_2 of $N \setminus C$ and all instances $\boldsymbol{x} = (x_1{:}N_1, x{:}C, x_2{:}N_2) \in \mathcal{I}_3(N)$, $x \in F(\boldsymbol{x})$. The first main step is to show the following property of dictator coalitions:

Lemma 1. *Let N be a set of $n \geq 4$ agents. If there exists a 3-location instance $\boldsymbol{x} = (x_1{:}N_1, x_2{:}N_2, x_3{:}N_3)$, with $x_1 < x_2 < x_3$ and $|N_2| \leq n - 3$, such that $x_2 \in F(\boldsymbol{x})$, then any coalition $N_2' \supseteq N_2$ is a dictator coalition for 3-location instances.*

To conclude the proof of Theorem 3, we show that for $n \geq 5$ agents, if there exists a 3-location instance where the middle coalition is allocated a facility, this coalition includes a unique agent serving as a dictator for all instances.

3.3 Strategyproof Allocation of 2 Facilities to n Agents

The final step is to extend Theorem 3 to general instances with $n \geq 5$ agents. The proof considers two different cases, depending on how the mechanism F behaves for 3-location instances, and proceeds by induction on the number of different locations.

We first consider the case where F admits a dictator j for 3-location instances, and show that the agent j is a dictator for all $\boldsymbol{x} \in \mathcal{I}(N)$. For sake of contradiction, we assume an instance $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathcal{I}(N)$ for which $x_j \notin F(\boldsymbol{x})$. Wlog., we let $k \neq j$ be the rightmost agent of $\boldsymbol{x}$ (if j is the rightmost agent, the argument is symmetric). Since $x_j \notin F(\boldsymbol{x})$, there is a x_j-hole (l, r) in the imageset $I_j(\boldsymbol{x}_{-j})$. For a small $\varepsilon \in (0, (r-l)/2)$, we consider the instance $\boldsymbol{x}_1 = (\boldsymbol{x}_{-j}, l + \varepsilon)$, where j moves from x_j to $l + \varepsilon$. By strategyproofness, and since $l + \varepsilon$ is in the left half of the hole (l, r), $l \in F(\boldsymbol{x}_1)$. Then, we iteratively move all agents $i \in N \setminus \{j, k\}$ from x_i to l. By strategyproofness, if F has a facility at l before i moves from x_i to l, F keeps its facility at l after i's move. Otherwise, agent i with location l could manipulate F by reporting x_i. Thus, we obtain a 3-location instance $\boldsymbol{x}' = (l{:}N \setminus \{j, k\}, l + \varepsilon{:}\{j\}, x_k{:}\{k\})$ with $l < l + \varepsilon < x_k$, such that $l \in F(\boldsymbol{x}')$. Moreover, since j is a dictator for 3-location instances, $l + \varepsilon \in F(\boldsymbol{x}')$, and thus $F(\boldsymbol{x}') = (l, l + \varepsilon)$. For ε sufficiently smaller than $x_k - l$, this contradicts the bounded approximation ration of F.

The case where F does not admit a dictator for 3-location instances is similar.

4 Inexistence of Anonymous Nice Mechanisms for $K \geq 3$

In this section, we obtain an impossibility result for anomynous nice K-Facility Location mechanisms, for all $K \geq 3$.

Theorem 4. *For every $K \geq 3$, any deterministic anonymous strategyproof mechanism for K-Facility Location with $n \geq K + 1$ agents on the real line has an unbounded approximation ratio.*

Proof sketch. We only consider here the case where $K = 3$ and $n = 4$. For sake of contradiction, we let F be an anonymous nice mechanism for 3-Facility Location, and let ρ be its approximation ratio for instances with 4 agents. Next, we construct a family of instances for which the approximation ratio of F is greater than ρ.

Since F is anonymous, we assume that for any instance $\boldsymbol{x}$, $x_1 < x_2 < x_3 < x_4$. For some large $\lambda > \rho$, we consider the instance $\boldsymbol{x} = (0, \lambda, 3\lambda^2 + \lambda, 3\lambda^2 + \lambda + 1)$, which is $(1|2|3, 4)$-well-separated. We first show that $F_3(\boldsymbol{x}) \in [x_3, x_4]$. Wlog., we assume that $F_3(\boldsymbol{x}) \in \{x_3, x_4\}$. Otherwise, if $F_3(\boldsymbol{x}) = a$ and $x_3 < a < x_4$, the instance $(\boldsymbol{x}_{-4}, a)$ is also $(1|2|3, 4)$-well-separated and has $F_3(\boldsymbol{x}_{-4}, a) = a$, due to F's strategyproofness.

Let $F_3(\boldsymbol{x}) = x_4$ (the case where $F_3(\boldsymbol{x}) = x_3$ is symmetric). Since $\boldsymbol{x}$ is an $(1|2|3, 4)$-well-separated instance, both x_3 and x_4 are served by the facility at x_4. Hence, there is a x_3-hole (l, r) in the image set $I_3(\boldsymbol{x}_{-3})$. We note that $3\lambda^2 + \lambda = x_3 < r \leq x_4 = 3\lambda^2 + \lambda + 1$, since $x_3 \notin F(\boldsymbol{x})$ and $x_4 \in F(\boldsymbol{x})$, and that $l \geq \lambda^2 + \lambda - 1$. The latter holds because if $l < \lambda^2 + \lambda - 1$, then $y = 2\lambda^2 + \lambda$ would lie in the right half of the hole (l, r). Thus, if agent 3 moves to y, by strategyproofness, the nearest facility to y in $F(\boldsymbol{x}_{-3}, y)$ would be at $r > 3\lambda^2 + \lambda$, and thus $\mathrm{cost}[F(\boldsymbol{x}_{-3}, y)] > \lambda^2$. Since the optimal cost for $(\boldsymbol{x}_{-3}, y)$ is λ, F's approximation ratio would be $\lambda > \rho$.

Let us now consider the instance $\boldsymbol{x}' = (\boldsymbol{x}_{-3}, l + \varepsilon)$, where $\varepsilon \in (0, 1]$ is chosen small enough that $l + \varepsilon$ lies in the left half of the hole (l, r) and the instance $(0, \lambda, l, l + \varepsilon)$ is $(1|2|3, 4)$-well-separated. Since F is strategyproof, and since l is the nearest point to $l + \varepsilon$ in $I_3(\boldsymbol{x}_{-3})$, $l \in F(\boldsymbol{x}')$. Then, we consider the instance $\boldsymbol{x}'' = (\boldsymbol{x}'_{-4}, l)$. Since F is anonymous and strategyproof, and since $l \in F(\boldsymbol{x}')$, $x''_3 = l \in F(\boldsymbol{x}'')$. Moreover, by Proposition 2, $x''_4 = l + \varepsilon \in F(\boldsymbol{x}'')$, because for the $(1|2|3, 4)$-well-separated instance $\boldsymbol{x}$, $F_3(\boldsymbol{x}) = x_4$, and $\boldsymbol{x}''$ is an $(1|2|3, 4)$-well-separated instance with $x''_4 \leq x_4$. Since both $x''_3, x''_4 \in F(\boldsymbol{x}'')$, either the agents 1 and 2 are served by the same facility of $F(\boldsymbol{x}'')$ or the agent 2 is served by the facility at l. In both cases, $\mathrm{cost}[F(\boldsymbol{x}'')] \geq \lambda$. But the optimal cost for $\boldsymbol{x}''$ is $\varepsilon \leq 1$, and F's approximation ratio is at least $\lambda > \rho$. □

5 Inexistence of Nice Mechanisms for $K = 2$ and General Metrics

Next, we consider 3-location instances of 2-Facility Location with $n \geq 3$ agents in a metric space consisting of 3 half-lines $[0, \infty)$ with a common origin O. This is conceptually equivalent to a continuous star with center O and 3 long branches starting at O. So, we refer to this metric as S_3, and to its 3 branches as b_1, b_2, and b_3. We show that:

Theorem 5. *Any deterministic strategyproof mechanism for 2-Facility Location with $n \geq 3$ agents in S_3 has an unbounded approximation ratio.*

Proof sketch. We first extend Theorem 2 so that we characterize nice mechanisms for 2-Facility Location with 3 agents in S_3, when all agents are located on (at most) two fixed branches. To this end, we use that for instances where the 3 agents lie on 2 branches only, nice mechanisms cannot take any advantage of the third branch.

Next we show that for all i-well-separated instances with the 3 agents on different branches of S_3, (i) a nice mechanism places the facility serving agents j and k in the closed interval between them, and (ii) if for such an instance, a facility is allocated to agent j, then as long as we "push" the location of j towards O, a facility stays with j.

For sake of contradiction, we let F be any nice mechanism for 2-Facility Location in S_3, and let i (resp. k) be the partial dictator of F on the line $b_1 - b_2$ (resp. $b_1 - b_3$, if F does not admit a partial dictator, i and k are any two agents). We consider an i-well-separated instance $\boldsymbol{x}$ where i is on the branch b_1, j is on b_2, and k is on b_3. Thus, agents

j and k are served by the same facility of $F(\boldsymbol{x})$, which, by property (i) above, is located between them. Wlog., we can assume that this facility is allocated to j. Then, "pushing" the location of j to the center O of S_3, we obtain an i-well separated instance $\boldsymbol{x}'$ on the line $b_1 - b_3$ for which F allocates a facility to agent j in the middle. This contradicts the hypothesis that $F(\boldsymbol{x}')$ places the facilities at the two extremes, because agent k is the dictator of F on the line $b_1 - b_3$.

Then, using partial group strategyproofness, we extend the proof to instances with $n \geq 3$ agents. To this end, we consider a 3-location instance where a coalition of $n-2$ agents plays the role of i, and the remaining 2 agents play the role of j and k. □

References

1. Alon, N., Feldman, M., Procaccia, A.D., Tennenholtz, M.: Strategyproof approximation of the minimax on networks. Mathematics of Operations Research 35(3), 513–526 (2010)
2. Barberà, S.: An introduction to strategyproof social choice functions. Social Choice and Welfare 18, 619–653 (2001)
3. Barberà, S., Beviá, C.: Locating public libraries by majority: Stability, consistency and group formation. Games and Economic Behaviour 56, 185–200 (2006)
4. Dokow, E., Feldman, M., Meir, R., Nehama, I.: Mechanism design on discrete lines and cycles. In: Proc. of the 13th ACM Conf. on Electronic Commerce (EC 2012), pp. 423–440 (2012)
5. Escoffier, B., Gourvès, L., Thang, N.K., Pascual, F., Spanjaard, O.: Strategy-proof mechanisms for facility location games with many facilities. In: Brafman, I., Roberts, F., Tsoukiás, A. (eds.) ADT 2011. LNCS (LNAI), vol. 6992, pp. 67–81. Springer, Heidelberg (2011)
6. Feldman, M., Wilf, Y.: Strategyproof Facility Location and the least squares objective. In: Proc. of the 14th ACM Conference on Electronic Commerce, EC 2013 (2013)
7. Fotakis, D., Tzamos, C.: Strategyproof Facility Location with concave costs. In: Proc. of the 14th ACM Conference on Electronic Commerce, EC 2013 (2013)
8. Fotakis, D., Tzamos, C.: Winner-imposing strategyproof mechanisms for multiple Facility Location games. Theoretical Computer Science 472, 90–103 (2013)
9. Ju, B.-G.: Efficiency and consistency for locating multiple public facilities. Journal of Economic Theory 138, 165–183 (2008)
10. Koutsoupias, E.: Scheduling without payments. In: Persiano, G. (ed.) SAGT 2011. LNCS, vol. 6982, pp. 143–153. Springer, Heidelberg (2011)
11. Lu, P., Sun, X., Wang, Y., Zhu, Z.A.: Asymptotically optimal strategyproof mechanisms for Two-Facility Games. In: Proc. of the 11th ACM Conf. on Electronic Commerce (EC 2010), pp. 315–324 (2010)
12. Lu, P., Wang, Y., Zhou, Y.: Tighter bounds for Facility Games. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 137–148. Springer, Heidelberg (2009)
13. Miyagawa, E.: Locating libraries on a street. Social Choice and Welfare 18, 527–541 (2001)
14. Moulin, H.: On strategy-proofness and single-peakedness. Public Choice 35, 437–455 (1980)
15. Nissim, K., Smorodinsky, R., Tennenholtz, M.: Approximately optimal mechanism design via Differential Privacy. In: Proc. of the 3rd Conference on Innovations in Theoretical Computer Science (ITCS 2012), pp. 203–213 (2012)
16. Procaccia, A.D., Tennenholtz, M.: Approximate mechanism design without money. In: Proc. of the 10th ACM Conference on Electronic Commerce (EC 2009), pp. 177–186 (2009)
17. Schummer, J., Vohra, R.V.: Strategyproof location on a network. Journal of Economic Theory 104, 405–428 (2002)
18. Schummer, J., Vohra, R.V.: Mechanism design without money. Algorithmic Game Theory 10, 243–299 (2007)

ℓ_2/ℓ_2-Foreach Sparse Recovery with Low Risk

Anna C. Gilbert[1,⋆], Hung Q. Ngo[2], Ely Porat[3],
Atri Rudra[2], and Martin J. Strauss[1]

[1] University of Michigan
[2] University at Buffalo (SUNY)
[3] Bar-Ilan University

Abstract. In this paper, we consider the "foreach" sparse recovery problem with failure probability p. The goal of the problem is to design a distribution over $m \times N$ matrices Φ and a decoding algorithm A such that for every $\mathbf{x} \in \mathbb{R}^N$, we have with probability at least $1 - p$

$$\|\mathbf{x} - A(\Phi\mathbf{x})\|_2 \leqslant C\|\mathbf{x} - \mathbf{x}_k\|_2,$$

where $\mathbf{x}_k$ is the best k-sparse approximation of $\mathbf{x}$.

Our two main results are: (1) We prove a lower bound on m, the number measurements, of $\Omega(k\log(n/k) + \log(1/p))$ for $2^{-\Theta(N)} \leqslant p < 1$. Cohen, Dahmen, and DeVore [4] prove that this bound is tight. (2) We prove nearly matching upper bounds that also admit *sub-linear* time decoding. Previous such results were obtained only when $p = \Omega(1)$. One corollary of our result is an an extension of Gilbert et al. [6] results for information-theoretically bounded adversaries.

1 Introduction

In a large number of modern scientific and computational applications, we have considerably more data than we can hope to process efficiently and more data than is essential for distilling useful information. Sparse signal recovery [7] is one method for both reducing the amount of data we collect or process initially and then, from the reduced collection of observations, recovering (an approximation to) the key pieces of information in the data. Sparse recovery assumes the following mathematical model: a data point is a vector $\mathbf{x} \in \mathbb{R}^N$, using a matrix Φ of size $m \times N$, where $m \ll N$, we collect "measurements" of $\mathbf{x}$ non-adaptively and linearly as $\Phi\mathbf{x}$; then, using a "recovery algorithm" A, we return a good approximation to $\mathbf{x}$. The error guarantee must satisfy

$$\|\mathbf{x} - A(\Phi\mathbf{x})\|_2 \leqslant C\|\mathbf{x} - \mathbf{x}_k\|_2, \tag{1}$$

where C is a constant (ideally arbitrarily close to 1) and $\mathbf{x}_k$ is the best k-sparse approximation of $\mathbf{x}$. This is customarily called an ℓ_2/ℓ_2-*error guarantee* in the

⋆ A full version of this paper may be found at `http://arxiv.org/abs/1304.6232`. AG is supported in part by NSF CCF 1161233, HN by NSF grant CCF-1161196, AR by NSF CAREER grant CCF-0844796 and NSF grant CCF-1161196, and MS by NSF CCF 0743372 and NSF CCF 1161233.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 461–472, 2013.

literature. This paper considers the sparse recovery problem with failure probability p, the goal of which is to design a distribution over $m \times N$ matrices Φ and a decoding algorithm A such that for every $\mathbf{x} \in \mathbb{R}^N$, the error guarantee holds with probability at least $1 - p$. The reader is referred to [7] and the references therein for a survey of sparse matrix techniques for sparse recovery, and to [1] for a collection of articles (and the references therein) that emphasize the applications of sparse recovery in signal and image processing.

There are many parameters of interest in the design problem: (i) number of measurements m; (ii) decoding time, i.e. runtime of algorithm A; (iii) approximation factor C and (iv) failure probability p. We would like to minimize all the four parameters simultaneously. It turns out, however, that optimizing the failure probability p can lead to wildly different recovery schemes. Much of the sparse recovery or compressive sensing literature has focused on the case of either $p = 0$ (which is called the "*forall*" model) or $p = \Omega(1)$ (the "*foreach*" model). Cohen, Dahmen, and DeVore [5] showed a lower bound of $m = \Omega(N)$ for the number of measurements when $p = 0$, rendering a sparse recovery system useless as one must collect (asymptotically) as many measurements as the length of the original signal[1]. Thus, algorithmically there is not much to do in this regime.

The case of $p \geqslant \Omega(1)$ has resulted in much more algorithmic success. Candés and Tao showed in [2] that $O(k \log(N/k))$ random measurements with a polynomial time recovery algorithm are sufficient for compressible vectors and Cohen et al. [5] show that $O(k \log(N/k))$ measurements are sufficient for any vector (but the recovery algorithm given is not polynomial time). In a subsequent paper, Cohen et al. [4] give a polynomial time algorithm with $O(k \log(N/k))$ measurements. The next goal was to match the $O(k \log(N/k))$ measurements but with *sub-linear* time decoding. This goal was achieved by Gilbert, Li, Porat, and Strauss [8] who showed that there is a distribution on $m \times N$ matrices with $m = O(k \log(N/k))$ and a decoding algorithm A such that, for each $\mathbf{x} \in \mathbb{R}^N$ the ℓ_2/ℓ_2-error guarantee is satisfied with probability $p = \Theta(1)$. The next natural goal was to nail down the correct dependence on $C = 1+\varepsilon$. Gilbert et al.'s result actually needs $O(\frac{1}{\varepsilon} k \log(N/k))$ measurements. This was then shown to be tight by Price and Woodruff [21].

At this point, we completely understand the problem for the case of $p = 0$ or $p = \Omega(1)$. Somewhat surprisingly, there is *no* work that has explicitly considered the ℓ_2/ℓ_2 sparse recovery problem when $0 < p \leqslant o(1)$. The main goal of this paper is to close this gap in our understanding.

Given the importance of the sparse recovery problem, we believe that it is important to close the gap. Similar studies have been done extensively in a closely related field: coding theory. While the model of worst-case errors pioneered by Hamming (which corresponds to the forall model) and the oblivious/stochastic error model pioneered by Shannon (which corresponds to the foreach model) are most well-known, there is a rich set of results in trying to understand the

[1] For this reason, all of the forall sparse signal recovery results satisfy a different, weaker error guarantee. E.g. in the ℓ_1/ℓ_1 forall sparse recovery we replace the condition (1) by $\|\mathbf{x} - A(\Phi\mathbf{x})\|_1 \leqslant C\|\mathbf{x} - \mathbf{x}_k\|_1$.

power of intermediate channels, including the arbitrarily varying channel [14]. Another way to consider intermediate channels is to consider computationally bounded adversaries [16]. Gilbert et al. [6] considered a computationally bounded adversarial model for the sparse recovery problem in which signals are generated neither obliviously (as in the foreach model) nor adversarially (in the forall model) in order to interpolate between the forall and foreach signal models. Our results in this paper imply new results for the ℓ_1/ℓ_1 sparse recovery problem as well as the ℓ_2/ℓ_2 sparse recovery problem against bounded adversaries.

Our main contributions are as follows.

1. We prove that the number measurements has to be $\Omega(k\log(N/k)+\log(1/p))$ for $2^{-\Theta(N)} \leqslant p < 1$.
2. We prove nearly matching upper bounds that also admit *sub-linear* time decoding.
3. We present applications of our result to obtain
   (i) the best known number of measurements for ℓ_1/ℓ_1 *forall* sparse recovery with sublinear $(\mathrm{poly}(k, \log N))$ time decoding (in [9]), and
   (ii) nearly tight upper and lower bounds on the number of measurements needed to perform ℓ_2/ℓ_2-sparse recovery against information-theoretically bounded adversary.

As was mentioned earlier, there are many parameters one could optimize. We will not pay very close attention to the approximation factor C, other than to stipulate that $C \leqslant O(1)$. In most of our upper bounds, we can handle $C = 1+\varepsilon$ for an arbitrary constant ε, but optimizing the dependence on ε is beyond the scope of this paper.

Lower Bound Result. We prove a lower bound of $\Omega(\log(1/p))$ on the number of measurements when the failure probability satisfies $2^{-\Theta(N)} \leqslant p < 1$. (When $p \leqslant 2^{-\Omega(N)}$, our results imply a tight bound of $m = \Omega(N)$.) The $\Omega(\log(1/p))$ lower bound along with the lower bound of $\Omega(k\log(N/k))$ from [21] implies the final form of the lower bound claimed above. The obvious follow-up question is whether this bound is tight. Indeed, an upper bound result Cohen, Dahmen, and DeVore [4] proves that this bound is tight if we only care about polynomial time decoding. Thus, the interesting algorithmic question is how close we can get to this bound with sub-linear time decoding.

Upper Bound Results. For the upper bounds, we provide several algorithms that span the trade-offs between number of measurements and failure probability. For completeness, we include the running times and the space requirements of the algorithms and measurement matrices in Table 1, which summarizes our main results and compares them with existing results.

We begin by first considering the most natural way to boost the failure probability of a given ℓ_2/ℓ_2 sparse recovery problem: we repeat the scheme s times with independent randomness and pick the "best" answer– see the full version [9] for more details. This boosts the decoding error probability from the original p to $p^{\Omega(s)}$– though the reduction does blow up the approximation factor by a multiplicative factor of $\sqrt{3}$.

Table 1. Summary of algorithmic results. The results in [20] are for ℓ_1/ℓ_1 forall sparse recovery but their results can be easily adapted to our setting with our proofs. c is some constant $\geqslant 8$ and $\alpha > 0$ is any arbitrary constant and we ignore the constant factors in front of all the expressions.

Reference	k	m	p	Decoding time	Space
[4]	Any k	$k\log(N/k) + \log(1/p)$	Any p	$\text{poly}(N)$	$\text{poly}(N)$
[8]	Any k	$k\log(N/k)$	$p \geqslant \Omega(1)$	$k \cdot \text{poly}\log N$	$k \cdot \text{poly}\log N$
[20]	$k \geqslant N^{\Omega(1)}$	$k\log(N/k)$	$p = (N/k)^{-k/\log^c k}$	$k^{1+\alpha}\text{poly}\log N$	$Nk^{0.2}$
	Any k	$k\log(N/k)\log_k^c N$	$p = k^{-k/\log^c k}$	$k^{1+\alpha}\text{poly}\log N$	$Nk^{0.2}$
[8] + weak/top conv.	Any k	$k\log(N/k)$	$p \geqslant 2^{-k/\log^c k}$	$k \cdot \text{poly}\log N$	$k \cdot \text{poly}\log N$
This paper	$k \geqslant N^{\Omega(1)}$	$k\log(N/k)$	$p = (N/k)^{-k/\log^c k}$	$k^{1+\alpha}\text{poly}\log N$	$k \cdot \text{poly}\log N$
	Any $k \geqslant \log(N/k)$	$k\log(N/k)\log_k^\alpha N$	$p = (N/k)^{-k/\log^c k}$	$k^{2^{\alpha^{-1}}}\text{poly}\log N$	$k \cdot \text{poly}\log N$

The above implies that if we can optimally solve the ℓ_2/ℓ_2 sparse recovery problem for $p \geqslant (N/k)^{-k}$ (i.e. with $O(k\log(N/k))$ measurements), then we can solve the problem optimally for smaller p. Hence, for the rest of the description we focus on the case $p \geqslant (N/k)^{-O(k)}$ (where the goal is to obtain $m = O(k\log(N/k))$). Note that in this case, the amplification does not help as even for $p = \Omega(1)$, previous results (e.g. [21]) imply that $m \geqslant \Omega(k\log(N/k))$. Thus, if the original decoding error probability is p then to obtain the $(N/k)^{-k}$ decoding error probability implies that the number of measurements will be larger than the optimal value a factor of $k\log(N/k)/\log(1/p)$. As we will see shortly the best know upper bound can achieve $p = 2^{-k}$, which implies that amplification will be larger than the optimal value of $\Omega(k\log(N/k)$ by a factor of $\log(N/k)$. In this work, we show how to achieve the same goal with an asymptotically smaller blow-up.

For $p \geqslant (N/k)^{-k}$, there are two related works. The first is that of Porat and Strauss [20] who considered the sparse recovery problem under the ℓ_1/ℓ_1 forall guarantee. Despite the different error guarantee, our construction is closely related to that of [20] and our proofs imply the results for [20] listed in Table 1. Note that the results for polynomially large k are pretty much the same except we have a better space complexity. For general k, our result also has better number of measurements and failure probability guarantee. The second work is that of Gilbert et al. [8]. Even though the results in that paper are cited for $p \geqslant \Omega(1)$, it can be shown that if one uses $O(k)$-wise independent random variables instead of the pair-wise independent random variables as used in [8], one can obtain a "weak system" with failure probability 2^{-k}. Then our "weak system to top level system conversion" leads to the result claimed in the second to last row in Table 1. Our results have a better failure probability at the cost of larger number of measurements.

It is natural to ask whether decreasing the failure probability (the base changed from 2 to (N/k)) is worth giving up the optimality in the number of measurements (which is what [8] obtains). Note that achieving a failure probability of $(N/k)^{-k}$ is a very natural goal and our results are better than those in [8] when we anchor on the failure probability goal first.

Bounded Adversary Results. We also obtain some results for ℓ_2/ℓ_2-sparse recovery against information-theoretically-bounded adversaries as considered by Gilbert et al. [6]. (See Section 2 for a formal definition of such bounded adversaries.) Gilbert et al. show that $O(k \log(N/k))$ measurements is sufficient for such adversaries with $O(\log N)$ bits of information. Our results allow us to prove results for a general number of information bits bound of s. In particular, we observe that for such adversaries $O(k \log(N/k) + s)$ measurements suffice. Further, if one desires sublinear time decoding then our results in Table 1 allows for a similar conclusion but with extra $\operatorname{poly} \log k$ factors. We also observe that one needs $\tilde{\Omega}(\sqrt{s})$ many measurements against such an adversary (assuming the entries are polynomially large). In the final version of the paper, we will present a proof suggested to us by an anonymous reviewer that leads to the optimal $\Omega(s)$ lower bound.

Lower Bound Techniques. Our lower bound technique is inspired by the geometric approach of Cohen et al. [5] for the $p = 0$ case. Our bound holds for the entire range of failure probability p. Our technique also yields a simpler and more intuitive proof of Cohen et al. result. Both results hold even for sparsity $k = 1$.

The technical crux of the lower bound result in [5] for $p = 0$ is to show that any measurement matrix Φ with $O(N/C^2)$ rows has a null space vector $\mathbf{n}$ that is "non-flat," – i.e. $\mathbf{n}$ has one coordinate that has most of the mass of $\mathbf{n}$. On the other hand since $\Phi\mathbf{n} = \mathbf{0}$, the decoding algorithm A has to output the same answer when $\mathbf{x} = \mathbf{n}$ and when $\mathbf{x} = \mathbf{0}$. It is easy to see that then A does not satisfy (1) for at least one of these two cases (the output for $\mathbf{0}$ has to be $\mathbf{0}$ while the output for $\mathbf{n}$ has to be non-flat and in particular not $\mathbf{0}$).

To briefly introduce our technique, consider the case of $p = 2^{-N}$ (where we want a lower bound of $m = \Omega(N)$). The straightforward extension of Cohen, et al.'s argument is to define a distribution over, say, all the unit vectors in $\mathbb{R}^N$, argue that this gives a large measure of "bad vectors," and then apply Yao's minimax lemma to obtain our final result; i.e., that there are "a lot" of non-flat vectors in the null space of a given matrix Φ. This argument fails because the distribution on the bad vectors must be independent of the measurement matrix Φ (and algorithm A) in order to apply Yao's lemma but null space vectors, of course, depend on Φ. On the other hand, if we define the "hard" distribution to be the uniform distribution, then the measure of null space vectors for any $m \geqslant 1$ is zero, and thus this obvious generalization does not work.

We overcome this obstacle with a simple idea. The hard distribution is still the uniform distribution on the unit sphere S^{N-1}. We first show that there is a region R on this sphere with large measure ($\geqslant p$) such that *all* vectors in R have a *positive* "spike" (large mass) at one particular coordinate $j^* \in [N]$. (The region R is simply a small *spherical cap* about the unit vector $\mathbf{e}_{j^*}$.) In particular, to recover an input vector $\mathbf{v} \in R$, the algorithm has to assign a large positive mass to the j^*th coordinate of $A(\Phi\mathbf{v})$. Next, by applying a certain invertible linear reflector to R, we can construct a region R' (which is also a region on the sphere, and is just a reflection of R) with the same measure satisfying the following: for

each vector $\mathbf{v} \in R$, the reflection $\mathbf{v}'$ of $\mathbf{v}$ ($\mathbf{v}' \in R'$) has a *negative* spike at the same coordinate j^*; furthermore, $\Phi\mathbf{v} = \Phi\mathbf{v}'$, which forces the algorithm A into a dichotomy. The algorithm can not recover both $\mathbf{v}$ and $\mathbf{v}'$ well at once. Roughly speaking, the algorithm will be wrong with probability at least half the total measure of R and R', which is p. Finally, Yao's lemma completes the lower bound proof. There are some additional technical obstacles that we need to overcome in this step—see [9] for more details.

Upper Bound Techniques. We believe that our main algorithmic contributions are the new techniques that we introduce in this paper, which should be useful in (similar) applications.

Our upper bounds follows the same outline used by Gilbert, Li, Porat and Strauss [8] and Porat and Strauss [20]. At a high level, the construction follows three steps. The first step is to design an "identification scheme," which in sub-linear time computes a set $S \subseteq [N]$ of size roughly k that contains $\Omega(k)$ of the "heavy hitters." (Heavy hitters are the coordinates where if the output vector does not put in enough mass then (1) will not be satisfied.) In the second step, we develop a "weak level system" which essentially estimates the values of coordinates in S. Finally, using a loop invariant iterative scheme, we convert the weak system into a "top level system," which is the overall system that we want to design. (The way this iterative procedure works is that it makes sure that after iteration i, one is missing only $O(k/2^i)$ heavy hitters– so after $\log k$ steps we would have recovered all of them.) The last two steps are designed to run in time $|S| \cdot \operatorname{poly} \log N$, so if the first step runs in sub-linear time, then the overall procedure is sub-linear. [2]. Our main contribution is in the first step, so we will focus on the identification part here. The second step (taking median of measurements like Count-Sketch [3]) is standard [12].

In order to highlight and to summarize our technical contributions, we present an overview of the scheme in [20] (when adapted to the ℓ_2/ℓ_2 sparse recovery problem). We focus only on the identification step. For near-linear time identification, one uses a lossless bipartite expander where each edge in the adjacency matrix is replaced by a random ± 1 value. The intuition is that because of the expansion property most heavy hitters will not collide with another heavy hitter in most of the measurements it participates in. Further, the expansion property implies that the ℓ_2^2 noise in most of the neighboring measurements will be low. (The random ± 1 is a standard trick to convert this to an low ℓ_2 noise.) Thus, if we define the value of an index to be median value of all the measurements, then we should get very good estimates for most of the heavy hitters (and in particular, we can identify them by outputting the top $O(k)$ median values). Since this step implies computing N medians overall we have a near linear time computation. However, note that if we had access to a subset $S' \subseteq [N]$ that had

[2] We would also like to point out that Gilbert et al.'s construction has a failure probability of $\Omega(1)$ in the very first iteration of the last step (weak to top level system conversion) and it seems unlikely that this can be made smaller without significantly changing their scheme.

most of the heavy hitters in it, we can get away with a run time nearly linear in $|S'|$ (by just computing the medians in S').

This seems like a chicken and egg problem as the set S' is what we were after to begin with! Porat and Strauss use recursion to compute S' in sub-linear time. (The scheme was also subsequently used by Ngo, Porat and Rudra to design near optimal sub-linear time decodable ℓ_1/ℓ_1 forall sparse recovery schemes for non-negative signals [19].) To give the main intuition, consider the scheme that results in $\tilde{O}(\sqrt{N})$ identification time. We think of the domain $[N]$ as $L \times R$, where both L and R are isomorphic to $[\sqrt{N}]$. (Think of L as the first $\frac{\log N}{2}$ bits in the $\log N$-bit representation of any index in $[N]$ and R to be the remaining bits.) If one can obtain lists $S_L \subset L$ and $S_R \subset R$ that contain the projections of the heavy hitters in L and R, respectively, then $S_L \times S_R$ will contain all the heavy hitters, i.e. $S' \subseteq S_L \times S_R$. (We can use the near linear time scheme to obtain S_L and S_R in $\tilde{O}(\sqrt{N})$ time in the base case. One also has to make sure that when going from $[N]$ to a domain of size $\sqrt{N}$, not too many heavy hitters collide. This can be done by, say, randomly permuting $[N]$ before applying the recursive scheme.) The simplest thing to do would be to set $S' = S_L \times S_R$. However, since both $|S_L|$ and $|S_R|$ can be $\Omega(k)$, this step itself will take $\Omega(k^2)$ time, which is too much if we are shooting for a decoding time of $k^{1+\alpha}\text{poly}\log N$ for $\alpha < 1$. The way Porat and Strauss solved this problem was to store the whole inversion map as a table. This allowed $k \cdot \text{poly}\log N$ decoding time but the scheme ended up needing $\Omega(N)$ space overall.

To get a running time of $k^{1+\alpha}\text{poly}\log N$ one needs to apply the recursive idea with more levels. One can think of the whole procedure as a recursion tree with $\mathcal{N} = O(\log_k N)$ nodes. Unfortunately, this process introduces another technical hurdle. At each node, the expander based scheme loses some, say ζ, fraction of the heavy hitters. To bound the overall fraction of lost heavy hitters, Porat and Strauss use the naive union bound of $\zeta \cdot \mathcal{N}$. However, we need the overall fraction of lost heavy hitters to be $O(1)$. This in turn introduces extra factors of $\log_k N$ in the number of measurements (resulting in the ultimate number of measurements of $k \log(N/k) \log_k^8 N$ in [20]).

We are now ready to present the new ideas that improve upon Porat and Strauss' solutions to solve the two issues raised above. Instead of dividing $[N]$ into $[\sqrt{N}] \times [\sqrt{N}]$, we first apply a code $\mathcal{C} : [N] \to [\sqrt[b]{N}]^r$. (Note that the Porat Strauss construction corresponds to the case when $r = b = 2$ and $\mathcal{C}$ just "splits" the $\log N$ bits into two equal parts.) Thus, in our recursive algorithm at the root we will get r subsets $S_1, \ldots, S_r \subseteq [\sqrt[b]{N}]$ with the guarantee that for (most) $i \in [r]$, S_i contains $\mathcal{C}(j)_i$ for most heavy hitters j. Thus, we need to recover the j's for which the condition in the last sentence is true. This is *exactly* the list recovery problem that has been studied in the coding theory literature. (See e.g. [22].) Thus, if we can design a code $\mathcal{C}$ that solves the list recovery problem very efficiently, we would solve the first problem above[3]. For the second problem,

[3] We would like to point out that [19] also uses list recoverable codes but those codes are used in a different context: they used it to replace expanders and further, the codes have the traditional parameters.

note that since we are using a code $\mathcal{C}$, even if we only have $\mathcal{C}(j)_i \in S_i$ for say $r/2$ positions $i \in [r]$, we can recover all such indices j. In other words, unlike in the Porat Strauss construction where we can lose a heavy hitter even if we lose it in any of the $\mathcal{N}$ recursive call, in our case we only lose a heavy hitter if it is lost in *multiple* recursive calls. This fact allows us to do a better union bound than the naive one used in [20].

The question then is whether there exists code $\mathcal{C}$ with the desired properties. The most crucial part is that the code needs to have a decoding algorithm whose running time is (near) linear in $\max_{i\in[r]} |S_i|$. Further, we need such codes with $r = O(1)$, i.e. of constant block length independent of $\max_{i\in[r]} |S_i|$. Unfortunately, the known results on list recovery, be it for Reed-Solomon codes [11] or folded Reed-Solomon codes [10] do not work well in this regime—these results need $r \geqslant \Omega(\max_{i\in[r]} |S_i|)$, which is way too expensive. For our setting, the best we can do with Reed-Solomon list recovery is to do the naive thing of going through all possibilities in $\times_{i\in[r]} S_i$. (These codes however can correct for optimal number of errors and lead to our result in the last row of Table 1.) Fortunately, a recent result of Ngo, Porat, Ré and Rudra [18] gave an algorithmic proof of the Loomis-Whitney inequality [17]. The (combinatorial) Loomis-Whitney inequality has found uses in theoretical computer science before [13,15]. In this work, we present the first application of the algorithmic Loomis Whitney inequality of [18] and show that it naturally defines a code C with the required (algorithmic) list recoverability. This code leads to the result in the second to last row of Table 1. Interestingly, we get *optimal* weak level systems by this method. We lose in the final failure probability because of the weak level to top level system conversion.

We conclude the contribution overview by pointing out three technical aspects of our results.

- As was mentioned earlier, we first randomly permute the columns of the matrix to make the recursion work. To complete our identification algorithm, we need to perform the inverse operation on the indices to be output. The naive way would be to use a table lookup, which will require $O(N \log N)$ space, but would still be an improvement over [20]. However, we are able to exploit the specific nature of the recursive tree and the fact that our main results use the Reed-Solomon code and the code based on Loomis-Whitney inequality to have sub-linear space usage.
- In the weak level to top level system, both Gilbert et al., and Porat and Strauss decrease the parameters geometrically—however in our case, we need to use different decay functions to obtain our failure probability.
- Unlike the argument in [20] we explicitly use an expander while Porat and Strauss used a random graph. However, because of this, [20] need at least N-wise independence in their random variables to make their argument go through. Our use of expanders allows us to get away with using only $\tilde{O}(k)$-wise independence, which among others leads to our better space usage.

2 Preliminaries

We fix notations, terminology, and concepts that will be used throughout the paper. Let $[N]$ denote the set $\{1, \ldots, N\}$. Let $G : [N] \times [\ell] \to [M]$ be an ℓ-regular bipartite graph, and $\mathcal{M}_G$ be its adjacency matrix. We will often switch back and forth between the graph G and the matrix $\mathcal{M}_G$.For any subset $S \subseteq [N]$, let $\Gamma(S) \subseteq [M]$ denote the set of neighbor vertices of S in G. Further, let $\mathcal{E}(S)$ denote the set of edges incident on S. A bipartite graph $G : [N] \times [\ell] \to [M]$ is a (t, ε)-expander if for every subset $S \subseteq [N]$ of $|S| \leqslant t$, we have $|\Gamma(S)| \geqslant |S|\ell(1-\varepsilon)$. Several expander properties used in our proofs are listed in the complete paper [9], along with some probability basics.

Sparse Recovery Basics. For a vector $\mathbf{x} = (x_i)_{i=1}^N \in \mathbb{R}^N$, the set of k highest-magnitude coordinates of $\mathbf{x}$ is denoted by $H_k(\mathbf{x})$. Such elements are called *heavy hitters*. Every element $i \in [N] \setminus H_k(\mathbf{x})$ such that $|x_i| \geqslant \sqrt{\frac{\zeta^2\eta}{k}} \cdot \|\mathbf{z}\|_2$ will be called a *heavy tail* element. Here, ζ and η are constants that will be clear from context. All the remaining indices will be called *light tail* elements; let $\mathcal{L}$ denote the set of light tail elements. A vector $\mathbf{w} = (w_i)_{i=1}^N \in \mathbb{R}^N$ is called a *flat tail* if $w_i = 1/|S|$ for every non-zero w_i, where $S = \mathrm{supp}(\mathbf{w})$.

Definition 1. *A probabilistic $m \times N$ matrix $\mathcal{M}$ is called an (k, C)-approximate sparse recovery system or* (k, C)-top level system *with failure probability p if there exists a* decoding algorithm *A such that for every $\mathbf{x} \in \mathbb{R}^N$, the following holds with probability at least $1 - p$:*

$$\|\mathbf{x} - A(\mathcal{M}\mathbf{x})\|_2 \leqslant C \cdot \|\mathbf{x} - \mathbf{x}_{H_k(\mathbf{x})}\|_2.$$

The parameter m is called the number of measurements of the system.

Definition 2. *A probabilistic matrix $\mathcal{M}$ with N columns is called a (k, ζ, η)-*weak identification *matrix with (ℓ, p)-*guarantee *if there is an algorithm that, given $\mathcal{M}\mathbf{x}$ and a subset $S \subseteq [N]$, with probability at least $1 - p$ outputs a subset $I \subseteq S$ such that (i) $|I| \leqslant \ell$ and (ii) at most ζk of the elements of $H_k(\mathbf{x})$ are not present in I. The time taken to compute I will be called* identification time.

Definition 3. *We will call a (random) $m \times N$ matrix $\mathcal{M}$ a (k, ζ, η) weak ℓ_2/ℓ_2 system if the following holds for any vector $\mathbf{x} = \mathbf{y} + \mathbf{z}$ such that $|\mathrm{supp}(\mathbf{y})| \leqslant k$. Given $\mathcal{M}\mathbf{x}$ one can compute $\hat{\mathbf{x}}$ such that there exist $\hat{\mathbf{y}}, \hat{\mathbf{z}}$ that satisfy the following properties: (1) $\mathbf{x} = \hat{\mathbf{x}} + \hat{\mathbf{y}} + \hat{\mathbf{z}}$; (2) $|\mathrm{supp}(\hat{\mathbf{x}})| \leqslant O(k/\eta)$;[4] (3) $|\mathrm{supp}(\hat{\mathbf{y}})| \leqslant \zeta k$; (4) $\|\hat{\mathbf{z}}\|_2 \leqslant (1 + O(\eta)) \cdot \|\mathbf{z}\|_2$*

Bounded Adversary Model. We summarize the relevant definitions of computationally bounded adversaries from [6]. In this setting, Mallory is the name of the process that generates inputs x to the sparse recovery problem. We recall two definitions for Mallory: (i) **Oblivious:** Mallory cannot see the matrix Φ and

[4] This part is different from the weak system in [20], where we have $|\mathrm{supp}(\hat{\mathbf{x}})| \leqslant O(k)$.

generates the signal x independent from Φ. For sparse signal recovery, this model is equivalent to the "foreach" signal model. (ii) **Information-Theoretic:** Mallory's output has bounded mutual information with the matrix. To cast this in a computational light, we say that an algorithm M is *(s-)information-theoretically-bounded* if $M(x) = M_2(M_1(x))$, where the output of M_1 consists of at most s bits. This model is similar to that of the "information bottleneck" [23].

Lemma 1 of [6] relates the information-theoretically bounded adversary to a bound on the success probability of an oblivious adversary. We re-state the lemma for completeness:

Lemma 1. *Pick $\ell = \ell(N)$, and fix $0 < \alpha < 1$. Let A be any randomized algorithm which takes input $x \in \{0,1\}^N$, $r \in \{0,1\}^m$, and "succeeds" with probability $1 - \beta$. Then for any information theoretically bounded algorithm M with space ℓ, $A(M(r), r)$ succeeds with probability at least*

$$\min\{1 - \alpha,\ 1 - \ell/\log(\alpha/\beta)\}$$

over the choices of r.

3 Lower Bounds

Lower Bound for ℓ_2/ℓ_2-Foreach Sparse Recovery with Low Risk. For the sake of completeness we present a simplified version of the proof of the $\Omega(N)$ lower bound from [5] for the ℓ_2/ℓ_2 forall sparse recovery in [9]. Our main result is the following, whose proof can be found in [9].

Theorem 4. *Let $C \geqslant 1$ and p be such that $\sqrt{12 + 16C^2} \cdot e^{-\frac{\ln(6+8C^2)}{2} \cdot N} \leqslant p < 1$. Then, any ℓ_2/ℓ_2 foreach sparse recovery scheme using $m \times N$ measurement matrices Φ with failure probability at most p and approximation factor C must have $m \geqslant \frac{1}{(6+8C^2)\ln(6+8C^2)} \ln\left(\frac{\sqrt{12+16C^2}}{p}\right) = \Omega(\log(1/p))$ measurements.*

Lower Bound for Bounded Adversary Model. In this section, we show the following result (proof is in [9]):

Theorem 5. *Any ℓ_2/ℓ_2 sparse recovery scheme that uses at most b bits in each entry of Φ needs at least $\Omega\left(\sqrt{\frac{s}{b}}\right)$ number of measurements to be successful against an s-information-theoretically-bounded adversary.*

4 Sublinear Decoding

We present known results with polynomial time decoding on ℓ_2/ℓ_2 sparse recovery problem in [9].

Our strategy for designing sub-linear time decodable top level systems will be as follows: we will first design weak identification matrices that have sublinear identification time. Then we (in a black-box manner) convert such matrices to

sub-linear time decodable top level systems. We now present an outline of how we implement our strategy. In [9] we show how expanders can be used to construct various schemes that will be useful later. In [9], we show how to convert weak identification systems to top level systems. The rest of technical development is in designing weak identification system with good parameters.

Our first main result on sub-linear time decodable top levels systems will be:

Theorem 6. *For any $k \geqslant N^{\Omega(1)}$ and $\varepsilon, \alpha > 0$, there exists a $(k, 1+\varepsilon)$-top level system with $O(\varepsilon^{-11} k \log(N/k))$ measurements, failure probability $(N/k)^{-k/\log^{13+\alpha} k}$ and decoding time $\varepsilon^{-4} \cdot k^{1+\alpha} \cdot \log^{O(1)} N$.*[5] *This scheme uses $O_\varepsilon(k \cdot \log^{O(1)} N)$ bits of space.*

In fact, our results also work for $k = N^{o(1)}$ but we then do not get the optimal number of measurements. However, an increase in the decoding time leads to our second main result, which has near-optimal number of measurements.

Theorem 7. *For any $1 \leqslant k \leqslant N$ and $\varepsilon, \alpha > 0$, there exists a $(k, 1+\varepsilon)$-top level system with $O(\varepsilon^{-11} k \log(N/k) \log_k^\alpha N)$ measurements, failure probability $(N/k)^{-k/\log^{13+\alpha} k}$ and decoding time $(k/\varepsilon)^{\Theta(2^{-\alpha})} \cdot \log^{O(1)} N$.*[6] *This scheme uses $O_\varepsilon(k \cdot \log^{O(1)} N)$ bits of space.*

The proofs are deferred to [9].

Consequences for the Bounded Adversary Model. Our first corollary is an upper bound for the information-theoretic bounded adversary and follows directly from Lemma 1 (by setting $\beta = \alpha 2^{-s/\alpha}$) and the result of [5].

Corollary 8. *Fix $0 < \alpha < 1$. There is a randomized sparse signal recovery algorithm that with $m = O(k \log(N/k) + s/\alpha)$ measurements will foil an s-information-theoretically bounded adversary; that is, the algorithm's output will meet the ℓ_2/ℓ_2 error guarantees with probability $1 - \alpha$.*

The algorithm in [5] does not have a sublinear running time. If the goal is to defeat such an adversary and to do so with a sublinear algorithm, we must adjust our measurements accordingly, using Table 1. We note that in [6], there was a single result for $O(\log N)$-information-theoretically bounded adversaries ($O(k \log(N/k)$ measurements are sufficient) and this corollary provides an upper bound for the entire range of parameter s.

References

1. Baraniuk, R.G., Candes, E., Nowak, R., Vetterli, M.: Compressive sampling. IEEE Signal Processing Magazine 25(2) (2008)

[5] The $O(\cdot)$ notation here hides the dependence on α.
[6] The $O(\cdot)$ notation here hides the dependence on α.

2. Candès, E.J., Tao, T.: Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies? IEEE Transactions on Information Theory 52(12), 5406–5425 (2006)
3. 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)
4. Cohen, A., Dahmen, W., De Vore, R.A.: Near Optimal Approximation of Arbitrary Vectors from Highly Incomplete Measurements. Bericht. Inst. für Geometrie und Praktische Mathematik (2007)
5. Cohen, A., Dahmen, W., DeVore, R.: Compressed sensing and best k-term approximation. J. Amer. Math. Soc. 22(1), 211–231 (2009)
6. Gilbert, A.C., Hemenway, B., Rudra, A., Strauss, M.J., Wootters, M.: Recovering simple signals. In: ITA, pp. 382–391 (2012)
7. Gilbert, A.C., Indyk, P.: Sparse recovery using sparse matrices. Proceedings of the IEEE 98(6), 937–947 (2010)
8. Gilbert, A.C., Li, Y., Porat, E., Strauss, M.J.: Approximate sparse recovery: Optimizing time and measurements. SIAM J. Comput. 41(2), 436–453 (2012)
9. Gilbert, A.C., Ngo, H., Porat, E., Rudra, A., Strauss, M.J.: L2/L2-foreach sparse recovery with low risk. ArXiv e-prints, arXiv:1304.6232 (April 2013)
10. Guruswami, V., Rudra, A.: Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. IEEE Transactions on Information Theory 54(1), 135–150 (2008)
11. Guruswami, V., Sudan, M.: Improved decoding of reed-solomon and algebraic-geometry codes. IEEE Transactions on Information Theory 45(6), 1757–1767 (1999)
12. Indyk, P., Ruzic, M.: Near-optimal sparse recovery in the l1 norm. In: FOCS, pp. 199–207 (2008)
13. Irony, D., Toledo, S., Tiskin, A.: Communication lower bounds for distributed-memory matrix multiplication. J. Parallel Distrib. Comput. 64(9), 1017–1026 (2004)
14. Lapidoth, A., Narayan, P.: Reliable communication under channel uncertainty. IEEE Transactions on Information Theory 44, 2148–2177 (1998)
15. Lehman, A.R., Lehman, E.: Network coding: does the model need tuning? In: SODA, pp. 499–504 (2005)
16. Lipton, R.J.: A new approach to information theory. In: Enjalbert, P., Mayr, E.W., Wagner, K.W. (eds.) STACS 1994. LNCS, vol. 775, pp. 699–708. Springer, Heidelberg (1994)
17. Loomis, L.H., Whitney, H.: An inequality related to the isoperimetric inequality. Bull. Amer. Math. Soc. 55, 961–962 (1949)
18. Ngo, H.Q., Porat, E., Ré, C., Rudra, A.: Worst-case optimal join algorithms. In: PODS, pp. 37–48 (2012)
19. Ngo, H.Q., Porat, E., Rudra, A.: Efficiently decodable compressed sensing by list-recoverable codes and recursion. In: STACS, pp. 230–241 (2012)
20. Porat, E., Strauss, M.J.: Sublinear time, measurement-optimal, sparse recovery for all. In: SODA, pp. 1215–1227 (2012)
21. Price, E., Woodruff, D.P.: $(1 + \varepsilon)$-approximate sparse recovery. In: FOCS, pp. 295–304 (2011)
22. Rudra, A.: List Decoding and Property Testing of Error Correcting Codes. PhD thesis, University of Washington (2007)
23. Tishby, N., Pereira, F.C., Bialek, W.: The information bottleneck method. In: The 37th Annual Allerton Conference on Communication, Control, and Computing, pp. 368–377 (1999)

Autoreducibility of Complete Sets for Log-Space and Polynomial-Time Reductions*

Christian Glaßer[1], Dung T. Nguyen[2], Christian Reitwießner[1,3,**], Alan L. Selman[2], and Maximilian Witek[1]

[1] Julius-Maximilians-Universität Würzburg
{glasser,reitwiessner,witek}@informatik.uni-wuerzburg.de
[2] University at Buffalo, The State University of New York
{dtn3,selman}@buffalo.edu
[3] CCTVal, Universidad Técnica Federico Santa María, Valparaíso

Abstract. We investigate the autoreducibility and mitoticity of complete sets for several classes with respect to different polynomial-time and logarithmic-space reducibility notions.

Previous work in this area focused on polynomial-time reducibility notions. Here we obtain new mitoticity and autoreducibility results for the classes EXP and NEXP with respect to some restricted truth-table reductions (e.g., $\leq^{\mathrm{p}}_{\text{2-tt}}, \leq^{\mathrm{p}}_{\text{ctt}}, \leq^{\mathrm{p}}_{\text{dtt}}$).

Moreover, we start a systematic study of logarithmic-space autoreducibility and mitoticity which enables us to also consider P and smaller classes. Among others, we obtain the following results:

- Regarding $\leq^{\log}_{\mathrm{m}}$, $\leq^{\log}_{\text{2-tt}}$, $\leq^{\log}_{\text{dtt}}$ and $\leq^{\log}_{\text{ctt}}$, complete sets for PSPACE and EXP are mitotic, and complete sets for NEXP are autoreducible.
- All $\leq^{\log}_{\text{1-tt}}$-complete sets for NL and P are $\leq^{\log}_{\text{2-tt}}$-autoreducible, and all $\leq^{\log}_{\text{btt}}$-complete sets for NL, P and $\Delta^{\mathrm{P}}_{\mathrm{k}}$ are $\leq^{\log}_{\text{log-T}}$-autoreducible.
- There is a $\leq^{\log}_{\text{3-tt}}$-complete set for PSPACE that is not even $\leq^{\log}_{\text{btt}}$-autoreducible.

Using the last result, we conclude that some of our results are hard or even impossible to improve.

1 Introduction

A set C is called autoreducible if C can be reduced to itself by a reduction that does not query its own input. In this way, each reducibility induces a corresponding autoreducibility notion. The main question in connection with autoreducibility asks whether all complete sets of a certain complexity class are autoreducible. Interestingly, in several cases answering such questions would lead to new separations of complexity classes.

For example, consider the question of whether all polynomial-time truth-table-complete sets for EXP are polynomial-time truth-table-autoreducible.

* Please see the technical report [9] for proofs omitted due to space restrictions.

** The third author is supported by Basal Project PB-821 CCTVal – Centro Científico Tecnológico de Valparaíso

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 473–484, 2013.

Buhrman et al. [5] show that a positive answer results in NL $\neq$ NP while a negative answer implies PH $\neq$ EXP. So the study of the autoreducibility of complete sets is a fascinating and important topic.

Mitoticity is another structural property of complete sets that could lead to separations of complexity classes. A set C is mitotic if it can be partioned into sets C_1 and C_2 such that C, C_1, and C_2 are equivalent. Here again each reducibility induces a corresponding notion of mitoticity.

Over the past 20 years, researchers were able to solve autoreducibility and mitoticity questions for several complexity classes and with respect to several polynomial-time reducibility notions. With our paper we further develop this knowledge in two ways: First, we extend techniques by Buhrman et al. [5] to show new mitoticity results for EXP and NEXP. Second, we start a systematic investigation of autoreducibility and mitoticity for logarithmic-space reductions. Since the previous research was concerned with polynomial-time reductions, it did not produce conclusions about P or smaller classes.

With respect to polynomial-time 2-truth-table reducibility ($\leq^{\mathrm{p}}_{\text{2-tt}}$) we show that all EXP-complete sets are mitotic and all NEXP-complete sets are autoreducible. With respect to logspace reducibilities (e.g., $\leq^{\log}_{\mathrm{m}}$, $\leq^{\log}_{\text{2-tt}}$, $\leq^{\log}_{\text{btt}}$, $\leq^{\log}_{\mathrm{T}}$) we obtain several autoreducibility and mitoticity results for complete sets of the classes NL, P, PSPACE, EXP, and NEXP. Table 1 summarizes previously known results and their references together with results newly obtained in this paper. For example, we show:

(i) All $\leq^{\log}_{\text{2-tt}}$-complete sets for PSPACE are $\leq^{\log}_{\text{2-tt}}$-mitotic.
(ii) All $\leq^{\log}_{\text{2-tt}}$-complete sets for EXP are $\leq^{\log}_{\text{2-tt}}$-mitotic.

Note that in both cases, mitoticity implies autoreducibility.

The restriction of the reduction allows us to show stronger negative results. We prove:

(iii) There exists a $\leq^{\log}_{\text{3-tt}}$-complete set for PSPACE that is not $\leq^{\log}_{\text{btt}}$-autoreducible.

This result is particularly interesting, since it shows that statement (i) does not hold for $\leq^{\log}_{\text{3-tt}}$ and that (ii) cannot be improved to $\leq^{\log}_{\text{3-tt}}$, unless one separates EXP from PSPACE. Furthermore, for logspace bounded-truth-table reducibility we obtain that resolving the autoreducibility or mitoticity of complete sets for classes between L and PSPACE in one or the other way implies new separations of complexity classes:

(iv) For every $\mathcal{C} \in \{\mathrm{P}, \mathrm{NP}, \Delta^{\mathrm{P}}_{\mathrm{k}}, \mathrm{PP}\}$ it holds that
- if all $\leq^{\log}_{\text{btt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{btt}}$-autoreducible, then $\mathcal{C} \neq$ PSPACE
- otherwise, $\mathrm{L} \neq \mathcal{C}$.

The paper is organized as follows. Section 2 contains the preliminary definitions and some basic propositions about autoreducibility and mitoticity. In section 3 we use search techniques in computation trees and self-reducibility to establish autoreducibility of complete sets for NL and P. In section 4 we use local checkability to

obtain further autoreducibility results for NL, P, and Δ_k^P. Moreover, we argue that some of those results are difficult to improve, as such an improvement would separate P or Δ_k^P from PSPACE. In section 5 we consider higher complexity classes such as PSPACE, EXP, NEXP and use diagonalization to obtain mitoticity and autoreducibility results, some of which again are hard or even impossible to improve.

2 Preliminaries

Let $\log 0 = 0$ and $\log n = \lceil \log_2 n \rceil$ for $n \geq 1$. A set is called *trivial* if it is finite or cofinite; otherwise the set is called *nontrivial*. The characteristic function of a set A is denoted by c_A or simply A. If M is a machine, then $M(x)$ denotes the computation of M on input x and $L(M)$ denotes the language accepted by M. Let $\langle \cdots \rangle$ be a standard pairing function computable in logarithmic space.

The operators $\wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow$ denote the usual 2-ary Boolean functions and $\neg\wedge, \neg\vee, \not\rightarrow, \not\leftarrow, \oplus$ denote the negations of these functions.

The notions of polynomial-time (resp., logspace) oracle Turing machine and polynomial-time-computable (resp., logspace-computable) function are defined according to Ladner, Lynch, and Selman [11,12]. These machines use a single write-only oracle query tape that is automatically erased after each query.

For sets A and B we say that A is polynomial-time Turing reducible to B ($A \leq_T^P B$), if there exists a polynomial-time oracle Turing machine that accepts A with B as its oracle. If M on input x asks at most $O(\log |x|)$ queries, then A is polynomial-time log-Turing reducible to B ($A \leq_{\text{log-T}}^P B$). If M's queries are nonadaptive (i.e., independent of the oracle), then A is polynomial-time truth-table reducible to B ($A \leq_{tt}^P B$). If M asks at most k nonadaptive queries, then A is polynomial-time k-truth-table reducible to B ($A \leq_{k\text{-tt}}^P B$). A is polynomial-time bounded-truth-table reducible to B ($A \leq_{btt}^P B$), if $A \leq_{k\text{-tt}}^P B$ for some k. A is polynomial-time disjunctive-truth-table reducible to B ($A \leq_{dtt}^P B$), if there exists a polynomial-time-computable function f such that for all x, $f(x) = (q_1, \ldots, q_n)$ for some $n \geq 1$ and $(x \in A \iff c_B(q_1) \vee \cdots c_B(q_n))$. If n is bounded by some constant k, then A is polynomial-time k-disjunctive-truth-table reducible to B ($A \leq_{k\text{-dtt}}^P B$). The polynomial-time conjunctive-truth-table reducibilities $\leq_{ctt}^P$ and $\leq_{k\text{-ctt}}^P$ are defined analogously. A is polynomial-time many-one reducible to B ($A \leq_m^P B$), if there exists a polynomial-time-computable function f such that $(x \in A \iff f(x) \in B)$. For a k-ary Boolean function α, A is polynomial-time α-truth-table reducible to a set B ($A \leq_{\alpha tt}^P B$), if there exists a polynomial-time-computable function f such that $f(x) = (q_1, \ldots, q_k)$ and $(x \in A \iff \alpha(c_B(q_1), \ldots, c_B(q_k)))$. We also use the following logspace reducibilities which are defined analogously in terms of logspace oracle Turing machines and logspace-computable functions: $\leq_T^{\log}$, $\leq_{\text{log-T}}^{\log}$, $\leq_{tt}^{\log}$, $\leq_{k\text{-tt}}^{\log}$, $\leq_{btt}^{\log}$, $\leq_{dtt}^{\log}$, $\leq_{k\text{-dtt}}^{\log}$, $\leq_{ctt}^{\log}$, $\leq_{k\text{-ctt}}^{\log}$, $\leq_m^{\log}$, $\leq_{\alpha tt}^{\log}$.

Consider a logspace oracle Turing machine M on input x. If we run through all configurations and compute the next string that is queried, we obtain a list L of all strings that are possibly queried during the computation of M on x. Now we can simulate the computation of M on x such that each time a string q is queried, we query *all* strings from L and use the answer to q in the further computation.

Table 1. Are all complete sets for certain classes autoreducible or even mititc? For each reduction $\leq$ in the first column and each class $\mathcal{C}$ in the first row, the corresponding cell shows whether all $\leq$-complete sets for $\mathcal{C}$ are autoreducible or mitotic, where M_y^x means $\leq_y^x$-mitotic, and A_y^x means $\leq_y^x$-autoreducible. From the first cell in the upper table we know for instance that all $\leq_{\mathrm{m}}^{\log}$-complete sets for NL are $\leq_{\text{1-tt}}^{\log}$-autoreducible. $k \geq 2$ is a fixed integer, α is an arbitrary binary boolean function. For the cells marked with X_1, X_2, and X_3, negative results are known: There is a $\leq_{\text{btt}}^{\log}$-complete set for PSPACE that is not $\leq_{\text{btt}}^{\log}$-autoreducible (Theorem 12) and a $\leq_{\text{btt}}^{\log}$-complete set for EXP that is not $\leq_{\text{btt}}^{\mathrm{p}}$-autoreducible [5]. Results implied by universal relations between reductions are omitted. For the definitions of the reductions and the classes, see section 2.

reduction	NL	P	$\Delta_{\mathrm{k}}^{\mathrm{P}}$	PSPACE	EXP	NEXP	references
$\leq_{\mathrm{m}}^{\log}$	$\mathrm{A}_{\text{1-tt}}^{\log}$, $\mathrm{A}_{\text{2-dtt}}^{\log}$, $\mathrm{A}_{\text{2-ctt}}^{\log}$	$\mathrm{A}_{\text{1-tt}}^{\log}$		$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{A}_{\mathrm{m}}^{\log}$	5, 9, 20, 23
$\leq_{\text{1-tt}}^{\log}$	$\mathrm{A}_{\text{2-tt}}^{\log}$	$\mathrm{A}_{\text{2-tt}}^{\log}$		$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{A}_{\mathrm{m}}^{\log}$	9, 29
$\leq_{\alpha\text{tt}}^{\log}$	$\mathrm{A}_{\text{btt}}^{\log}$	$\mathrm{A}_{\text{btt}}^{\log}$					17
$\leq_{\text{2-tt}}^{\log}$				$\mathrm{M}_{\text{2-tt}}^{\log}$	$\mathrm{M}_{\text{2-tt}}^{\log}$	$\mathrm{A}_{\text{2-tt}}^{\log}$	24, 27
$\leq_{k\text{-ctt}}^{\log}$	$\mathrm{A}_{k\text{-tt}}^{\log}$, $\mathrm{A}_{2k\text{-ctt}}^{\log}$	$\mathrm{A}_{k\text{-tt}}^{\log}$		$\mathrm{M}_{k\text{-ctt}}^{\log}$	$\mathrm{M}_{k\text{-ctt}}^{\log}$	$\mathrm{A}_{k\text{-ctt}}^{\log}$	5, 9, 27
$\leq_{k\text{-dtt}}^{\log}$	$\mathrm{A}_{k\text{-tt}}^{\log}$, $\mathrm{A}_{2k\text{-dtt}}^{\log}$	$\mathrm{A}_{k\text{-tt}}^{\log}$		$\mathrm{M}_{k\text{-dtt}}^{\log}$	$\mathrm{M}_{k\text{-dtt}}^{\log}$	$\mathrm{A}_{k\text{-dtt}}^{\log}$	5, 9, 27
$\leq_{\text{btt}}^{\log}$	$\mathrm{A}_{\text{log-T}}^{\log}$	$\mathrm{A}_{\text{log-T}}^{\log}$	$\mathrm{A}_{\text{log-T}}^{\log}$	X_1	X_2		11, 12, 15, [5]
$\leq_{\text{ctt}}^{\log}$	$\mathrm{A}_{\text{ctt}}^{\log}$			$\mathrm{M}_{\text{ctt}}^{\log}$	$\mathrm{M}_{\text{ctt}}^{\log}$	$\mathrm{A}_{\text{ctt}}^{\log}$	9, 23, 27
$\leq_{\text{dtt}}^{\log}$	$\mathrm{A}_{\text{dtt}}^{\log}$			$\mathrm{M}_{\text{dtt}}^{\log}$	$\mathrm{M}_{\text{dtt}}^{\log}$	$\mathrm{A}_{\text{dtt}}^{\log}$	9, 23, 27
$\leq_{\text{tt}}^{\log}$	$\mathrm{A}_{\text{tt}}^{\log}$	$\mathrm{A}_{\text{tt}}^{\log}$	$\mathrm{A}_{\text{tt}}^{\log}$				9, 15
$\leq_{\mathrm{T}}^{\log}$	$\mathrm{A}_{\text{tt}}^{\log}$	$\mathrm{A}_{\text{tt}}^{\log}$	$\mathrm{A}_{\text{tt}}^{\log}$				9, 15

reduction	NP	$\Delta_{\mathrm{k}}^{\mathrm{P}}$	PSPACE	EXP	NEXP	references
$\leq_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	[3,6,7,8]
$\leq_{\text{1-tt}}^{\mathrm{p}}$	$\mathrm{M}_{\text{1-tt}}^{\mathrm{p}}$		$\mathrm{M}_{\text{1-tt}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	[4,7,8,10], 29
$\leq_{\text{2-tt}}^{\mathrm{p}}$				$\mathrm{M}_{\text{2-tt}}^{\mathrm{p}}$	$\mathrm{A}_{\text{2-tt}}^{\mathrm{p}}$	[5], 24, 27
$\leq_{k\text{-ctt}}^{\mathrm{p}}$				$\mathrm{M}_{k\text{-ctt}}^{\mathrm{p}}$	$\mathrm{A}_{k\text{-ctt}}^{\mathrm{p}}$	23, 27
$\leq_{k\text{-dtt}}^{\mathrm{p}}$	$\mathrm{A}_{k\text{-dtt}}^{\mathrm{p}}$		$\mathrm{A}_{k\text{-dtt}}^{\mathrm{p}}$	$\mathrm{M}_{k\text{-dtt}}^{\mathrm{p}}$	$\mathrm{A}_{k\text{-dtt}}^{\mathrm{p}}$	[7], 23, 27
$\leq_{\text{btt}}^{\mathrm{p}}$				X_3		[5]
$\leq_{\text{ctt}}^{\mathrm{p}}$				$\mathrm{M}_{\text{ctt}}^{\mathrm{p}}$	$\mathrm{A}_{\text{ctt}}^{\mathrm{p}}$	23, 27
$\leq_{\text{dtt}}^{\mathrm{p}}$	$\mathrm{A}_{\text{dtt}}^{\mathrm{p}}$		$\mathrm{A}_{\text{dtt}}^{\mathrm{p}}$	$\mathrm{M}_{\text{dtt}}^{\mathrm{p}}$	$\mathrm{A}_{\text{dtt}}^{\mathrm{p}}$	[7], 23, 27
$\leq_{\text{tt}}^{\mathrm{p}}$	$\mathrm{A}_{\text{tt}}^{\mathrm{BPP}}$	$\mathrm{A}_{\text{tt}}^{\mathrm{p}}$		$\mathrm{A}_{\text{tt}}^{\mathrm{BPP}}$		[2,5]
$\leq_{\mathrm{T}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{T}}^{\mathrm{p}}$		$\mathrm{A}_{\mathrm{T}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{T}}^{\mathrm{p}}$		[2,5]

This shows that we may assume that logspace oracle Turing machines query nonadaptively.

Proposition 1 ([11]). *$A \leq_{\mathrm{tt}}^{\log} B$ if and only if $A \leq_{\mathrm{T}}^{\log} B$.*

Definition 2 (autoreducibility). *For $\leq \in \{\leq_{\mathrm{T}}^{\mathrm{p}}, \leq_{\text{log-T}}^{\mathrm{p}}, \leq_{\mathrm{tt}}^{\mathrm{p}}, \leq_{k\text{-tt}}^{\mathrm{p}}, \leq_{\mathrm{btt}}^{\mathrm{p}}, \leq_{\mathrm{T}}^{\log}, \leq_{\text{log-T}}^{\log}, \leq_{\mathrm{tt}}^{\log}, \leq_{k\text{-tt}}^{\log}, \leq_{\mathrm{btt}}^{\log}\}$, a set A is $\leq$-autoreducible, if $A \leq A$ via an oracle Turing machine that on input x does not query x.*

For $\leq \in \{\leq_{\mathrm{dtt}}^{\mathrm{p}}, \leq_{k\text{-dtt}}^{\mathrm{p}}, \leq_{\mathrm{ctt}}^{\mathrm{p}}, \leq_{k\text{-ctt}}^{\mathrm{p}}, \leq_{\mathrm{m}}^{\mathrm{p}}, \leq_{\alpha\mathrm{tt}}^{\mathrm{p}}, \leq_{\mathrm{dtt}}^{\log}, \leq_{k\text{-dtt}}^{\log}, \leq_{\mathrm{ctt}}^{\log}, \leq_{k\text{-ctt}}^{\log}, \leq_{\mathrm{m}}^{\log}, \leq_{\alpha\mathrm{tt}}^{\log}\}$, a set A is $\leq$-autoreducible, if $A \leq A$ via a function f such that if $f(x) = (q_1, \ldots, q_n)$, then $x \notin \{q_1, \ldots, q_n\}$.

Definition 3 (mitoticity). *For any polynomial-time reducibility $\leq^{\mathrm{p}}$, a set A is $\leq^{\mathrm{p}}$-mitotic, if there exists a separator $S \in \mathrm{P}$ such that $A \equiv^{\mathrm{p}} A \cap S \equiv^{\mathrm{p}} A \cap \overline{S}$. Analogously we define mitoticity for logspace reducibilities for which the separator is chosen from* L.

Proposition 4. *For every reduction $\leq$ and every non-trivial set A, if A is $\leq$-mitotic, then A is $\leq$-autoreducible.*

Proposition 5. *Let $k \geq 1$ and let $\mathcal{C}$ be a complexity class closed under complementation. All $\leq_{\mathrm{m}}^{\log}$-complete sets for $\mathcal{C}$ are $\leq_{1\text{-tt}}^{\log}$-autoreducible and all $\leq_{k\text{-dtt}}^{\log}$- or $\leq_{k\text{-ctt}}^{\log}$-complete sets for $\mathcal{C}$ are $\leq_{k\text{-tt}}^{\log}$-autoreducible.*

3 Autoreducibility by Self-reducibility

We use the notion of self-reducibility to show the autoreducibility of complete sets. For NL and P there exist self-reducible, $\leq_{\mathrm{m}}^{\log}$-complete sets. In this section we argue that this implies that all complete sets for NL and P are autoreducible (not only for $\leq_{\mathrm{m}}^{\log}$, but also several other logspace reducibility notions). For example, we obtain that all $\leq_{\mathrm{tt}}^{\log}$-complete sets for NL and P are $\leq_{\mathrm{tt}}^{\log}$-autoreducible.

The following notion of $\leq_{\mathrm{T}}^{\log}$-self-reducibility is a restriction of $\leq_{\mathrm{T}}^{\log}$-autoreducibility which demands that oracle queries have a certain structure.

Definition 6 ([1]). *A is $\leq_{\mathrm{T}}^{\log}$-self-reducible if there is a logspace oracle Turing machine M that accepts A with oracle A such that on input x, the queries asked by M are of the same length as x, lexicographically smaller than x, and differ from x at most in the last $\log |x|$ symbols.*

The notions of $\leq_{\mathrm{tt}}^{\log}$-self-reducibility and $\leq_{k\text{-tt}}^{\log}$-self-reducibility are defined analogously. By Proposition 1, a set is $\leq_{\mathrm{T}}^{\log}$-self-reducible if and only if it is $\leq_{\mathrm{tt}}^{\log}$-self-reducible.

There is a technical difficulty in defining self-reducibility for disjunctive and conjunctive truth-table reducibilities. In these cases, the reduction cannot simply accept or reject, but must generate queries that represent the answer. However, a self-reduction on input $x = y0^{|y|}$ is not allowed to make any query, since the

last $\log|x|$ symbols of x already reached the minimal possible value. Therefore, in the definition below the self-reduction may accept or reject without asking any queries.

Definition 7. *A set A is $\leq^{\log}_{\text{dtt}}$-self-reducible if there is a logspace-computable function f whose values can be 0, 1, or a list of words $(y_1, \ldots y_n)$ where $n \geq 1$ such that the following holds: If $f(x) \in \{0,1\}$, then $c_A(x) = f(x)$. Otherwise, it holds that $f(x) = (y_1, \ldots y_n)$ such that the y_i are of the same length as x, are lexicographically smaller than x, differ from x at most in the last $\log|x|$ symbols, and $x \in A \Leftrightarrow (c_A(y_1) \vee \cdots \vee c_A(y_n))$. If n is bounded by some constant k, then A is $\leq^{\log}_{k\text{-dtt}}$-self-reducible. The notions of $\leq^{\log}_{\text{ctt}}$-self-reducibility and $\leq^{\log}_{k\text{-ctt}}$-self-reducibility are defined analogously. A is $\leq^{\log}_{\text{m}}$-self-reducible if it is $\leq^{\log}_{1\text{-dtt}}$-self-reducibile.*

Each nontrivial self-reducible set B is autoreducible. The lemma below says that if a set A is in some sense equivalent to a self-reducible set B, then also A is autoreducible. The proof for the easiest case of $\leq^{\log}_{\text{dtt}}$ works by first executing the reduction $A \leq^{\log}_{\text{m}} B$ and then it iteratively follows exactly one of the self-reducibility-queries of B. For each of these queries, the $\leq^{\log}_{\text{dtt}}$-reduction to A is computed. If x does not occur among the queries of this reduction, we can complete the reduction. Otherwise, we continue the self-reduction on this path, as it positively depends on whether $x \in A$.

Lemma 8. *Let $l \geq 1$ and A, B be sets.*

1. $A \leq^{\log}_{\text{m}} B \leq^{\log}_{\text{tt}} A$ *and* B *is* $\leq^{\log}_{\text{tt}}$*-self-reducible* $\implies$ A *is* $\leq^{\log}_{\text{tt}}$*-autoreducible.*
2. $A \leq^{\log}_{\text{m}} B \leq^{\log}_{1\text{-tt}} A$ *and* B *is* $\leq^{\log}_{2\text{-tt}}$*-self-reducible* $\implies$ A *is* $\leq^{\log}_{2\text{-tt}}$*-autoreducible.*
3. $A \leq^{\log}_{\text{m}} B \leq^{\log}_{\text{dtt}} A$ *and* B *is* $\leq^{\log}_{\text{dtt}}$*-self-reducible* $\implies$ A *is* $\leq^{\log}_{\text{dtt}}$*-autoreducible.*
4. $A \leq^{\log}_{\text{m}} B \leq^{\log}_{l\text{-dtt}} A$, B *is* $\leq^{\log}_{2\text{-dtt}}$*-self-reducible* $\implies$ A *is* $\leq^{\log}_{2l\text{-dtt}}$*-autoreducible.*

A suitable encoding of $\{(C, g) \mid \text{gate } g \text{ in circuit } C \text{ computes } 1\}$ is $\leq^{\log}_{\text{m}}$-complete for P and $\leq^{\log}_{\text{tt}}$-self-reducible. By Lemma 8 this shows that all $\leq^{\log}_{\text{m}}$-complete sets for P are $\leq^{\log}_{\text{tt}}$-autoreducible. Restrictions of the circuit type yield similar results for other classes and reductions.

Theorem 9. *Let $k \geq 1$.*

1. *All $\leq^{\log}_{\text{tt}}$-complete sets for* NL, P, AC^k, SAC^k, NC^k *are $\leq^{\log}_{\text{tt}}$-autoreducible.*
2. *All $\leq^{\log}_{1\text{-tt}}$-complete sets for* NL, P *are $\leq^{\log}_{2\text{-tt}}$-autoreducible.*
3. *All $\leq^{\log}_{\text{dtt}}$-complete sets for* NL *are $\leq^{\log}_{\text{dtt}}$-autoreducible.*
4. *All $\leq^{\log}_{\text{ctt}}$-complete sets for* NL *are $\leq^{\log}_{\text{ctt}}$-autoreducible.*
5. *All $\leq^{\log}_{k\text{-dtt}}$-complete sets for* NL *are $\leq^{\log}_{2k\text{-dtt}}$-autoreducible.*
6. *All $\leq^{\log}_{k\text{-ctt}}$-complete sets for* NL *are $\leq^{\log}_{2k\text{-ctt}}$-autoreducible.*
7. *All $\leq^{\log}_{\text{m}}$-complete sets for* NL *are $\leq^{\log}_{2\text{-dtt}}$- and $\leq^{\log}_{2\text{-ctt}}$-autoreducible.*

We now use self-reducibility to show that for restricted $\leq^{\log}_{2\text{-tt}}$ reducibility it holds that complete sets for NL and P are autoreducible.

Theorem 10. *All sets that are $\leq^{\log}_{(\rightarrow)\text{tt}}$-complete for* NL *(resp.,* P*) are $\leq^{\log}_{\text{btt}}$-autoreducible.*

4 Autoreducibility by Local Checkability of Computations

If we represent computations of NL, P, and Δ_k^P machines in tableaus or configuration graphs, we can locally check the consistency of these computations. This technique allows us to show

(i) the $\leq^{\log}_{\text{log-T}}$-autoreducibility of all $\leq^{\log}_{\text{btt}}$-complete sets for NL, P, and Δ_k^P, and
(ii) the $\leq^{\log}_{\text{btt}}$-autoreducibility of all $\leq^{\log}_{\alpha\text{tt}}$-complete sets for NL and P, for all 2-ary Boolean α.

Using techniques by Buhrman et al. [5] we show that not all $\leq^{\log}_{\text{btt}}$-complete sets for PSPACE are $\leq^{\log}_{\text{btt}}$-autoreducible. Hence certain improvements of (i) and (ii) are difficult to obtain: Improving (i) to $\leq^{\log}_{\text{btt}}$-autoreducibility for P (resp., Δ_k^P) implies P $\neq$ PSPACE (resp., $\Delta_k^P \neq$ PSPACE), and improving (ii) to 3-ary Boolean α for P implies P $\neq$ PSPACE.

Moreover, we obtain that resolving the $\leq^{\log}_{\text{btt}}$-autoreducibility of $\leq^{\log}_{\text{btt}}$-complete sets for P, NP, PP, Δ_k^P, Σ_k^P, or Π_k^P leads to unknown separations of complexity classes.

Theorem 11. *1. All $\leq^{\log}_{\text{btt}}$-complete sets for* NL *are $\leq^{\log}_{\text{log-T}}$-autoreducible.*
2. All $\leq^{\log}_{\text{btt}}$-complete sets for P *are $\leq^{\log}_{\text{log-T}}$-autoreducible.*

Proof (Sketch for 2.). Let A be $\leq^{\log}_{\text{btt}}$-complete for P. Computing a bit in the computation tableau of the machine for A on input x can be done in polynomial time and thus is $\leq^{\log}_{\text{btt}}$-reducible to A. Fixing the answer for the query x in this reduction to 0 or 1 leads to two tableaus of which at least one is correct. For simplicity, we assume that they are not equal. Computing the index to the first difference in the two tableaus is again a polynomial task and thus reducible to A with at most logarithmically many queries (the length of the index is at most logarithmic). Let i be the index obtained from the reduction when queries to x are always answered by 1. If the tableaus are equal at index i, we reject (since the assumption $x \in A$ was incorrect). Otherwise, we accept if and only if there is a local inconsistency at index i in the tableau computed when answering queries to x by 0. If $x \notin A$, there will be no inconsistency and otherwise, the index i is obtained from the right assumption and the inconsistency will be found. □

Theorem 11 raises the question of whether one can improve this result to obtain a *non-adaptive* autoreduction, especially because only a constant number of queries in the proof is adaptive. The next theorem and its corollary show that at least in the case of P (Theorem 11.2) such an improvement is difficult to obtain as it separates P from PSPACE. The proof is based on an idea by Buhrman et al. [5] who show that not all $\leq^{\text{p}}_{\text{btt}}$-complete sets for EXP are $\leq^{\text{p}}_{\text{btt}}$-autoreducible. Note that $\leq^{\log}_{\text{2-T}}$-reducibility implies $\leq^{\log}_{\text{3-tt}}$-reducibility.

Theorem 12. *For every k there is a $\leq^{\log}_{\text{2-T}}$-complete set for* PSPACE *that is not $\leq^{\log}_{n^k\text{-tt}}$-autoreducible. In particular, not all $\leq^{\log}_{\text{btt}}$-complete sets for* PSPACE *are $\leq^{\log}_{\text{btt}}$-autoreducible.*

It follows that improving Theorem 11.2 to $\leq^{\log}_{\log\text{-tt}}$-autoreducibility or $\leq^{\log}_{n^c\text{-tt}}$-autoreducibility separates P from PSPACE. At this point we also observe two similar statements (13.2 and 13.3) which will explain the difficulty of improving the Theorems 15 and 17 below.

Corollary 13. *Let $c \geq 1$ and $k \geq 2$.*

1. *If all $\leq^{\log}_{\text{btt}}$-complete sets for* P *are $\leq^{\log}_{n^c\text{-tt}}$-autoreducible, then* P $\neq$ PSPACE.
2. *If all $\leq^{\log}_{3\text{-tt}}$-complete sets for* P *are $\leq^{\log}_{\text{btt}}$-autoreducible, then* P $\neq$ PSPACE.
3. *If all $\leq^{\log}_{\text{btt}}$-complete sets for $\Delta^{\text{P}}_{\text{k}}$ are $\leq^{\log}_{n^c\text{-tt}}$-autoreducible, then $\Delta^{\text{P}}_{\text{k}} \neq$* PSPACE.

Trivially, all $\leq^{\log}_{\text{btt}}$-complete sets for L are $\leq^{\log}_{\text{btt}}$-autoreducible and $\leq^{\log}_{\text{btt}}$-mitotic. This shows that proving or refuting Theorem 12 for class like P, NP, or $\Delta^{\text{P}}_{\text{k}}$ leads to unknown separations of complexity classes.

Corollary 14. *Let $k \geq 1$ and $\mathcal{C} \in \{\text{P}, \text{NP}, \text{PP}, \Delta^{\text{P}}_{\text{k}}, \Sigma^{\text{P}}_{\text{k}}, \Pi^{\text{P}}_{\text{k}}\}$.*

1. *If all $\leq^{\log}_{\text{btt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{btt}}$-autoreducible, then $\mathcal{C} \neq$* PSPACE*, otherwise $\mathcal{C} \neq$* L.
2. *If all $\leq^{\log}_{\text{btt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{btt}}$-mitotic, then $\mathcal{C} \neq$* PSPACE*, otherwise $\mathcal{C} \neq$* L.

Using another technique by Buhrman at al. [5] we generalize the proof of Theorem 11 and obtain similar results for all Δ-levels of the polynomial-time hierarchy. By Corollary 13.3, improving the theorem to $\leq^{\log}_{n^c\text{-tt}}$-autoreducibility or even $\leq^{\log}_{\text{btt}}$-autoreducibility separates $\Delta^{\text{P}}_{\text{k}}$ from PSPACE.

Theorem 15. *Let $k \geq 0$. All $\leq^{\log}_{\text{btt}}$-complete sets for $\Delta^{\text{P}}_{\text{k}}$ are $\leq^{\log}_{\log\text{-T}}$-autoreducible and all $\leq^{\log}_{\text{tt}}$-complete sets for $\Delta^{\text{P}}_{\text{k}}$ are $\leq^{\log}_{\text{tt}}$-autoreducible.*

In the proof of the next lemma we locally check configuration graphs of nondeterministic (resp., alternating) logspace machines to obtain the $\leq^{\log}_{\text{btt}}$-autoreducibility of $\leq^{\log}_{(\leftrightarrow)\text{tt}}$-complete sets for NL and P. Together with the results from previous sections this shows Theorem 17, which states that for every fixed 2-ary Boolean function α, all $\leq^{\log}_{\alpha\text{tt}}$-complete sets for NL and P are $\leq^{\log}_{\text{btt}}$-autoreducible. By Theorem 12, improving the theorem to 3-ary Boolean functions separates P from PSPACE.

Lemma 16. *All sets $\leq^{\log}_{(\leftrightarrow)\text{tt}}$-complete for* NL *or $\leq^{\log}_{(\leftrightarrow)\text{tt}}$-complete for* P *are $\leq^{\log}_{\text{btt}}$-autoreducible.*

Theorem 17. *For every $\alpha\colon \{0,1\}^2 \to \{0,1\}$, all $\leq^{\log}_{\alpha\text{tt}}$-complete sets for* NL *and all $\leq^{\log}_{\alpha\text{tt}}$-complete sets for* P *are $\leq^{\log}_{\text{btt}}$-autoreducible.*

5 Mitoticity and Autoreducibility by Diagonalization

We use diagonalization to obtain logspace mitoticity and autoreducibility results. Generally speaking, the diagonalization prevents difficult cases that could not be handled. Buhrman et al. [5] use this technique to show for EXP and other classes that all $\leq^{\mathrm{p}}_{\text{2-tt}}$-complete sets are $\leq^{\mathrm{p}}_{\text{2-tt}}$-autoreducible. We extend this technique and show the statement for NEXP.

Moreover, we consider several logspace reducibilities and obtain results for PSPACE, EXP, and NEXP, as those classes are powerful enough to diagonalize against logspace reductions. Since PSPACE and EXP are closed under complementation, we obtain that their many-one complete sets are mitotic. For NEXP we can only show that many-one complete sets are autoreducible.

5.1 Complete Sets for Classes Closed under Complementation

We will first show mitoticity and autoreducibility results for $\leq^{\log}_{\mathrm{m}}$-complete sets and for $\leq^{\log}_{\text{2-tt}}$-complete sets, that generally apply to classes that have certain closure properties. For this purpose, we first define length-restricted reductions computable in space $\log^2$.

Definition 18. *Let A and B be sets.*

1. *We define $A \leq^{\log^2\text{-lin}}_{\mathrm{m}} B$ similarly to $A \leq^{\log}_{\mathrm{m}} B$, except that, on input x, the computation of $f(x)$ is allowed to use $(\log|x|)^2$ space, but it must hold that $|f(x)| \leq c \cdot |x|$, where $c > 0$ is some constant.*
2. *We define $A \leq^{\log^2\text{-lin}}_{\text{1-tt}} B$ similarly to $A \leq^{\log}_{\text{1-tt}} B$, except that, on input x, the oracle machine is allowed to use $(\log|x|)^2$ space, but may only ask one oracle question of length at most $c \cdot |x|$, where $c > 0$ is some constant.*

We obtain the following results.

Theorem 19. *If a class $\mathcal{C}$ is closed under union, complement, and $\leq^{\log^2\text{-lin}}_{\mathrm{m}}$-reducibility, then all $\leq^{\log}_{\mathrm{m}}$-complete sets in $\mathcal{C}$ are $\leq^{\log}_{\mathrm{m}}$-mitotic.*

Corollary 20. *All $\leq^{\log}_{\mathrm{m}}$-complete sets for the following classes are $\leq^{\log}_{\mathrm{m}}$-mitotic:* $\mathrm{QP} = \mathrm{DTIME}(2^{\mathrm{polylog}(n)})$, PSPACE, EXP, REC, $\mathrm{DSPACE}(s)$ *and* $\mathrm{NSPACE}(s)$ *for all space-constructible $s \geq \log^2$.*

We adapt a technique by Buhrman et al. [5] to the logspace setting and obtain the following result.

Theorem 21. *If a class $\mathcal{C}$ is closed under $\leq^{\log^2\text{-lin}}_{\text{1-tt}}$-reducibility, then all $\leq^{\log}_{\text{2-tt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{2-tt}}$-autoreducible.*

Corollary 22. *All $\leq^{\log}_{\text{2-tt}}$-complete sets for the following classes are $\leq^{\log}_{\text{2-tt}}$-autoreducible:* $\mathrm{QP} = \mathrm{DTIME}(2^{\mathrm{polylog}(n)})$, PSPACE, EXP, REC, $\mathrm{DSPACE}(s)$ *and* $\mathrm{NSPACE}(s)$ *for all space-constructible $s \geq \log^2$.*

5.2 Complete Sets for NEXP

It is unknown whether the above results apply to NEXP, so here we cannot conclude mitoticity. We can, however, show that sets complete for NEXP are at least autoreducible.

Theorem 23. *1. For every $k \geq 1$ and $\leq \in \{\leq^{\mathrm{p}}_{k\text{-dtt}}, \leq^{\mathrm{p}}_{k\text{-ctt}}, \leq^{\mathrm{p}}_{\mathrm{dtt}}, \leq^{\mathrm{p}}_{\mathrm{ctt}}\}$, every $\leq$-complete set for* NEXP *is $\leq$-autoreducible.*
2. For every $k \geq 1$ and $\leq \in \{\leq^{\log}_{k\text{-dtt}}, \leq^{\log}_{k\text{-ctt}}, \leq^{\log}_{\mathrm{dtt}}, \leq^{\log}_{\mathrm{ctt}}\}$, every $\leq$-complete set for NEXP *is $\leq$-autoreducible.*

Proof (partial). In order to illustrate the diagonalization technique, we show the basic result for $\leq^{\mathrm{p}}_{\mathrm{dtt}}$ and NEXP. Let A be a $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete set for NEXP. Recall that $A \leq^{\mathrm{p}}_{\mathrm{dtt}} B \iff$ there exists a polynomial-time computable function f such that for all x, $f(x) = \langle q_1, \ldots, q_n\rangle$ and $(x \in A \iff B(q_1) \vee \cdots \vee B(q_n))$. Let $\{f_i\}_{i\geq 1}$ be an enumeration of all polynomial-time Turing transducers such that the computation of f_i on x can be simulated in time $|x|^i + i$. Let B be the set of inputs $\langle 0^i, x\rangle$ accepted by the following nondeterministic algorithm in exponential time.

1. $Q :=$ `set of all queries of` f_i `on input` $\langle 0^i, x\rangle$
2. `if` $x \notin Q$, `then accept` $\langle 0^i, x\rangle \iff x \in A$
3. `otherwise, reject` $\langle 0^i, x\rangle$

Obviously $B \in$ NEXP. So $B \leq^{\mathrm{p}}_{\mathrm{dtt}} A$ via some disjunctive truth-table reduction f_j. For every x, if x is one of the queries of $f_j(\langle 0^j, x\rangle)$, then, by the above algorithm, $\langle 0^j, x\rangle \notin B$. Hence for each query q of $f_j(\langle 0^j, x\rangle)$ we have $q \notin A$. In particular $x \notin A$. On the other hand, if $f_j(\langle 0^j, x\rangle) = \langle q_1, \ldots, q_m\rangle$ and $x \neq q_i$ for all i, then $x \in A \Leftrightarrow \langle 0^j, x\rangle \in B \Leftrightarrow c_A(q_1) \vee \cdots \vee c_A(q_m)$. Based on this observation, we obtain the following autoreduction for A, where x is the input.

1. $Q :=$ `set of all queries of` f_j `on input` $\langle 0^j, x\rangle$
2. `if` $x \notin Q$, `then return` $f_j(\langle 0^j, x\rangle)$
3. `otherwise, return some fixed value` $y \in \overline{A} - \{x\}$

□

Note that every non-trivial $\leq^{\log}_{1\text{-dtt}}$-autoreducible set is also $\leq^{\log}_{\mathrm{m}}$-autoreducible, and similarly every non-trivial $\leq^{\mathrm{p}}_{1\text{-dtt}}$-autoreducible set is also $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreducible, hence Theorem 23 covers $\leq^{\log}_{\mathrm{m}}$ and $\leq^{\mathrm{p}}_{\mathrm{m}}$ as a special case.

Theorem 24. *1. Every $\leq^{\mathrm{p}}_{2\text{-tt}}$-complete set for* NEXP *is $\leq^{\mathrm{p}}_{2\text{-tt}}$-autoreducible.*
2. Every $\leq^{\log}_{2\text{-tt}}$-complete set for NEXP *is $\leq^{\log}_{2\text{-tt}}$-autoreducible.*

Note that by Theorem 12 (resp., [5]), there exist $\leq^{\log}_{3\text{-tt}}$-complete sets for PSPACE (resp., $\leq^{\mathrm{p}}_{3\text{-tt}}$-complete sets for EXP) that are not $\leq^{\log}_{\mathrm{btt}}$-autoreducible (resp., $\leq^{\mathrm{p}}_{\mathrm{btt}}$-autoreducible), hence improving Theorem 24.1 to $\leq^{\mathrm{p}}_{3\text{-tt}}$ separates NEXP from EXP, and improving Theorem 24.2 to $\leq^{\log}_{3\text{-tt}}$ separates NEXP from PSPACE.

5.3 Complete Sets for PSPACE and EXP

We show that for some restricted polynomial-time truth-table reductions, complete sets for EXP are complete under length-increasing reductions, and the same holds considering logspace reductions for PSPACE and EXP. By carefully repeating the length-increasing reductions in such a way that we switch between stages defined by a separator set we obtain mitoticity for PSPACE and EXP.

Definition 25. *Given two sets A and B, we define $A \leq^{\mathrm{p}}_{\text{T-li}} B$ if there is a Turing machine M such that $A = L(M^B)$ and all queries made by $M^B(x)$ are of length strictly greater than $|x|$. The notions $\leq^{\mathrm{p}}_{\text{2-tt-li}}$, $\leq^{\mathrm{p}}_{k\text{-ctt-li}}$, $\leq^{\mathrm{p}}_{k\text{-dtt-li}}$, $\leq^{\mathrm{p}}_{\text{dtt-li}}$, $\leq^{\mathrm{p}}_{\text{ctt-li}}$, $\leq^{\mathrm{p}}_{\text{m-li}}$ and $\leq^{\log}_{\text{T-li}}$, $\leq^{\log}_{\text{2-tt-li}}$, $\leq^{\log}_{k\text{-ctt-li}}$, $\leq^{\log}_{k\text{-dtt-li}}$, $\leq^{\log}_{\text{dtt-li}}$, $\leq^{\log}_{\text{ctt-li}}$, $\leq^{\log}_{\text{m-li}}$ are defined similarly.*

Berman [3] and Ganesan and Homer [6] show that all many-one complete sets for EXP are many-one length-increasing equivalent. In the following lemma, we generalize to show that it also holds for some certain polynomial-time reductions and logspace reductions under EXP and PSPACE.

Lemma 26. *1. For every $k \geq 2$ and $\leq \in \{\leq^{\mathrm{p}}_{\text{2-tt}}, \leq^{\mathrm{p}}_{k\text{-ctt}}, \leq^{\mathrm{p}}_{k\text{-dtt}}, \leq^{\mathrm{p}}_{\text{ctt}}, \leq^{\mathrm{p}}_{\text{dtt}}\}$, all $\leq$-complete sets for* EXP *are $\leq$-li equivalent.*
2. For every $k \geq 2$ and $\leq \in \{\leq^{\log}_{\text{2-tt}}, \leq^{\log}_{\text{k-ctt}}, \leq^{\log}_{\text{k-dtt}}, \leq^{\log}_{\text{ctt}}, \leq^{\log}_{\text{dtt}}\}$, all $\leq$-complete sets for EXP *(resp.,* PSPACE*) are $\leq$-li equivalent.*

Theorem 27. *1. For every $k \geq 2$ and $\leq \in \{\leq^{\mathrm{p}}_{\text{2-tt}}, \leq^{\mathrm{p}}_{k\text{-ctt}}, \leq^{\mathrm{p}}_{k\text{-dtt}}, \leq^{\mathrm{p}}_{\text{ctt}}, \leq^{\mathrm{p}}_{\text{dtt}}\}$, every $\leq$-complete set for* EXP *is $\leq$-mitotic.*
2. For every $k \geq 2$ and $\leq \in \{\leq^{\log}_{\text{2-tt}}, \leq^{\log}_{k\text{-ctt}}, \leq^{\log}_{k\text{-dtt}}, \leq^{\log}_{\text{ctt}}, \leq^{\log}_{\text{dtt}}\}$, every $\leq$-complete set for EXP *(resp.,* PSPACE*) is $\leq$-mitotic.*

Note that by Theorem 12 (resp., [5]), there exist $\leq^{\log}_{\text{3-tt}}$-complete sets for PSPACE (resp., $\leq^{\mathrm{p}}_{\text{3-tt}}$-complete sets for EXP) that are not $\leq^{\log}_{\text{btt}}$-autoreducible (resp., $\leq^{\mathrm{p}}_{\text{btt}}$-autoreducible), hence Theorem 27 cannot be improved to $\leq^{\log}_{\text{3-tt}}$ or $\leq^{\mathrm{p}}_{\text{3-tt}}$.

5.4 Complete Sets for 1-Truth-Table Reductions

Homer, Kurtz, and Royer [10] and Buhrman [4] showed that for EXP and NEXP, every $\leq^{\mathrm{p}}_{\text{1-tt}}$-complete set is also $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete. Their approach also applies to $\leq^{\log}_{\text{1-tt}}$-complete sets for PSPACE, EXP and NEXP, so we have the following theorem.

Theorem 28 ([4,10])

1. *All $\leq^{\log}_{\text{1-tt}}$-complete sets for* PSPACE *(resp.,* EXP, NEXP*) are $\leq^{\log}_{\mathrm{m}}$-complete for* PSPACE *(resp.,* EXP, NEXP*).*
2. *All $\leq^{\mathrm{p}}_{\text{1-tt}}$-complete sets for* EXP *(resp.,* NEXP*) are $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for* EXP *(resp.,* NEXP*).*

This means that most of the obtained results for many-one complete sets also hold for 1-truth-table complete sets. We obtain the following corollary.

Corollary 29. *1. All* $\leq^{\log}_{1\text{-tt}}$*-complete sets for* PSPACE, EXP *are* $\leq^{\log}_{m}$*-mitotic.*
2. All $\leq^{\log}_{1\text{-tt}}$*-complete sets for* NEXP *are* $\leq^{\log}_{m}$*-autoreducible.*
3. All $\leq^{p}_{1\text{-tt}}$*-complete sets for* EXP, NEXP *are* $\leq^{p}_{m}$*-mitotic.*

References

1. Balcázar, J.L.: Self-reducibility. Journal of Computer and System Sciences 41(3), 367–388 (1990)
2. Beigel, R., Feigenbaum, J.: On being incoherent without being very hard. Computational Complexity 2, 1–17 (1992)
3. Berman, L.: Polynomial Reducibilities and Complete Sets. PhD thesis, Cornell University, Ithaca, NY (1977)
4. Buhrman, H.: Resource Bounded Reductions. PhD thesis, University of Amsterdam (1993)
5. Buhrman, H., Fortnow, L., van Melkebeek, D., Torenvliet, L.: Separating complexity classes using autoreducibility. SIAM Journal on Computing 29(5), 1497–1520 (2000)
6. Ganesan, K., Homer, S.: Complete problems and strong polynomial reducibilities. SIAM Journal on Computing 21, 733–742 (1992)
7. Glaßer, C., Ogihara, M., Pavan, A., Selman, A.L., Zhang, L.: Autoreducibility, mitoticity, and immunity. Journal of Computer and System Sciences 73(5), 735–754 (2007)
8. Glaßer, C., Pavan, A., Selman, A.L., Zhang, L.: Splitting NP-complete sets. SIAM Journal on Computing (2008)
9. Glaßer, C., Nguyen, D.T., Reitwießner, C., Selman, A.L., Witek, M.: Autoreducibility of complete sets for log-space and polynomial-time reductions. Technical Report TR13-047, Electronic Colloquium on Computational Complexity (2013)
10. Homer, S., Kurtz, S.A., Royer, J.S.: On 1-truth-table-hard languages. Theoretical Computer Science 115(2), 383–389 (1993)
11. Ladner, R.E., Lynch, N.A.: Relativization of questions about log space computability. Mathematical Systems Theory 10, 19–32 (1976)
12. Ladner, R.E., Lynch, N.A., Selman, A.L.: A comparison of polynomial time reducibilities. Theoretical Computer Science 1, 103–123 (1975)

An Incremental Polynomial Time Algorithm to Enumerate All Minimal Edge Dominating Sets*

Petr A. Golovach[1], Pinar Heggernes[1], Dieter Kratsch[2], and Yngve Villanger[1]

[1] Department of Informatics, University of Bergen, Norway
{petr.golovach,pinar.heggernes,yngve.villanger}@ii.uib.no
[2] LITA, Université de Lorraine - Metz, France
kratsch@univ-metz.fr

Abstract. We show that all minimal edge dominating sets of a graph can be generated in incremental polynomial time. We present an algorithm that solves the equivalent problem of enumerating minimal (vertex) dominating sets of line graphs in incremental polynomial, and consequently output polynomial, time. Enumeration of minimal dominating sets in graphs has very recently been shown to be equivalent to enumeration of minimal transversals in hypergraphs. The question whether the minimal transversals of a hypergraph can be enumerated in output polynomial time is a fundamental and challenging question; it has been open for several decades and has triggered extensive research. To obtain our result, we present a flipping method to generate all minimal dominating sets of a graph. Its basic idea is to apply a flipping operation to a minimal dominating set D^* to generate minimal dominating sets D such that $G[D]$ contains more edges than $G[D^*]$. We show that the flipping method works efficiently on line graphs, resulting in an algorithm with delay $O(n^2m^2|\mathcal{L}|)$ between each pair of consecutively output minimal dominating sets, where n and m are the numbers of vertices and edges of the input graph, respectively, and $\mathcal{L}$ is the set of already generated minimal dominating sets. Furthermore, we are able to improve the delay to $O(n^2m|\mathcal{L}|)$ on line graphs of bipartite graphs. Finally we show that the flipping method is also efficient on graphs of large girth, resulting in an incremental polynomial time algorithm to enumerate the minimal dominating sets of graphs of girth at least 7.

1 Introduction

Enumerating, i.e., generating or listing, all vertex or edge subsets of a graph that satisfy a specified property plays a central role in graph algorithms; see e.g. [1, 2, 8–10, 16, 20–22, 24, 29, 30, 32]. Enumeration algorithms with running

* This work is supported by the European Research Council, the Research Council of Norway, and the French National Research Agency. Due the space restrictions, proofs of various claims are omitted. The complete version of the paper can be found on-line in [13].

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 485–496, 2013.

time that is polynomial in the size of the input plus the size of the output are called output polynomial time algorithms. For various enumeration problems it has been shown that no output polynomial time algorithm can exist unless P = NP [20, 22, 24]. A potentially better behavior than output polynomial time is achieved by so called incremental polynomial time algorithms, which means that the next set in the list of output sets is generated in time that is polynomial in the size of the input plus the size of the already generated part of the output. Incremental polynomial time immediately implies output polynomial time.

One of the most classical and widely studied enumeration problems is that of listing all minimal transversals of a hypergraph, i.e., minimal hitting sets of its set of hyperedges. This problem has applications in areas like database theory, machine learning, data mining, game theory, artificial intelligence, mathematical programming, and distributed systems; extensive lists of corresponding references are provided by e.g., Eiter and Gottlob [10], and Elbassioni, Makino, and Rauf [11]. Whether or not all minimal transversals of a hypergraph can be listed in output polynomial time has been identified as a fundamental challenge in a long list of seminal papers, e.g., [8–12, 16, 27], and it remains unresolved despite continuous attempts since the 1980's.

Recently Kanté, Limouzy, Mary, and Nourine [18] have proved that enumerating the minimal transversals of a hypergraph is equivalent to enumerating the minimal dominating sets of a graph. In particular, they show that an output polynomial time algorithm for enumerating minimal dominating sets in graphs implies an output polynomial time algorithm for enumerating minimal transversals in hypergraphs. Dominating sets form one of the best studied notions in computer science; the number of papers on domination in graphs is in the thousands, and several well known surveys and books are dedicated to the topic (see, e.g., [14]).

Given the importance of the hypergraph transversal enumeration problem and the failed attempts to resolve whether it can be solved in output polynomial time, efforts to identify tractable special cases have been highly appreciated [3, 4, 6–8, 10, 11, 25, 26]. The newly proved equivalence to domination allows for new ways to attack this long-standing open problem. In fact some results on output polynomial algorithms to enumerate minimal dominating sets in graphs already exist for graphs of bounded treewidth and of bounded clique-width [5], interval graphs [8], strongly chordal graphs [8], planar graphs [10], degenerate graphs [10], and split graphs [17].

In this paper we show that all minimal dominating sets of line graphs and of graphs of large girth can be enumerated in incremental polynomial time. More precisely, we give algorithms where the time delay between two consecutively generated minimal dominating sets is $O(n^2m^2|\mathcal{L}|)$ on line graphs, $O(n^2m|\mathcal{L}|)$ on line graphs of bipartite graphs, and $O(n^2m|\mathcal{L}|^2)$ on graphs of girth at least 7, where $\mathcal{L}$ is the set of already generated minimal dominating sets of an input graph on n vertices and m edges. Line graphs form one of the oldest and most studied graph classes [15, 23, 33] and they can be recognized in linear time [28]. Our results, in addition to proving tractability for two substantial cases of

the hypergraph transversal enumeration problem, imply incremental polynomial time enumeration of minimal edge dominating sets in *arbitrary* graphs. In particular, we obtain an algorithm with delay $O(m^6|\mathcal{L}|)$ to enumerate all minimal edge dominating sets of any graph on m edges, where $\mathcal{L}$ is the set of already generated edge dominating sets. For bipartite graphs, we are able to reduce the delay to $O(m^4|\mathcal{L}|)$.

Our algorithms are based on the supergraph technique for enumerating vertex subsets in graphs [2, 21, 29, 32]. As a central tool in our algorithms, we present a new *flipping* method to generate the out-neighbors of a node of the supergraph, in other words, to generate new minimal dominating sets from a parent dominating set. Given a minimal dominating set D^*, our flipping operation replaces an isolated vertex of $G[D^*]$ with a neighbor outside of D^*, and, if necessary, supplies the resulting set with additional vertices to obtain new minimal dominating sets D, such that $G[D]$ has more edges compared to $G[D^*]$. Each of our algorithms starts with enumerating all maximal independent sets of the input graph G using the algorithm of Johnson, Papadimitriou, and Yannakakis [16], which gives the initial set of minimal dominating sets. Then the flipping operation is applied to every appropriate minimal dominating set found, to find new minimal dominating sets inducing subgraphs with more edges. We show that on all graphs, the flipping method enables us to identify a unique parent for each minimal dominating set. On line graphs and graphs of girth at least 7, we are able to prove additional (different) properties of the parents, which allow us to obtain the desired running time on these graph classes.

In a very recent publication of their work that was simultaneous with and independent from our work, Kanté, Limouzy, Mary, and Nourine [19] give output polynomial time algorithms for enumerating the minimal dominating sets of line graphs and path graphs. Their method is completely different from ours, as they obtain their algorithms through proving that line graphs and path graphs have closed neighborhood hypergraphs of bounded conformality.

2 Definitions and Preliminary Results

As input graphs to our enumeration problem, we consider finite undirected graphs without loops or multiple edges. Given such a graph $G = (V, E)$, its vertex and edge sets, V and E, are also denoted by $V(G)$ and $E(G)$, respectively. The subgraph of G induced by a subset $U \subseteq V$ is denoted by $G[U]$. For a vertex v, we denote by $N(v)$ its *(open) neighborhood*, that is, the set of vertices that are adjacent to v. The *closed neighborhood* of v is the set $N(v) \cup \{v\}$, and it is denoted by $N[v]$. If $N(v) = \emptyset$ then v is *isolated*. For a set $U \subseteq V$, $N[U] = \cup_{v \in U} N[v]$, and $N(U) = N[U] \setminus U$. The *girth* $g(G)$ of a graph G is the length of a shortest cycle in G; if G has no cycles, then $g(G) = +\infty$. A set of vertices is a *clique* if it induces a complete subgraph of G. A clique is *maximal* if no proper superset of it is a clique.

Two edges in E are adjacent if they share an endpoint. The *line graph* $L(G)$ *of* G is the graph whose set of vertices is $E(G)$, such that two vertices e and e' of

$L(G)$ are adjacent if and only if e and e' are adjacent edges of G. A graph H is a *line graph* if H is isomorphic to $L(G)$ for some graph G. Equivalently, a graph is a line graph if its edges can be partitioned into maximal cliques such that no vertex lies in more than two maximal cliques. This implies in particular that the neighborhood of every vertex can be partitioned into at most two cliques. It is well known that line graphs do not have induced subgraphs isomorphic to $K_{1,3}$, also called a *claw*.

A vertex v *dominates* a vertex u if $u \in N(v)$; similarly v dominates a set of vertices U if $U \subseteq N[v]$. For two sets $D, U \subseteq V$, D dominates U if $U \subseteq N[D]$. A set of vertices D is a *dominating set* of $G = (V, E)$ if D dominates V. A dominating set is *minimal* if no proper subset of it is a dominating set. Let D be a dominating set of G, and let $v \in D$. Vertex u is a *private vertex*, or simply *private*, for vertex v (with respect to D) if u is dominated by v but is not nominated by $D \setminus \{v\}$. Clearly, D is a minimal dominating set if and only if each vertex of D has a private vertex. We denote by $P_D[v]$ the set of all private vertices for v. Notice that a vertex of D can be private for itself. Vertex u is a *private neighbor* of $v \in D$ if $u \in N(v) \cap P_D[v]$. The set of all private neighbors of v is denoted by $P_D(v)$. Note that $P_D[v] = P_D(v) \cup \{v\}$ if v is isolated in $G[D]$, and otherwise $P_D[v] = P_D(v)$.

A set of edges $A \subseteq E$ is an *edge dominating* set if each edge $e \in E$ is either in A or is adjacent to an edge in A. An edge dominating set is *minimal* if no proper subset of it is an edge dominating set. It is easy to see that A is a (minimal) edge dominating set of G if and only if A is a (minimal) dominating set of $L(G)$.

Let $\phi(X)$ be a property of a set of vertices or edges X of a graph, e.g., "X is a minimal dominating set". The *enumeration problem for property* $\phi(X)$ for a given graph G on n vertices and m edges asks for the set $\mathcal{C}$ of all subsets of vertices or edges X of G that satisfy $\phi(X)$. An *enumeration algorithm* is an algorithm that solves this problem, i.e., that lists the elements of $\mathcal{C}$ without repetitions. An enumeration algorithm $\mathcal{A}$ is said to be *output polynomial time* if there is a polynomial $p(x, y)$ such that all elements of $\mathcal{C}$ are listed in time bounded by $p((n + m), |\mathcal{C}|)$. Assume now that $X_1, \ldots, X_\ell$ are the elements of $\mathcal{C}$ enumerated in the order in which they are generated by $\mathcal{A}$. The *delay* of $\mathcal{A}$ is the maximum time $\mathcal{A}$ requires between outputting X_{i-1} and X_i, for $i \in \{1, \ldots, \ell\}$. Algorithm $\mathcal{A}$ is *incremental polynomial time* if there is a polynomial $p(x, i)$ such that for each $i \in \{1, \ldots, \ell\}$, X_i is generated in time bounded by $p((n + m), i)$. Finally, $\mathcal{A}$ is a *polynomial delay* algorithm if there is a polynomial $p(x)$ such that for each $i \in \{1, \ldots, \ell\}$, the delay between outputting X_{i-1} and X_i is at most $p(n + m)$.

A set of vertices $U \subseteq V$ is an *independent set* if no two vertices of U are adjacent in G, and an independent set is *maximal* if no proper superset of it is an independent set. The following observation is folklore.

Observation 1. *Every maximal independent set of a graph G is a minimal dominating set of G. Furthermore, the set of all maximal independent sets of G is exactly the set of all its minimal dominating sets D such that $G[D]$ has no edges.*

Theorem 1 ([16]). *All maximal independent sets of a graph with n vertices and m edges can be enumerated in lexicographic order with polynomial delay $O(n(m+n\log|\mathcal{I}|))$, where $\mathcal{I}$ is the set of already generated maximal independent sets.*

Let $v_1,\ldots,v_n$ be the vertices of a graph G. Suppose that D' is a dominating set of G. We say that a minimal dominating set D is obtained from D' by *greedy removal of vertices* (*with respect to order* $v_1,\ldots,v_n$) if we initially let $D=D'$, and then recursively apply the following rule: *If D is not minimal, then find a vertex v_i with the smallest index i such that $D\setminus\{v_i\}$ is a dominating set in G, and set $D=D\setminus\{v_i\}$*. Clearly, when we apply this rule, we never remove vertices of D' that have private neighbors.

Finally, give some definitions on directed graphs, as the supergraph technique that we use creates an auxiliary directed graph. To distinguish this graph from the input graph, we will call the vertices of a directed graph *nodes*. The edges of a directed graph have directions and are called *arcs*. An arc (u,v) has direction from node u to node v. The *out-neighbors* of a node u are all nodes v such that (u,v) is an arc. Similarly, the *in-neighbors* of a node v are all nodes u such that (u,v) is an arc. In this paper, an in-neighbor will sometimes be called a *parent*.

3 Enumeration by Flipping: The General Approach

In this section we describe the general scheme of our enumeration algorithms. Let G be a graph; we fix an (arbitrary) order of its vertices: $v_1,\ldots,v_n$. Observe that this order induces a lexicographic order on the set $2^{V(G)}$. Whenever greedy removal of vertices of a dominating set is performed further in the paper, it is done with respect to this ordering.

Let D be a minimal dominating set of G such that $G[D]$ has at least one edge uw. Then vertex $u\in D$ is dominated by vertex $w\in D$. Let $v\in P_D(u)$. Let $X_{uv}\subseteq P_D(u)\setminus N[v]$ be a maximal independent set in $G[P_D(u)\setminus N[v]]$ selected greedily with respect to ordering $v_1,\ldots,v_n$, i.e., we initially set $X_{uv}=\emptyset$ and then recursively include in X_{uv} the vertex of $P_D(u)\setminus(N[\{v\}\cup X_{uv}])$ with the smallest index as long as it is possible. Consider the set $D'=(D\setminus\{u\})\cup X_{uv}\cup\{v\}$. Notice that D' is a dominating set in G, since all vertices of $P_D(u)$ are dominated by $X_{uv}\cup\{v\}$. Let Z_{uv} be the set of vertices that are removed to ensure minimality, and let $D^*=((D\setminus\{u\})\cup X_{uv}\cup\{v\})\setminus Z_{uv}$.

Lemma 1. *The set D^* is a minimal dominating set in G such that $X_{uv}\cup\{v\}\subseteq D^*$, $|E(G[D^*])|<|E(G[D])|$ and v is an isolated vertex of $G[D^*]$.*

Our main tool, the *flipping* operation is exactly the *reverse* of how we generated D^* from D; i.e., it replaces an isolated vertex v of $G[D^*]$ with a neighbor u in G to obtain D. In particular, we are interested in all minimal dominating sets D that can be generated from D^* in this way.

Given D and D^* as defined above, we say that D^* is a *parent of D with respect to flipping u and v*. We say that D^* is a *parent* of D if there are vertices

$u, v \in V(G)$ such that D^* is a parent with respect to flipping u and v. It is important to note that each minimal dominating set D such that $E(G[D]) \neq \emptyset$ has a unique parent with respect to flipping of any vertices $u \in D \cap N[D \setminus \{u\}]$ and $v \in P_D(u)$, as both sets X_{uv} and Z_{uv} are lexicographically first sets selected by a greedy algorithm. Similarly, we say that D is a *child* of D^* (with respect to flipping u and v) if D^* is the parent of D (with respect to flipping u and v).

Assume that there is an enumeration algorithm $\mathcal{A}$ that, given a minimal dominating set D^* of a graph G such that $G[D^*]$ has isolated vertices, an isolated vertex v of $G[D^*]$, and a neighbor u of v in G, generates with polynomial delay a set of minimal dominating sets $\mathcal{D}$ with the property that $\mathcal{D}$ contains all minimal dominating sets D that are children of D^* with respect to flipping u and v. In this case we can enumerate all minimal dominating sets of the graph G with n vertices and m edges as follows.

Our method is a variant of the supergraph technique that has been applied for enumerating subsets with various properties in graphs [2, 21, 29, 32]. More precisely, we define a directed graph $\mathcal{G}$ whose nodes are minimal dominating sets of G, with an additional special node r, called the *root*, that has no in-neighbors. Recall that by Observation 1., maximal independent sets are minimal dominating sets, i.e., they are nodes of $\mathcal{G}$. We add an arc from the root r to every maximal independent set of G. For each minimal dominating set $D^* \in V(\mathcal{G})$, we add an arc from D^* to every minimal dominating set D such that $\mathcal{A}$ generates D from D^* for some choice of u and v.

Next we run Depth-First Search in $\mathcal{G}$ starting from r. Observe that we need not construct $\mathcal{G}$ explicitly to do this, as for each node $W \neq r$ of $\mathcal{G}$ we can use $\mathcal{A}$ to generate all out-neighbors of W, and we can generate the out-neighbors of r with polynomial delay by Theorem 1. Hence, we maintain a list $\mathcal{L}$ of minimal dominating sets of G sorted in lexicographic order that are already visited nodes of $\mathcal{G}$. Also we keep a stack $\mathcal{S}$ of records R_W for $W \in V(\mathcal{G})$ that are on the path from r to the current node of $\mathcal{G}$. These records are used to generate out-neighbors. The record R_r contains the last generated maximal independent set and the information that is necessary to proceed with the enumeration of maximal independent sets. Each of the records R_W, for $W \neq r$, contains the current choice of u and v, the last set D generated by $\mathcal{A}$ for the instance (W, u, v), and the information that is necessary for $\mathcal{A}$ to proceed with the enumeration.

Lemma 2. *Suppose that $\mathcal{A}$ generates the elements of $\mathcal{D}$ for a triple (D^*, u, v) with polynomial delay $O(p(n, m))$. Let $\mathcal{L}^*$ be the set of all minimal dominating sets. Then the algorithm described above enumerates all minimal dominating sets as follows:*

- *with delay $O((p(n, m) + n^2)m|\mathcal{L}|^2)$ and total running time $O((p(n, m) + n^2)m|\mathcal{L}^*|^2)$;*
- *if $|E(G[D])| > |E(G[D^*])|$ for every $D \in \mathcal{D}$, then the delay is $O((p(n, m) + n^2)m^2|\mathcal{L}|)$, and the total running time is $O((p(n, m) + n^2)m|\mathcal{L}^*|^2)$;*
- *if $\mathcal{D}$ contains only children of D^* with respect to flipping of u and v, then the delay is $O((p(n, m) + n^2)m|\mathcal{L}|)$, and the total running time is $O((p(n, m) + n^2)m|\mathcal{L}^*|)$.*

Proof. Recall that any minimal dominating set D with at least one edge has a parent D^* and $|E(G[D^*])| < |E(G[D])|$. Because $\mathcal{A}$ generates D from D^*, (D^*, D) is an arc in $\mathcal{G}$. It follows that for any minimal dominating set $D \in V(\mathcal{G})$ with at least one edge, there is a maximal independent set $I \in V(\mathcal{G})$ such that I and D are connected by a directed path in $\mathcal{G}$. As (r, I) is an arc in $\mathcal{G}$, D is reachable from r. We conclude that Depth-First Search visits, and thus enumerates all nodes of $\mathcal{G}$. It remains to evaluate the running time.

To get a new minimal dominating set, we consider the records in $\mathcal{S}$. For each record R_W for $W \neq r$, we have at most m possibilities for u and v to get a new set D. As soon as a new set is generated it is added to $\mathcal{L}$ unless it is already in $\mathcal{L}$. Hence, we generate at most $m|\mathcal{L}|$ sets for W in time $(p(n,m) + n^2)m|\mathcal{L}|$, as each set is generated with polynomial delay $O(p(n,m))$, and after its generation we immediately test whether or not it is already in $\mathcal{L}$, which takes $O(n \log |\mathcal{L}|) = O(n^2)$ time, because $|\mathcal{L}| \leq 2^n$. For R_r, we generate at most $|\mathcal{L}|$ sets. Because any isolated vertex of G belongs to every maximal independent set, each set is generated with delay $O(n'(m + n' \log |\mathcal{L}|))$, i.e., in time $O(n'(m + n'^2))$ by Theorem 1, where n' is the number of non-isolated vertices. As $n' \leq 2m$, these sets are generated in time $O(n^2 m|\mathcal{L}|)$. Since $|\mathcal{S}| \leq |\mathcal{L}|$, in time $O((p(n,m) + n^2)m|\mathcal{L}|^2)$ we either obtain a new minimal dominating set or conclude that the list of minimal dominating sets is exhausted.

To get the bound for the total running time, recall that Depth-First Search runs in time that is linear in $|E(\mathcal{G})|$. As for each arc we perform $O((p(n,m) + n^2)m)$ operations, the total running time is $O((p(n,m) + n^2)m|\mathcal{L}^*|^2)$.

If for every $D \in \mathcal{D}$, $|E(G[D])| > |E(G[D^*])|$, then the delay is less. To see this, we observe that the number of edges in any minimal dominating set is at most m. Hence, any directed path starting from r in $\mathcal{G}$ has length at most m and, therefore, $|\mathcal{S}| \leq m + 1$. By the same arguments as above, we get that in time $O((p(n,m) + n^2)m^2|\mathcal{L}|)$ we either obtain a new minimal dominating set or conclude that the list of minimal dominating sets is complete.

Assume finally that $\mathcal{D}$ contains only children of D^* with respect to flipping of u and v. Since each minimal dominating set D with $E(G[D]) \neq \emptyset$ has a unique parent with respect to flipping of any vertices $u \in D \cap N[D \setminus \{u\}]$ and $v \in P_D(u)$, each D has at most m parents. Hence, we generate at most $m|\mathcal{L}|$ sets until we obtain a new minimal dominating set or conclude that the list is exhausted. As to generate a set and check whether it is already listed we spend time $O(p(n,m) + n^2)$, the delay between two consecutive sets that are output is $O((p(n,m) + n^2)m|\mathcal{L}|)$ and the total running time is $O((p(n,m) + n^2)m|\mathcal{L}^*|)$. □

To be able to apply our method, we have to show how to construct an algorithm, like algorithm $\mathcal{A}$ above, that produces $\mathcal{D}$ with polynomial delay. We will use the following lemma for this purpose.

Lemma 3. *Let D be a child of D^* with respect to flipping u and v; $D^* = ((D \setminus \{u\}) \cup X_{uv} \cup \{v\}) \setminus Z_{uv}$. Then for every vertex $z \in Z_{uv}$, the following three statements are true:*

1. $z \notin N[X_{uv} \cup \{v\}]$,
2. *z is dominated by a vertex of* $D^* \setminus (X_{uv} \cup \{v\})$,
3. *there is a vertex* $x \in N[X_{uv} \cup \{v\}] \setminus N[u]$ *adjacent to* z *such that* $x \notin N[D^* \setminus (X_{uv} \cup \{v\})]$.

Furthermore, for every $x \in N[X_{uv} \cup \{v\}] \setminus N[u]$ *such that* $x \notin N[D^* \setminus (X_{uv} \cup \{v\})]$, *there is a vertex* $z \in Z_{uv}$ *such that* x *and* z *are adjacent.*

We use this lemma to construct an algorithm for generating $\mathcal{D}$. The idea is to generate $\mathcal{D}$ by considering all possible candidates for X_{uv} and Z_{uv}. It would be interesting to know whether this can be done efficiently in general. On line graphs and graphs of girth at least 7, we are able to prove additional properties of the parent minimal dominating sets which result in efficient algorithms for generating $\mathcal{D}$, as will be explained in the sections below.

4 Enumeration of Minimal Edge Dominating Sets

In this section we show that all minimal edge dominating sets of an *arbitrary* graph can be enumerated in incremental polynomial time. We achieve this by enumerating the minimal dominating sets in line graphs.

For line graphs, we construct an enumeration algorithm that, given a minimal dominating set D^* of a graph G such that $G[D^*]$ has isolated vertices, an isolated vertex v of $G[D^*]$, and a neighbor u of v in G, generates with polynomial delay a set of minimal dominating sets $\mathcal{D}$ that contains all children of D^* with respect to flipping u and v, and has the property that $|E(G[D])| > |E(G[D^*])|$, for every $D \in \mathcal{D}$.

This is possible because on line graphs we can prove additional properties of a parent in the flipping method. Let D be a minimal dominating set of a graph G such that $G[D]$ has at least one edge uw, and assume that $v \in P_D(u)$. Recall that D^* is defined by choosing a maximal independent set $X_{uv} \subseteq P_D(u) \setminus N[v]$ in $G[P_D(u) \setminus N[v]]$, then considering the set $D' = (D \setminus \{u\}) \cup X_{uv} \cup \{v\}$, and letting $D^* = D' \setminus Z_{uv}$ where $Z_{uv} \subseteq D \cap D'$.

Lemma 4. *If* G *is a line graph, then:*

- $X_{uv} = \emptyset$,
- *each vertex of* Z_{uv} *is adjacent to exactly one vertex of* $P_{D^*}(v) \setminus N[u]$,
- *each vertex of* $P_{D^*}(v) \setminus N[u]$ *is adjacent to exactly one vertex of* Z_{uv}.

Consider a line graph G with n vertices $v_1, \ldots, v_n$ and m edges. Let D^* be a minimal dominating set and let v be an isolated vertex of $G[D^*]$. Suppose that u is a neighbor of v. Let $\{x_1, \ldots, x_k\} = P_{D^*}(v) \setminus N[u]$. We construct minimal dominating sets from $(D^* \setminus \{v\}) \cup \{u\}$ by adding a set Z that contains a neighbor of each x_i from $N(x_i) \setminus N[v]$. Recall that the vertices $x_1, \ldots, x_k$ should be dominated by Z_{uv} for every child of D^* by Lemma 3, and by the same lemma each x_i is dominated by a vertex from $N(x_i) \setminus N[v]$.

Let $U = N[u] \cup (\bigcup_{i=1}^{k} (N[x_i] \setminus N[v]) \cup \{x_i\})$. We need the following observation that also will be used in the next section. To see its correctness, it is sufficient to notice that because G contains no claws, $N[x_i] \setminus N[v]$ is a clique.

Lemma 5. *For any choice of a set $Z = \{z_1, \ldots, z_k\}$ such that $z_i \in N(x_i) \setminus N[v]$ for $i \in \{1, \ldots, k\}$, U is dominated by $Z \cup \{u\}$.*

We want to ensure that by subsequent removal of vertices of $D^* \setminus \{v\}$ (which we do to guarantee minimality), the number of edges in the obtained minimal dominating set is not decreased. To do it, for each vertex $v_j \in V(G)$, we construct the sets of vertices R_j that cannot belong to Z_{uv} for any child D of D^*, where both D and D^* contain v_j. First, we set $R_j = \emptyset$ for every $v_j \notin D^* \setminus \{v\}$. Let v_j be a vertex of $D^* \setminus \{v\}$ that has a neighbor v_s such that either $v_s \in D^*$ or $v_s = u$. As G does not contain a claw, $K = N(v_j) \setminus N[v_s]$ is a clique. Then we set $R_j = K$ in this case. Notice that we can have several possibilities for v_s. In this case v_s is chosen arbitrary. For all other $v_j \in D^* \setminus \{v\}$, $R_j = \emptyset$. Denote by R the set $\cup_{j=1}^{n} R_j$. For each $i \in \{1, \ldots, k\}$, let

$$Z_i = \{z \in V | z \in N(D^* \setminus \{v\}) \cap (N(x_i) \setminus (N[v] \cup R)), N(z) \cap (P_{D^*}(v) \setminus N[u]) = \{x_i\}\}.$$

We generate a set $\mathcal{D}$ of minimal dominating sets as follows.

Case 1. If at least one of the following three conditions is fulfilled, then we set $\mathcal{D} = \emptyset$:

i) there is a vertex $x \in D^* \setminus \{v\}$ such that $N[x] \subseteq N[D^* \setminus \{v, x\}] \cup U$,
ii) $k \geq 1$ and there is an index $i \in \{1, \ldots, k\}$ such that $Z_i = \emptyset$,
iii) u is not adjacent to any vertex of $D^* \setminus \{v\}$ and $N(u) \cap (\cup_{i=1}^{k} Z_i) = \emptyset$.

Otherwise, we consider two other cases.

Case 2. If u is adjacent to a vertex of $D^* \setminus \{v\}$, then we consecutively construct all sets $Z = \{z_1, \ldots, z_k\}$ where $z_i \in Z_i$, for $1 \leq i \leq k$ (if $k = 0$, then $Z = \emptyset$). For each Z, we construct the set $D' = (D^* \setminus \{v\}) \cup \{u\} \cup Z$. Notice that D' is a dominating set as all vertices of $P_{D^*}[v]$ are dominated by D', but D' is not necessarily minimal. Hence, we construct a minimal dominating set D from D' by the greedy removal of vertices. The obtained set D is unique for a given set Z, and it is added to $\mathcal{D}$.

Recall that by the definition of the parent-child relation, u should be dominated by a vertex in a child. If u is not adjacent to a vertex of $D^* \setminus \{v\}$, it should be adjacent to at least one of the added vertices. This gives us the next case.

Case 3. If u is not adjacent to any vertex of $D^* \setminus \{v\}$, and $N(u) \cap (\bigcup_{i=1}^{k} Z_i) \neq \emptyset$, then we proceed as follows. Let j be the smallest index such that $N(u) \cap Z_j \neq \emptyset$, and let j' be the smallest index at least j such that $Z_{j'} \setminus N(u) = \emptyset$ ($j' = k$ if they are all non-empty). For each t starting from $t = j$ and continuing until $t = j'$, we do the following. If $N(u) \cap Z_t = \emptyset$ then we go to next step $t = t + 1$. Otherwise, for each $w \in N(u) \cap Z_t$, we consider all possible sets $Z = \{z_1, \ldots, z_{t-1}, z_{t+1}, \ldots, z_k\} \cup \{w\}$ such that $z_i \in Z_i \setminus N(u)$ for $1 \leq i \leq t-1$, and $z_i \in Z_i$ for $t+1 \leq i \leq k$. As above, for each such set Z, we construct the set $D' = (D^* \setminus \{v\}) \cup \{u\} \cup Z$ and then create a minimal dominating set D from D' by the greedy removal of vertices. The obtained set D is unique for a given set Z, and it is added to $\mathcal{D}$.

We summarize the properties of the above algorithm in the following lemma.

Lemma 6. *The set $\mathcal{D}$ is a set of minimal dominating sets such that $\mathcal{D}$ contains all children of D^* with respect to flipping u and v. Furthermore, $|E(G[D])| > |E(G[D^*])|$ for every $D \in \mathcal{D}$, and the elements of $\mathcal{D}$ are generated with delay $O(n+m)$.*

Combining Lemmas 2 and 6, we obtain the following theorem and corollary.

Theorem 2. *All minimal dominating sets of a line graph can be enumerated in incremental polynomial time. On input graphs with n vertices and m edges, the delay is $O(n^2m^2|\mathcal{L}|)$, and the total running time is $O(n^2m|\mathcal{L}^*|^2)$, where $\mathcal{L}$ is the set of already generated minimal dominating sets and $\mathcal{L}^*$ is the set of all minimal dominating sets.*

Corollary 1. *All minimal edge dominating sets of an arbitrary graph be enumerated in incremental polynomial time. On input graphs with m edges, the delay is $O(m^6|\mathcal{L}|)$ and the total running time is $O(m^4|\mathcal{L}^*|^2)$, where $\mathcal{L}$ is the set of already generated minimal edge dominating sets and $\mathcal{L}^*$ is the set of all minimal edge dominating sets.*

We can improve the dependence of the total running time on the size of the output if we restrict our attention to edge dominating sets of bipartite graphs. Again, we work on the equivalent problem of generating minimal dominating sets of line graphs of bipartite graphs.

Theorem 3. *All minimal dominating sets of the line graph of a bipartite graph can be enumerated in incremental polynomial time. On input graphs with n vertices and m edges, the delay is $O(n^2m|\mathcal{L}|)$, and the total running time is $O(n^2m|\mathcal{L}^*|)$, where $\mathcal{L}$ is the set of already generated minimal dominating sets, and $\mathcal{L}^*$ is the set of all minimal dominating sets.*

Corollary 2. *All minimal edge dominating sets of a bipartite graph edges can be enumerated in incremental polynomial time. On input graphs with m edges, the delay is $O(m^4|\mathcal{L}|)$, and the total running time is $O(m^4|\mathcal{L}^*|)$, where $\mathcal{L}$ is the set of already generated minimal dominating sets, and $\mathcal{L}^*$ is the set of all minimal edge dominating sets.*

5 Graphs of Large Girth and Concluding Remarks

On line graphs we were able to observe properties of the parent relation in addition to uniqueness, which made it possible to apply the flipping method and design efficient algorithms for enumerating the minimal dominating sets. As another application of the flipping method, we show that it also works on graphs of girth at least 7. To do this, we observe other desirable properties of the parent relation on this graph class. As a result, we obtain an algorithm that enumerates the minimal dominating sets of a graph of girth at least 7 with delay $O(n^2m|\mathcal{L}|^2)$.

To conclude, the flipping method that we have described in this paper has the property that each generated minimal dominating set has a unique parent. It would be very interesting to know whether this can be used to obtain output polynomial time algorithms for enumerating minimal dominating sets in general. For the algorithms that we have given in this paper, on the studied graph classes we were able to give additional properties of the parents to obtain the desired running times. Are there additional properties of parents in general graphs that can result in efficient algorithms?

As a first step towards resolving these questions, on which other graph classes can the flipping method be used to enumerate the minimal dominating sets in output polynomial time? Another interesting question is whether the minimal dominating sets of line graphs or graphs of large girth can be enumerated with polynomial delay.

References

1. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics 65, 21–46 (1996)
2. Boros, E., Elbassioni, K., Gurvich, V.: Transversal hypergraphs to perfect matchings in bipartite graphs: characterization and generation algorithms. Journal of Graph Theory 53, 209–232 (2006)
3. Boros, E., Gurvich, V., Hammer, P.L.: Dual subimplicants of positive boolean functions. Optimization Methods & Software 10, 147–156 (1998)
4. Boros, E., Hammer, P.L., Ibaraki, T., Kawakami, K.: Polynomial time recognition of 2-monotonic positive Boolean functions given by an oracle. SIAM Journal on Computing 26, 93–109 (1997)
5. Courcelle, B.: Linear delay enumeration and monadic second-order logic. Discrete Applied Mathematics 157, 2675–2700 (2009)
6. Domingo, C., Mishra, N., Pitt, L.: Efficient read-restricted monotone cnf/dnf dualization by learning with membership queries. Machine Learning 37, 89–110 (1999)
7. Eiter, T.: Exact transversal hypergraphs and application to Boolean μ-functions. Journal of Symbolic Computing 17, 215–225 (1994)
8. Eiter, T., Gottlob, G.: Identifying the minimal transversals of a hypergraph and related problems. SIAM Journal on Computing 24, 1278–1304 (1995)
9. Eiter, T., Gottlob, G.: Hypergraph transversal computation and related problems in logic and AI. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 549–564. Springer, Heidelberg (2002)
10. Eiter, T., Gottlob, G., Makino, K.: New results on monotone dualization and generating hypergraph transversals. SIAM Journal on Computing 32, 514–537 (2003) (Preliminary version in STOC 2002)
11. Schwikowski, B., Speckenmeyer, E.: On enumerating all minimal solutions of feedback problems. ESA 2009 117, 253–265 (2002)
12. Fredman, M.L., Khachiyan, L.: On the complexity of dualization of monotone disjunctive normal forms. Journal of Algorithms 21, 618–628 (1996)
13. Golovach, P.A., Heggernes, P., Kratsch, D., Villanger, Y.: Generating all minimal edge dominating sets with incremental-polynomial delay. CoRR, abs/1208.5345 (2012)

14. Haynes, T.W., Hedetniemi, S.T.: Domination in graphs. Marcel Dekker Inc., New York (1998)
15. Hemminger, R.L., Beineke, L.W.: Line graphs and line digraphs. In: Beineke, L.W., Wilson, R.J. (eds.) Selected Topics in Graph Theory, pp. 271–305. Academic Press (1978)
16. Johnson, S., Papadimitriou, C.H., Yannakakis, M.: On generating all maximal independent sets. Information Processing Letters 27, 119–123 (1988)
17. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: Enumeration of minimal dominating sets and variants. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 298–309. Springer, Heidelberg (2011)
18. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: On the enumeration of minimal dominating sets and related notions (submitted for journal publication), `http://www.isima.fr/~kante/research.php`
19. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: On the Neighbourhood Helly of some Graph Classes and Applications to the Enumeration of Minimal Dominating Sets. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 289–298. Springer, Heidelberg (2012)
20. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K.M., Gurvich, V.: Generating all vertices of a polyhedron is hard. Discrete & Computational Geometry 39, 174–190 (2008)
21. Khachiyan, L., Boros, L., Borys, K., Elbassioni, K.M., Gurvich, V., Makino, K.: Generating Cut Conjunctions in Graphs and Related Problems. lgorithmica 51, 239–263 (2008)
22. Khachiyan, L., Boros, E., Elbassioni, K.M., Gurvich, V.: On enumerating minimal dicuts and strongly connected subgraphs. Algorithmica 50, 159–172 (2008)
23. Krausz, J.: Démonstration nouvelle d'un théorème de Whitney sur les réseaux. Mat. Fiz. Lapok 50, 75–85 (1943)
24. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. SIAM Journal on Computing 9, 558–565 (1980)
25. Makino, K., Ibaraki, T.: The maximum latency and identification of positive Boolean functions. SIAM J. Comput. 26, 1363–1383 (1997)
26. Makino, K., Ibaraki, T.: A fast and simple algorithm for identifying 2-monotonic positive Boolean functions. Journal of Algorithms 26, 293–305 (1998)
27. Papadimitriou, C.: NP-completeness: A retrospective. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 2–6. Springer, Heidelberg (1997)
28. Roussopoulos, N.D.: A max $\{m, n\}$ algorithm for determining the graph H from its line graph G. Information Processing Letters 2, 108–112 (1973)
29. Schwikowski, B., Speckenmeyer, E.: On enumerating all minimal solutions of feedback problems. Discrete Applied Mathematics 117, 253–265 (2002)
30. Tarjan, R.E.: Enumeration of the elementary circuits of a directed graph. SIAM Journal on Computing 2, 211–216 (1973)
31. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all the maximal independent sets. SIAM Journal on Computing 6, 505–517 (1977)
32. Tsukiyama, S., Shirakawa, I., Ozaki, H., Ariyoshi, H.: An algorithm to enumerate all cutsets of a graph in linear time per cutset. Journal of the ACM 27, 619–632 (1980)
33. Whitney, H.: Congruent graphs and the connectivity of graphs. American Journal of Mathematics 54, 150–168 (1932)

Deciding the Winner of an Arbitrary Finite Poset Game Is PSPACE-Complete

Daniel Grier*

University of South Carolina
grierd@email.sc.edu

Abstract. A poset game is a two-player game played over a partially ordered set (poset) in which the players alternate choosing an element of the poset, removing it and all elements greater than it. The first player unable to select an element of the poset loses. Polynomial time algorithms exist for certain restricted classes of poset games, such as the game of Nim. However, until recently the complexity of arbitrary finite poset games was only known to exist somewhere between NC^1 and PSPACE. We resolve this discrepancy by showing that deciding the winner of an arbitrary finite poset game is PSPACE-complete. To this end, we give an explicit reduction from Node Kayles, a PSPACE-complete game in which players vie to chose an independent set in a graph.

1 Introduction

A partially ordered set, or poset, is a set of elements with a binary relation (denoted $\leq$) indicating the ordering of elements that is reflexive, transitive, and antisymmetric. A poset game is an impartial two-player game played over some poset. Each turn, a player selects an element of the poset, removing it and all elements greater than it. A player loses when faced with the empty set. Equivalently, the last player able to select an element wins. We will assume that the number of elements in the poset is finite, which ensures that the game will eventually end in such a manner.

Poset games have been studied in various forms since a complete analysis of the game of Nim was given in 1901 by C. Bouton [2]. Other poset games with explicit polynomial time strategies include Von Neumann's Hackendot [17] and impartial Hackenbush on trees [1]. The above games have no induced subposet of cardinality four that form an 'N'. In fact, it is shown in [4] that all N-free poset games can be solved in polynomial time.

However there are several other well-studied poset games played over specific structures with unknown complexity [8]. Perhaps the most popular is the game of Chomp, which was introduced by Gale in 1974 and is played on the cross product of two Nim stacks [10]. Work by Byrnes [3] shows that certain Chomp

* This work was funded in part by the South Carolina Honors College Science Undergraduate Research Funding Program. This work was also supported by the Barry M. Goldwater Scholarship.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 497–503, 2013.

positions exhibit periodic behavior, but a quick general solution still does not exist. In Subset Takeaway [11], introduced by Gale in 1982, the players take turns removing a set and all its supersets from a collection of sets. In Shuh's Game of Divisors [15], the players alternate removing a divisor of n and its multiples. In fact, both Chomp and Subset Takeaway are special cases of the Game of Divisors, with n the product of at most two primes and n square-free, respectively.

In this paper, we discuss the complexity of deciding the winner of an arbitrary finite poset game, which has remained a longstanding question in the attempt to classify the tractability of combinatorial games [8,9]. Let PG be the language consisting of poset games with a winning strategy for the first player. Let $\langle P, \leq \rangle \in PG$ and $P = \{p_1, p_2, \ldots, p_n\}$. We will assume that $\langle P, \leq \rangle$ is represented as input in some explicit manner, such as a 0-1 matrix A where $a_{ij} = 1$ iff $p_i \leq p_j$. Although not all matrices of this type describe a legitimate poset, it is easy to check the validity of a matrix representation of a poset in polynomial time.

In [13], Kalinich shows that PG is at least as hard as NC^1 under AC^0 reductions by creating a correspondence with boolean circuits. Weighted poset games, which are a generalization of poset games, were shown to be PSPACE-complete in [12]. That result, which uses a completely different technique than the one described in this paper, along with another proof in [16], clearly show that PG is in PSPACE. We show that PG is indeed PSPACE-complete.

In [14], Schaefer shows that the two-player game Node Kayles is PSPACE-complete. In Node Kayles, the players take turns removing a vertex and all neighbors of that vertex from a graph. The first player unable to move loses.

In Section 2 we will give two constructions that serve as the basis for a reduction from Node Kayles to PG. We will then give a variety of lemmas demonstrating the desirable properties of these constructions in Section 3. In Section 4 we will combine these lemmas to show that PG is PSPACE-complete.

2 Constructions

Below we will give two constructions, ψ and φ. When applied in succession, they reduce an instance of Node Kayles into an instance of PG such that the winning player is preserved. Let G be the class of finite simple graphs and P be the class of finite posets. For $g \in G$ we will write $g = (V, E)$ where V is the set of vertices and E is the set of edges. We will use K_n to denote the complete graph on n vertices.

2.1 ψ-Construction

Define $\psi : G \to G$ such that

- $|E|$ is odd $\implies \psi(g) = g \cup K_2 \cup K_2$
- $|E|$ is even $\implies \psi(g) = g \cup K_2 \cup K_4$

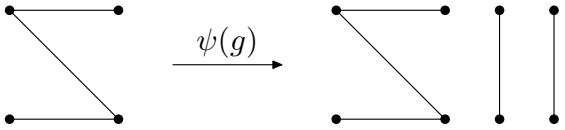


Fig. 1. Example of ψ-construction when $|E|$ is odd

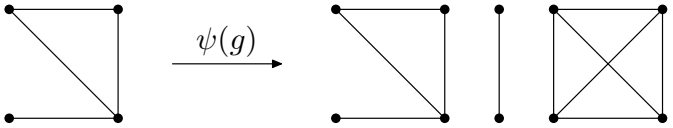


Fig. 2. Example of ψ-construction when $|E|$ is even

This construction serves two purposes. First, the edge cardinality of the resulting graph is always odd. Second, for every vertex, there is an edge that is not incident to it. It is also important to note that the winning player of the Node Kayles game does not change (see Lemma 1).

2.2 φ-Construction

Let $\varphi : G \to P$ be a function from simple graphs to posets, where $\varphi(g) = A \cup B \cup C$ is a three-level poset with disjoint levels A, B, and C from lowest to highest. That is, for any $a \in A$, $b \in B$, and $c \in C$, $b \not\leq a$, $c \not\leq b$, and $c \not\leq a$. Furthermore, any two elements on the same level are incomparable.

Fix $g = (V, E)$. The elements of the poset $\varphi(g)$ are as follows:

- The elements of C are the edges of g. That is, $C = E$.
- The elements of B are the vertices of g. That is, $B = V$.
- The elements of A are copies of the edges of g. To represent this, let $\gamma : C \to A$ be a 1-1 correspondence between the elements of C and the elements of A.

For each edge $e = (v_1, v_2)$ and $b \in B$, the $\leq$ relationship of the poset $\varphi(g)$ is as follows:

- $b \leq e$ iff $b = v_1$ or $b = v_2$. That is, e lies directly above its endpoints in B.
- $\gamma(e) \leq b$ iff $b \neq v_1$ and $b \neq v_2$. That is, $\gamma(e)$ is less than all the elements in B except the endpoints of e.

3 Lemmas

Lemma 1. *Player 1 wins the Node Kayles game on g iff Player 1 wins the Node Kayles game on $\psi(g)$.*

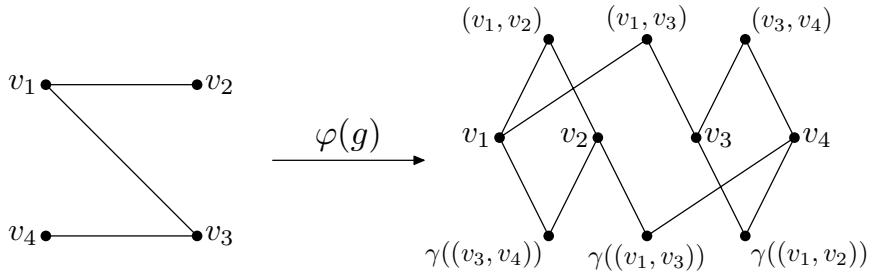


Fig. 3. Example φ-construction. Note that the left picture is an undirected graph representing a Node Kayles game, and the right picture is a Hasse Diagram representing the resultant poset game.

Proof. Suppose that the Node Kayles game played on g is a win for Player 1, who we will assume by convention is the first to play. We will show that this gives Player 1 an explicit winning strategy on $\psi(g)$. Player 1 first chooses the winning move in g. If Player 2 chooses a vertex in g, Player 1 can always respond with another move in g because Player 1 has the winning strategy on g. If Player 2 chooses a vertex in one of the complete graphs, Player 1 can respond with a vertex in the other complete graph, removing both complete graphs from consideration for the remainder of the game. Because Player 1 can respond to any move of Player 2, Player 1 will eventually win. Of course, this argument holds if Player 2 has the winning strategy in g, and similarly shows that a player has a winning strategy on g if he has a winning strategy on $\psi(g)$.

In terms of Sprague-Grundy theory, the disjoint union of the two complete graphs has Grundy number zero. Adding a game of Grundy number zero to an existing game does not change the winner of the original game [1]. In particular, the Grundy number of g is equal to the Grundy number of $\psi(g)$. □

Let $g = (V, E)$ be a finite simple graph and $e = (v_1, v_2)$ be an arbitrary edge in $\psi(g)$. For the following lemmas, assume that two players are playing the poset game on $\varphi(\psi(g))$. Also assume, for simplicity, that the players are Alice and Bob.

Lemma 2. *Assume no moves in A or C have yet been chosen. If both v_1 and v_2 have been chosen, then $\gamma(e)$ is a winning move.*

Proof. Because the ψ-construction always leaves a graph with an odd number of edges, choosing $\gamma(e)$ leaves an even number of incomparable points in A. □

Lemma 3. *Assume no moves in A or C have yet been chosen. If exactly one of v_1 and v_2 has been chosen, then $\gamma(e)$ is a losing move.*

Proof. First notice that e has already been removed from the poset because both $v_1 \leq e$ and $v_2 \leq e$. Because $\gamma(e) \not\leq v_1$ and $\gamma(e) \not\leq v_2$, choosing $\gamma(e)$ leaves a single point (either v_1 or v_2) in B. Thus, the next player can win by choosing the lone element in B, leaving an even number of incomparable points in A. □

Lemma 4. *Assume no moves in A or C have yet been chosen. If neither v_1 nor v_2 has been chosen, then both e and $\gamma(e)$ are losing moves.*

Proof. Assume that either player, say Alice, chooses $\gamma(e)$, which results in an even number of incomparable points in A, v_1 and v_2 in B, and e in C. Bob can then respond by choosing e. If Alice responds with v_1, then Bob can respond with v_2 (and vice versa), resulting in an even number of points in A, which is a win for Bob.

If, however, Alice responds with a point $a \in A$, there are three cases: $a \leq v_1$ and $a \leq v_2$, $a \leq v_1$ and $a \not\leq v_2$, or $a \leq v_2$ and $a \not\leq v_1$. Note that, by construction, there is no point a such that $a \not\leq v_1$ and $a \not\leq v_2$. That is, the only point that is not less than both v_1 and v_2 is $\gamma(e)$, which has already been taken by assumption. So first assume that $a \leq v_1$ and $a \leq v_2$. This would leave an odd number of elements in A, resulting in a win for Bob. Consider then that $a \leq v_1$ and $a \not\leq v_2$ or $a \leq v_2$ and $a \not\leq v_1$. Without loss of generality we can assume $a \leq v_1$ and $a \not\leq v_2$. Because $\psi(g)$ has at least two distinct components, each having at least one edge, there exists an edge e_2 that is not incident to either v_1 or v_2. By construction, $\gamma(e_2) \leq v_2$. Thus, Bob can choose $\gamma(e_2)$, leaving only an even number of elements in A, resulting in a win for Bob.

If Alice had initially chosen e instead of $\gamma(e)$, then Bob could have responded with $\gamma(e)$, which leads to the same game as played as above, which was a win for Bob. □

4 Main Theorem

Theorem 1. *PG is* PSPACE*-complete.*

Proof. It is straightforward to check and demonstrated explicitly in [16] that PG is in PSPACE. We will next give a reduction from Node Kayles to PG to show that the latter is also PSPACE-hard. First note that $\varphi(\psi(g))$ is computable in polynomial time.

We will argue inductively that Player 1 has a winning strategy for the poset game played on $\varphi(\psi(g))$ iff Player 1 has a winning strategy for the Nodes Kales game played on g. The idea behind the construction is that both players are forced to play elements in B until two elements v_1 and v_2 representing adjacent vertices in $\psi(g)$ have been chosen. At this point the following player can win by choosing the element $\gamma((v_1, v_2))$ in A.

Assume that the poset game played on $\varphi(\psi(g))$ has been played in the prescribed manner so far. That is, no elements from A or C have yet been chosen. Lemma 2 ensures that choosing a vertex neighboring a vertex that has already been chosen is a losing move. Lemma 3 and Lemma 4 ensure that choosing any point in A or C before two neighboring vertices have been chosen is a losing move. Thus, a player has a winning strategy on $\varphi(\psi(g))$ iff that player has a winning strategy on $\psi(g)$, since there is an obvious correspondence between the moves in $\varphi(\psi(g))$ and the moves in $\psi(g)$. Lemma 1 ensures that a player has a winning strategy on $\psi(g)$ iff he has a winning strategy on g. □

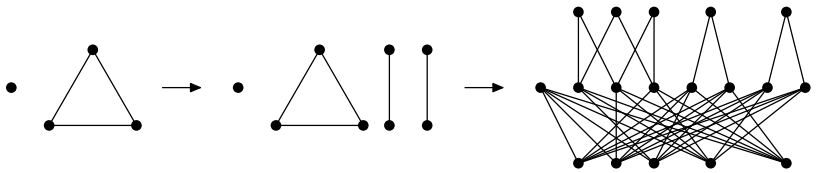


Fig. 4. Example of full reduction from g to $\psi(g)$ to $\varphi(\psi(g))$

5 Future Work

Using the above theorem, it follows easily that deciding the winner of a finite poset game with any height $k \geq 3$ is PSPACE-complete. In contrast, determining the winner of single-level poset games is trivially obtained by considering the parity of the poset elements. There are also polynomial time algorithms for some two-level poset games. In [7], Fraenkel and Aviezri give a polynomial time algorithm for finding the Grundy number of poset games played over a restricted class of two-level posets whose upper elements act like edges of a hypergraph. In [5], Fenner, Gurjar, Korwar, and Thierauf give a natural generalization of that algorithm and explore other possible avenues for finding the winner in polynomial time. However, neither of these results yield a general algorithm, and the complexity of two-level poset games remains an open problem.

This work has also spawned a new PSPACE-complete game on sets invented by Fenner and Fortnow [6]. Given a collection of finite sets $S_1, \ldots, S_k$, each player takes turns picking a non-empty set S_i, removing the elements of S_i from all the sets S_j. The player who empties all the sets wins. To reduce a poset game into an instance of set-game, simply take the sets as the upper cones of the poset. That is, each set consists of an element and all elements greater than it. However, if the cardinality of the sets is bounded, the complexity is still open.

Acknowledgements. I would like to thank Dr. Stephen Fenner for almost everything leading to this result. Perpetually busy, he still always finds the time to teach me and listen to my ideas. I am also very grateful for the support I received from the University of South Carolina Honors College and for all of those who helped me edit and refine this paper.

References

1. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for Your Mathematical Plays, vol. 1. AK Peters (2004)
2. Bouton, C.L.: Nim, a game with a complete mathematical theory. The Annals of Mathematics 3(1/4), 35–39 (1901)
3. Byrnes, S.: Poset game periodicity. Integers: Electronic Journal of Combinatorial Number Theory 3(G03), 2 (2003)
4. Deuber, W., Thomassé, S.: Grundy Sets of Partial Orders (1996)

5. Fenner, S., Gurjar, R., Korwar, A., Thierauf, T.: Two-level posets (2012) (manuscript)
6. Fortnow, L.: A simple PSPACE-complete problem, http://blog.computationalcomplexity.org/2012/11/a-simple-pspace-complete-problem.html
7. Fraenkel, A.S., Scheinerman, E.R.: A deletion game on hypergraphs. Discrete Applied Mathematics 30(2-3), 155–162 (1991)
8. Fraenkel, A.S.: Recent results and questions in combinatorial game complexities. Theoretical Computer Science 249(2), 265–288 (2000)
9. Fraenkel, A.S.: Complexity, appeal and challenges of combinatorial games. Theoretical Computer Science 313(3), 393–415 (2004)
10. Gale, D.: A curious Nim-type game. The American Mathematical Monthly 81(8), 876–879 (1974)
11. Gale, D., Neyman, A.: Nim-type games. International Journal of Game Theory 11(1), 17–20 (1982)
12. Ito, H., Takata, S.: PSPACE-completeness of the Weighted Poset Game (2011)
13. Kalinich, A.O.: Flipping the winner of a poset game. Information Processing Letters (2011)
14. Schaefer, T.J.: On the complexity of some two-person perfect-information games. Journal of Computer and System Sciences (1978)
15. Schuh, F.: Spel van delers (The game of divisors). Nieuw Tijdschrift voor Wiskunde 39, 299–304 (1952)
16. Soltys, M., Wilson, C.: On the complexity of computing winning strategies for finite poset games. Theory of Computing Systems 48(3), 680–692 (2011)
17. Úlehla, J.: A complete analysis of Von Neumann's Hackendot. International Journal of Game Theory 9(2), 107–113 (1980)

Dynamic Compressed Strings with Random Access

Roberto Grossi[1], Rajeev Raman[2], Satti Srinivasa Rao[3], and Rossano Venturini[1]

[1] Dipartimento di Informatica, Università di Pisa, Italy
[2] Department of Computer Science, University of Leicester, UK
[3] School of Computer Science and Engineering, Seoul National Univerity, Korea

Abstract. We consider the problem of storing a string S in dynamic compressed form, while permitting operations directly on the compressed representation of S: access a substring of S; replace, insert or delete a symbol in S; count how many occurrences of a given symbol appear in any given prefix of S (called rank operation) and locate the position of the ith occurrence of a symbol inside S (called select operation). We discuss the time complexity of several combinations of these operations along with the entropy space bounds of the corresponding compressed indexes. In this way, we extend or improve the bounds of previous work by Ferragina and Venturini [TCS, 2007], Jansson et al. [ICALP, 2012], and Nekrich and Navarro [SODA, 2013].

1 Introduction

The volume of unstructured, semi-structured, and replicated data, such as textual data, text with markup, and data backup and log analysis, has been growing much faster than structured (relational) data in recent years [IDC's "Digital Universe Survey", 2011]. Such non-relational data is commonly viewed as a (often highly compressible) string, and the processing of this data is not always amenable to external-memory solutions. Considerations like these, and the common practice of storing important data in the main memory of a computing cluster, have motivated the development of *compressed string storage schemes* [6,20,8], which store a string $S[1,n]$ from the alphabet $[\sigma] = \{1,\ldots,\sigma\}$ in compressed form, while allowing random access to the string via the operation:

$\mathsf{Access}(i,m)$**:** return the substring $S[i,i+m-1]$ for $m \geq 1$ and $1 \leq i \leq n-m+1$.

These schemes have been developed on the standard *RAM* model with a word size of $\Theta(\lg n)$ bits. They support Access optimally (as a substring of length $\ell = \Theta(\lg_\sigma n)$ fits in $O(1)$ words, $\mathsf{Access}(i,\ell)$ executes in $O(1)$ time) and aim to minimize the *space* in bits, expressed as the sum of two components that, when S is compressible, is smaller than the $n \lg \sigma$ bits used by S in 'raw' form:

- $nH_k(S)$ where $H_k(S)$ is the *k-th order empirical entropy* of S (a measure of compressibility, see the end of this section), and $k \geq 0$ is an integer, and
- the *redundancy*, or any additional space required to support random access.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 504–515, 2013.

Table 1. Summary of discussed results. Here $\ell = \Theta(\lg_\sigma n)$, $\rho = (k \lg \sigma + \lg\lg n)/\lg_\sigma n$, $\lambda = \lg n / \lg\lg n$, $\delta = \min\{\lg_\sigma n, (k+1)\lambda\}$, $\dagger$ = time to Access *one* symbol.

Access(i,ℓ)	Replace	Insert/Delete	Rank/Select	space (bits)	ref.
$O(1)$	—	—	—	$n(H_k + O(\rho))$	[6,20,8]
$O(1)$	$O(\delta)$	—	—	$n(H_k + O(\rho))$	[11]
$O(1)$	$O(1/\epsilon)$	—	—	$n(H_k + O(\epsilon(k+1)\lg\sigma + \rho))$	[11]
$\boldsymbol{O(1)}$	$\boldsymbol{O(1)}$	—	—	$\boldsymbol{n(H_k + O(\rho))}$	**Thm.1**
$O(\lambda)$	$O(\lambda)$	$O(\lambda)$	—	$n(H_0 + O(\lg\lg n/\lg_\sigma n))$	[11]
$O(\lambda)$	$O(\lambda)$	$O(\lambda)$	—	$n(H_k + O(\rho + k\lg\lg n/\lg n))$	[11]
$\boldsymbol{O(\lambda)}$	$\boldsymbol{O(\lambda)}$	$\boldsymbol{O(\lambda)}$	—	$\boldsymbol{n(H_k + O(\lg\lg n/\lg_\sigma n))}$	**Thm.2**
$O(\lambda)^\dagger$	$O(\lambda)$	$O(\lambda)$	$O(\lambda)$	$n(H_0 + O(1)) + O(\sigma(\lg\sigma + (\lg n)^{1+\epsilon}))$	[13]
$\boldsymbol{O(1)}$	$\boldsymbol{O(\lambda)}$	—	$\boldsymbol{O(\lambda)}$	$\boldsymbol{n(H_k + O(\lg\lg\sigma + \rho\lg\lg n))}$	**Cor.1**

The redundancy is a quantity of significant fundamental interest, particularly for lower bounds (see [17] and references therein), and is critical in practice. The best space upper bound is currently $n(H_k(S) + O(\rho(k,\sigma,n)))$ bits, which holds *simultaneously* for all $0 \le k \le \lg_\sigma n$, where $\rho(k,\sigma,n) = \frac{k\lg\sigma + \lg\lg n}{\lg_\sigma n}$. As k increases, the H_k term decreases, but ρ increases. However, so long as $k = o(\lg_\sigma n)$, $\rho = o(n\lg\sigma)$ is asymptotically smaller than S in 'raw' form. The data structure of [6] attains the above space bound and supports Access in $O(1)$ time. We now describe our contributions in the context of related work.

Dynamic Access*-only Sequences.* The storage schemes of [6,20,8] are all *static*, i.e., do not permit changes to S (see Table 1, first row). Although there has been work on storing dynamic sequences in a compressed format, the gap (in compression performance and Access time) with the static storage schemes remained large, until a recent breakthrough result by Jansson et al. [11]. They considered expanding the repertoire to include, in addition to Access:

Replace(i,c)**:** replace $S[i]$ by a symbol $c \in [\sigma]$.
Insert(i,c)**:** insert the symbol c into S between positions $i-1$ (if it exists) and i, and make S one symbol longer.
Delete(i)**:** delete $S[i]$ and make S one symbol shorter.

Jansson et al. gave two schemes (Table 1, rows 2 and 3) that achieve "high-order" compression and $O(1)$ time for Access(i,ℓ), but effectively do not support $O(1)$-time Replace. To achieve a redundancy of $O(\rho(k,\sigma,n))$, Replace takes $O(\min\{\lg_\sigma n, (k+1)\lg n/\lg\lg n\})$ time (Table 1, row 2). Unfortunately, the redundancy needed to get $O(1)$-time Replace (take $1/\epsilon = O(1)$ in row 3) is rather large: the first term is asymptotically larger even than S in 'raw' form, unless k is constant. Jansson et al. state that removing the first term while obtaining $O(1)$-time Access(i,ℓ) and Replace is an open question, which we resolve here.

– We give a representation whose space usage is $n(H_k(S) + O(\rho(k,\sigma,n)))$, which matches the redundancy of the best known static schemes [6,20,8], and supports Access(i,ℓ) and Replace in $O(1)$ time (Table 1, row 4). The data structure is simple and has potential to be practical.

Jansson et al. also gave a storage scheme (Table 1, rows 5 and 6) supporting all four operations in $O(\lg n/\lg\lg n)$ time, which is optimal [7].

– We give a scheme with space usage $n(H_k(S) + O(\lg\lg n/\lg_\sigma n))$ bits, for all $k \leq \frac{1}{8}\lg_\sigma n$, that supports all four operations in $O(\lg n/\lg\lg n)$ time (Table 1, row 7). While the redundancy of Jansson et al. is greater than that of the static schemes of [6,20,8], ours is *less*. Our redundancy has (surprisingly) no significant dependency on k: increasing k does not affect space usage.

Dynamic Rank/Select *Sequences.* In other recent work, Navarro and Nekrich [13] considered dynamic sequences that support further operations on S:

Rank(c, i) : return $|\{j \leq i | S[j] = c\}|$, for any $c \in [\sigma]$ and $1 \leq i \leq n$.
Select(c, i) : return the position of the i-th occurrence of c in S (-1 if there are fewer than i occurrences of c in S), for any $c \in [\sigma]$ and $i \geq 1$.

Rank and Select operations are fundamental components of a number of space-efficient data structures used in a variety of applications, such as indexing compressed XML documents [4] or compressed text [5,10]. Improving on a number of earlier papers, Navarro and Nekrich finally achieved an optimal [7] (amortized) time of $O(\lg n/\lg\lg n)$ (Table 1, second-to-last row). A key shortcoming is that they are unable to achieve "higher-order" entropy compression. Also, their Access operation costs $O(\lg n/\lg\lg n)$ time *per* symbol retrieved.

– We give a scheme that occupies $nH_k(S) + O(n(\lg\lg\sigma + (k\lg\sigma\lg\lg n)/\lg_\sigma n)$ bits and supports Access(i, ℓ) in $O(1)$ time, Rank and Select in $O(\lg n/\lg\lg n)$ time, and Replace in $O(\lg n/\lg\lg n)$ amortized time (Table 1, last row).

Note that a lower bound of $\Omega(\lg n/\lg\lg n)$ time holds for the above operations by a simple reduction from the Subset Rank problem as discussed in [19]. Compared to [13], we achieve higher-order entropy compression, which had not been achieved before for dynamic strings supporting Rank/Select and we can retrieve substrings of length ℓ quickly. However, our redundancy is worse and the Replace operation is weaker than a general Insert/Delete. We obtain this result by a technique that could be used in other problems. Compressed string storage schemes are the basis of an effective way of designing space-efficient data structures [1]: de-couple the storage of the data (S) from the *succinct indices* used to support operations. Our technique is to 'encapsulate' a *static* Rank/Select succinct index [9] within a 'dynamization' index, while updating S using the dynamic storage scheme of Theorem 1.

Empirical Entropy. Here we recall this notion. For each $c \in [\sigma]$, let $p_c = n_c/n$ be its empirical probability of occurring in S, where n_c is its number of occurrences. The *zero-th order empirical entropy* of S is defined as $H_0(S) = -\sum_{c=1}^{\sigma} p_c \lg p_c$. For any string w of length k, let w_S be the string of single symbols following the occurrences of w in S, taken from left to right. The kth order empirical entropy of S is defined as $H_k(S) = \frac{1}{n}\sum_{w\in[\sigma]^k} |w_S|\, H_0(w_S)$. Not surprisingly, for any $k \geq 0$ we have $H_k(S) \geq H_{k+1}(S)$. The value $nH_k(S)$ is a lower bound to the output size of any compressor that encodes each symbol of S only considering the symbol itself and the k immediately preceding symbols [12].

2 Entropy Bounds for Dynamic Storage of Strings

This section presents two main results on storing a dynamic compressed string.

Theorem 1. *Given a string $S[1,n]$ over the alphabet $\Sigma = [\sigma] = \{1,\ldots,\sigma\}$, there exists an index storing S that supports* Access(i,m) *in $O(1+\frac{m}{\lg_\sigma n})$ time and* Replace *in $O(1)$ time (bounds are worst-case and optimal). The overall space occupancy is $nH_k(S)+O(n\frac{k\lg\sigma+\lg\lg n}{\lg_\sigma n})$ bits for all $k=o(\lg_\sigma n)$.*

Theorem 2. *Given a string $S[1,n]$ over the alphabet $\Sigma=[\sigma]$, there exists an index storing S that supports* Access(i,m) *in $O(\frac{\lg n}{\lg\lg n}+\frac{m}{\lg_\sigma n})$ time, and* Replace, Insert, Delete *in $O(\frac{\lg n}{\lg\lg n})$ time (bounds are worst-case and optimal). The overall space occupancy is $nH_k(S)+O(n\frac{\lg\lg n}{\lg_\sigma n})$ bits for all $k\le\frac{1}{8}\lg_\sigma n$.*

2.1 Supporting Access and Replace (Theorem 1)

Squeezing the current string S into S_ℓ. At any time we conceptually represent S as a sequence S_ℓ of $n'=\lceil n/\ell\rceil$ macro-symbols over the macro-alphabet Σ^ℓ, where $\ell=\lceil\frac{1}{2}\lg_\sigma n\rceil$. Each macro-symbol is made up of ℓ consecutive symbols in S (namely, $S_\ell[i]=S[(i-1)\cdot\ell+1,\ldots,i\cdot\ell]$, for any $1\le i\le n'$) and thus the macro-alphabet has size $\sigma^\ell=\Theta(\sqrt{n})$. In this way, the 0-th order entropy-encoding of S_ℓ gives the k-th order entropy-encoding of S as stated below.

Lemma 1 ([6]). *For any ℓ, with $1\le\ell\le n$, it holds $n'H_0(S_\ell)\le nH_k(S)+O(n'k\lg\sigma)$, simultaneously over all $k\le\ell$.*

Lemma 1 implies that if we can maintain a dynamic compressed representation of S_ℓ in $n'H_0(S_\ell)+O(n'\lg\lg n)$ bits, we obtain a dynamic compressed representation of S in $nH_k(S)+O(n\frac{k\lg\sigma+\lg\lg n}{\lg_\sigma n})$ bits.

Codewords for the macro-alphabet Σ^ℓ of S_ℓ. We now focus on a dynamic encoding of the macro-symbols in Σ^ℓ to obtain a 0-th order entropy-encoding of S_ℓ. We divide the whole set of assigned codewords into classes C_j, where $0\le j\le\frac{1}{2}\lg n'$, and each of the codewords in C_j is of fixed length $j+3$ bits. We also assign a nonempty *set* Γ_c of codewords to each macro-symbol $c\in\Sigma^\ell$ (and remark that c can be encoded by any codeword in Γ_c). We want to preserve the following invariants (on the number of codewords):

1. $|C_j|<2^{j+3}$, for any $0\le j\le\frac{1}{2}\lg n'$.
2. $|C_j\cap\Gamma_c|\le 1$, for any class C_j and any macro-symbol $c\in\Sigma^\ell$.
3. $|\Gamma_c\cap\Gamma_{c'}|=0$, for any two distinct macro-symbols $c,c'\in\Sigma^\ell$.

We will discuss the rationale of the invariants 1–3 and how to maintain them in the next paragraphs. From a data structure point of view, for each class C_j we keep (i) a decoding table (an array of 2^{j+3} entries) D_j that maps each codeword to its assigned macro-symbol and (ii) a free-list F_j of available codewords that

can be still assigned to that class if required (where the next available codeword is the head of F_j). Moreover, for each macro-symbol c we keep (iii) an array W_c of $\frac{1}{2}\lg n'$ entries that represent Γ_c: entry $W_c[j]$ contains the codeword of C_j that has been assigned to c, or $\perp$ when there is no such codeword; and (iv) a counter f_c that stores the number of occurrences of c within the current S_ℓ. The space required for storing (i)–(iv) is $O(\sum_{j=0}^{\frac{1}{2}\lg n'} 2^{j+3}(\lg\sigma^\ell + \lg n) + \sigma^\ell \lg^2 n' + \sigma^\ell \lg n') = O(\sqrt{n}\lg n \lg n') = o(n')$ bits.

Dynamic 0*-th order entropy-encoding of* S_ℓ. The encoding of S_ℓ is obtained by concatenating the codewords of its n' macro-symbols: given any occurrence of macro-symbol c, this occurrence is encoded by any chosen codeword from Γ_c. However the codewords may change during the lifetime of S_ℓ, and thus we cannot simply store the resulting encoding of S_ℓ as a binary string: we need to access and modify codewords, and to increase or reduce the space reserved to them. Hence, we use the data structure in [11, Th.6] that stores a set of n' binary strings of up to $\lg n'$ bits each, and supports the following two operations: *(1)* Address(i) returns a memory pointer to the ith binary string ($1 \le i \le n'$), and *(2)* Realloc(i, b) changes the length of the ith binary string to b bits ($b \le \lg n'$), as restated next using our parameters.

Lemma 2 ([11]). *Let B be a set of n' binary strings, each of length at most $\lg n'$, and let s be the total number of bits for all the strings in B. We can store B in $s + O(n' \lg\lg n + \lg^4 n)$ bits while supporting* address *and* realloc *in $O(1)$ time.*

Our plan is to use Lemma 2 with $s = n'H_0(S_\ell) + O(n')$: by the discussion in the previous paragraphs, the total space occupancy of our encoding scheme is $n'H_0(S_\ell) + O(n' \lg\lg n)$ bits as claimed.

Initialization. The macro-symbols in S_ℓ are grouped into classes having approximately the same number of occurrences. Specifically, we scan S_ℓ and compute the number of occurrences f_c of each macro-symbol c that appear in it. Then we assign one class C_j to each each c using f_c, so that $\frac{n'}{2^j} < f_c \le \frac{n'}{2^{j+1}}$.[1] We initialize each class C_j to be empty and set its free-list F_j to contain all the 2^{j+3} binary codewords of fixed length $j+3$ bits. We also initialize each array W_c to contain all $\perp$s (see points (i)–(iv) in a previous paragraph). Then, for each macro-symbol c with $f_c > 0$, say of class C_j, we extract a codeword w from F_j and set $D_j[w] = c$ and $W_c[j] = w$. Note that after the initialization, we have a single codeword shared by all the occurrences of the same macro-symbol but, as we will see shortly, this is not mandatory during the rest of the lifetime of S_ℓ.

Operation Access(i, m). Let $p = \lceil i/\ell \rceil$. We retrieve the pth macro-symbol and its next $O(m/\ell)$ macro-symbols (if needed), taking $O(1)$ time per macro-symbol

[1] An exception is the last class C_j for $j = \frac{1}{2}\lg n'$, containing the macro-symbols with less than $\frac{n'}{2^{j+1}} = O(\sqrt{n'})$ occurrences.

(i.e. ℓ symbols in S) as follows. We first retrieve the pth codeword, say w, and its length, say j', using address as in Lemma 2. In this way we infer that $w \in C_{j'-3}$ and return the ℓ symbols of Σ that are stored in table entry $D_{j'-3}[w]$ in $O(1)$ time (since they take a total of $O(\lg n)$ bits). We repeat this task $O(1 + m/\ell)$ times for $p, p+1, \ldots$, thus attaining a cost of $O(1 + \frac{m}{\lg_\sigma n})$ time.

Operation Replace(i, c). We first describe an amortized implementation, which will then be deamortized. During an execution of Replace, the number of occurrences of a macro-symbol can change. Thus, macro-symbols may move to different classes in the lifetime of the data structure. Once a macro-symbol enters a class for the first time, it is assigned an available codeword of that class. Since the number of available codewords in any class is limited, it may happen that the last available codeword is consumed in this way. For the moment, we rebuild the whole data structure from scratch in that case. We have thus two conflicting goals. On one hand, the codewords of a class should be as large as possible to postpone the rebuilding. On the other hand, these codewords should be as small as possible to limit the loss with respect to entropy. We will show that rebuilding is needed only after $\Omega(n')$ Replace operations and, simultaneously, that the loss in the entropy is just $O(1)$ bits per macro-symbol of S_ℓ.

Operation Replace(i, c) must change the pth macro-symbol in S_ℓ, where $p = \lceil i/\ell \rceil$. Wlog assume that $S_\ell[p] = x$ has to be replaced by macro-symbol y (i.e., y is obtained from x by substituting x's symbol in position $i - (p-1)\ell$ with c). We perform the following steps and maintain Invariants 1–3 mentioned earlier:

1. Set $f_x = f_x - 1$ and $f_y = f_y + 1$ (data structures (iv)).
2. Let C_j be the current class of macro-symbol y (for the updated f_y):
   (a) If there is a codeword in C_j assigned to y, let e be such a codeword.
   (b) If such a codeword does not exist, extract e from F_j and assign it to y.
3. Encode $S_\ell[p]$ with e (see Lemma 2) and update data structures (i)–(iii).
4. If $|C_j| = 2^{j+3}$ (i.e., F_j is empty), rebuild all the data structures from scratch.

Lemma 3. *$\Omega(n')$ Replace operations are required before $|C_j| = 2^{j+3}$, for any j.*

Proof. Initially, at most 2^j codewords of class C_j are used. Indeed, a macro-symbol c is in class C_j iff its number of occurrences f_c is in $(\frac{n'}{2^j}, \frac{n'}{2^{j+1}}]$. Pessimistically, assume that all the macro-symbols in the classes C_{j-1} and C_{j+1} move to class C_j. We have at most 2^{j+2} codewords assigned to macro-symbols in classes C_{j-1}, C_j and C_{j+1}. Any other macro-symbol c has a number of occurrences $\Theta(\frac{n'}{2^j})$ away from the interval $(\frac{n'}{2^j}, \frac{n'}{2^{j+1}}]$. Thus, to use the remaining (at least) $2^{j+3} - 2^{j+2} = 2^{j+2}$ codewords, we need at least $2^{j+2} \times \Theta(\frac{n'}{2^j}) = \Omega(n')$ Replace operations. □

The time complexity of Replace is clearly dominated by Step 4. Indeed, Steps 1–3 take $O(1)$ time while Step 4 requires $O(n')$ time. Lemma 3 implies that we can amortize this cost to $O(1)$ time. To deamortize the cost, we employ an incremental rebuilding scheme to obtain our claimed bounds.

We scan the macro-symbols in S_ℓ and update their codewords and data structures, namely, the codeword of the macro-symbol at hand is replaced with the codeword of its current class. We can release and reassign any codeword that is no longer in use in the encoding of S_ℓ using the free-lists. More precisely, we fix a constant d and, at the rth Replace, we update the codewords of the d macro-symbols of S_ℓ in positions $1 + (d \cdot r) \mod n', \ldots, 1 + (d \cdot r + d - 1) \mod n'$. If a codeword is replaced, we check if it is no longer in use and, if so, we release it to F_j, where its length is $j + 3$. By Lemma 3, it follows that this mechanism guarantees that no class can use all its codewords because the reassignment is faster by a constant factor and terminates before any condition $|C_j| = 2^{j+3}$ may happen (i.e., the condition in Step 4 does not hold anymore at this point).

Bounding the space occupancy. It remains to prove that in Lemma 2, our s is at most $\sum_{c \in \Sigma^\ell}(f_c \lg \frac{n'}{f_c} + O(f_c)) = n' H_0(S_\ell) + O(n')$ bits. (Recall that this gives the bound of Theorem 1 by Lemma 1.) It suffices to prove that the overall length of the codewords in Γ_c representing the f_c occurrences of any macro-symbol c in S_ℓ can be bounded by $f_c \lg \frac{n'}{f_c} + O(f_c)$ bits.

Lemma 4. *For any macro-symbol $c \in \Sigma^\ell$, the overall space required by the f_c codewords of c in the encoding of S_ℓ is $f_c \lg \frac{n'}{f_c} + O(f_c)$ bits.*

Proof. Assume that C_j is the current class of macro-symbol c (i.e., $\frac{n'}{2^j} < f_c \leq \frac{n'}{2^{j+1}}$). In the ideal scenario, all the f_c occurrences are encoded with the $j + 3$-length codeword of class C_j. In this case, each of them would require $j + 3 = \lg \frac{n'}{f_c} + O(1)$ bits and the thesis would follow. However, there are occurrences of c encoded with codewords from other classes. For each class C_i, let $f_{c,i}$ be the number of occurrences of c encoded with a codeword of class C_i. The amount of additional space over the ideal scenario above can be bounded by $\sum_{i=0}^{\frac{1}{2}\lg n'} f_{c,i} \cdot (i - j) \leq \sum_{i>j}^{\frac{1}{2}\lg n'} f_{c,i} \cdot (i - j) = O(f_c)$, where the latter equality follows by observing that $f_{c,i} \leq \frac{f_c}{2^{i-j-1}}$, for any $i > j$. □

2.2 Supporting Access, Replace, Insert and Delete (Theorem 2)

Different rules of the game. As previously mentioned, the lower bound of $\Omega(\frac{\lg n}{\lg \lg n})$ time in [7] applies to the operations in this setting. Keeping this in mind, we can orchestrate a different data layout than the one described in Section 2.1 as we have $O(\frac{\lg n}{\lg \lg n})$ time per operation. This, in turn, allows us to reduce the redundancy from $O(n\frac{k \lg \sigma + \lg \lg n}{\lg_\sigma n})$ to $O(n\frac{\lg \lg n}{\lg_\sigma n})$ bits and, moreover, the resulting redundancy is now *independent* of k.

We need an alternative to Lemma 1 that uses the *first* order entropy H_1 and *variable-length* blocks. Consider *any* partition of S as a sequence $S_{m,M}$ of n' blocks, where $0 \leq m \leq M \leq n$. Each block is a substring of S of length ranging from m to M and can be seen as belonging to the macro-alphabet $\Lambda = \cup_{i=m}^{M} \Sigma^i$. Note that we use the term 'block' rather than 'macro-symbol' (as in Section 2.1)

because blocks are now of variable length and will be split and merged when necessary; thus, we will refer to $S_{m,M}$ as a sequence of blocks. We now relate $H_1(S_{m,M})$ to $H_k(S)$ as follows.

Lemma 5. *For any m, M, with $0 \leq m \leq M \leq n$, it holds $n'H_1(S_{m,M}) \leq nH_k(S) + O\left(n'(1 + \lg(M - m + 1)) + k \lg \sigma\right)$, simultaneously over all $k \leq m$.*

Proof. To obtain this inequality we define a k-order encoder E giving a compression size for S that is lower bounded by $n'H_1(S_{m,M})$ bits and upper bounded by $nH_k(S) + O(n'(1 + \lg(M - m + 1)) + k \lg \sigma)$ bits (so the claim will follow).

We first discuss the upper bound. For every position i ($k \leq i \leq n$), let p_i denote the empirical conditional probability of seeing symbol $S[i]$ after the k-order context $S[i - k, i - 1]$. Given these p_i's, the k-order arithmetic encoder represents S within $\sum_{i=k}^{n} \lg p_i + 2 + k \lg \sigma \leq nH_k(S) + 2 + k \lg \sigma$ bits (see e.g., [6,8]).[2] Using this fact, our encoder E encodes the blocks of S individually: *(1)* it writes the length of the block by using $O(1 + \lg(M - m + 1))$ bits; *(2)* it encodes the symbols in the block with the aforementioned k-order arithmetic encoder. This approach increases the above encoding by $O(n' + n' \lg(M - m + 1))$ bits. The size is at most $nH_k(S) + O(n' + n' \lg(M - m + 1)) + O(k \lg \sigma)$ bits, as claimed.

We now discuss the lower bound. Note that the information content of $S_{m,M}$ and S is the same as they represent the same string, and E assigns to each block of $S_{m,M}$ its own binary codeword. These codewords do not uniquely identify a block (i.e., there may exist two different blocks that have been assigned the same codeword). However, since the k-order context of any symbol is within its own block or the preceding block, for any block B, these codewords uniquely identify all the blocks that follows B in $S_{m,M}$. Thus, the compression size of E has to be at least $n'H_1(S_{m,M})$ bits, where blocks of $S_{m,M}$ are seen as macro-symbols from the alphabet Λ: the first-order entropy is a lower bound for any compressor which encodes each macro-symbol with a codeword that only depends on the macro-symbol itself and the immediately preceding one [12]. □

Toy case. Armed with Lemma 5, let us first study the static case to achieve the first order entropy for $S_{\ell,\ell}$, with $\ell = m = M = \frac{1}{8} \lg_\sigma n$, and $O(\lg n / \lg \lg n)$ time for Access. Lemma 5 implies that we obtain a compressed representation of S in $nH_k(S) + O(n \frac{\lg \lg n}{\lg_\sigma n})$ bits.

We divide $S_{\ell,\ell}$ into super-blocks of $b = \lg n / \lg \lg n$ blocks each. (*a*) For each block $c \in S_{\ell,\ell}$, we construct a table T_c^1 that stores the blocks following c in $S_{\ell,\ell}$, sorted by their *conditional* frequencies. Each of these blocks has been assigned a codeword from the set $\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$: the larger the conditional frequency of a block, the shorter its codeword.

In order to achieve $n'H_1(S_{\ell,\ell})$ plus lower order terms, each super-block is encoded as follows: the first block of the super-block is written uncompressed; each remaining block c' is encoded by its codeword $T_c^1[c']$, where c is the block preceding c' in the super-block.

[2] The term $k \lg \sigma$ accounts for the cost of writing explicitly the first k symbols of S, which do not have any k-order context.

Apart from the tables in (a), the encoding of $S_{\ell,\ell}$ comprises (b) the concatenation of the aforementioned encodings of its super-blocks, and (c) a suitable encoding of the starting positions in (b) of the individual encodings of the super-blocks and their blocks (this is necessary as the codewords are not prefix-free).

As for the space occupancy, we have that the tables in (a) require $O(\sigma^{2\ell} \lg n) = o(n^{1/2})$ bits; the encoding in (b) requires at most $H_1(S_{\ell,\ell})$ bits plus $O(\lg \sigma^{\ell}) = O(\lg n)$ bits per super-block and $O(1)$ bits per block; the encoding in (c) requires $O(n\frac{\lg\lg n}{\lg_\sigma n})$ bits using a standard two-level solution with succinct dictionaries [18]. Summing up, the space for this static scheme is $n'H_1(S_{\ell,\ell})+O(n\frac{\lg\lg n}{\lg_\sigma n}) \leq nH_k(S) + O(n\frac{\lg\lg n}{\lg_\sigma n})$ bits by Lemma 5 (setting $\ell = m = M = \frac{1}{8}\lg_\sigma n$).

Operation $\mathsf{Access}(i, m)$ identifies the pth block of $S_{\ell,\ell}$ as in Section 2.1 but it now decompresses the whole super-block containing that block. This requires $O(\lg n/\lg\lg n)$ time. After that, the decoding of the next $O(m/\ell)$ blocks adds $O(m/\ell)$ time to the latter cost.

Fully dynamic representation. We maintain a *flexible* partition $S_{\ell,2\ell}$ of S, where super-blocks contains between b and $2b$ blocks of $S_{\ell,2\ell}$ (i.e., between $b\ell$ and $4b\ell$ symbols of S). We do not give here the implementation details that are of common use in dynamic data structures. For example, once a block (or a super-block) becomes too large or too small, it is split or merged with the preceding block (or super-block). Instead, we discuss how to maintain the compressed representation for $S_{\ell,2\ell}$ in terms of the data structures (a)–(c) previously discussed.

As for (c), we adopt the dynamic binary vector in [21] using $O(n \lg\lg n/\lg_\sigma n)$ bits, and supporting Rank, Select, Insert, and Delete in $O(\lg n/\lg\lg n)$ time.

As for (a) and (b), observe that when any $\mathsf{Replace}$, Insert or Delete of a symbol is performed in S, we have to change at most 2 blocks of $S_{\ell,2\ell}$. Consequently, some conditional frequencies should be changed in the tables T_c^1 in (a) but we cannot afford the time cost of this change. Here is our solution in short.

Be lazy. Let us consider the snapshot of the tables T_c^1 in (a) for a certain instance of $S_{\ell,2\ell}$ during its lifetime. We use them *anyway* to encode the blocks in (b) of the current instance, called $S'_{\ell,2\ell}$, obtained after say r updates of S. This potentially introduces a loss in space as we are using the outdated statistics of $S_{\ell,2\ell}$ to encode also the blocks of $S'_{\ell,2\ell}$. The result in [11, Th.4] shows that this loss can be bounded by $O(r \lg n)$ bits.[3] By choosing $r = \Theta(n \lg\sigma \lg\lg n/\lg^2 n)$, this loss remains dominated by the redundancy of Theorem 2.

Use amortization and deamortize. By our choice of r, we have $\Theta(r\frac{\lg n}{\lg\lg n}) = \Theta(n/\ell)$ credits after r update operations: they are enough to cover the cost of updating the tables in (a) and re-encoding all the $\Theta(n/\ell)$ blocks (and their super-blocks) in (b) and (c). We thus have a scheme supporting all the operations in $O(\frac{\lg n}{\lg\lg n})$ amortized time. From this point, the deamortization scheme follows the same idea in [11], which consists in dividing updates into phases and spreading the cost of the re-encoding within each phase. We refer to [11] for the details.

[3] We measure the loss with respect to $|S'_{\ell,2\ell}|H_1(S'_{\ell,2\ell})$ of the current partition.

3 Dynamizing Static Rank/Select Succinct Indexes

This section presents a dynamization result and an example of its application. A Rank/Select succinct index for a static string is a data structure that supports Rank and Select on a string S but reads symbols from S *solely* through (constant-time) *probe* operations. A Rank/Select succinct index for a dynamic string S also supports Rank and Select on S, also only probes S, but is notified when S is modified by a Replace operation. We show:

Theorem 3. *Let $\mathcal{I}$ be a succinct index for a static string $S[1..n]$ over the alphabet $[\sigma]$ that occupies at most $f(n,\sigma)$ bits of space, where f is a convex function of n. Further, suppose that $\mathcal{I}$ can be constructed in linear time using $O(f(n,\sigma))$ bits of space. Then, $\mathcal{I}$ can be used to build a succinct index for a dynamic string S, supporting* Rank *and* Select *with an additive overhead of $O(\lg n/\lg\lg n)$ time per operation and* Replace *on S in $O(\lg n/\lg\lg n)$ amortized time. The resulting index requires $O(n\frac{(\lg\sigma+\lg\lg n)\lg\lg n}{\lg n})$ bits of additional space, plus $O(f(n,\sigma))$ bits of temporary working space.*

Choosing the Rank/Select index from [9, Theorem 4(a)] as $\mathcal{I}$ and Theorem 1 to store S, we obtain the following result via Theorem 3:

Corollary 1. *A string $S[1,n]$ over the alphabet $\Sigma=[\sigma]$ can be stored in $nH_k(S)+O(n(\lg\lg\sigma+(k\lg\sigma\lg\lg n)/\lg_\sigma n))$ bits for all $k=o(\lg_\sigma n)$, so as to support* Rank *and* Select *in $O(\frac{\lg n}{\lg\lg n})$ time,* Access(i,m) *in $O(1+\frac{m}{\lg_\sigma n})$ time, and* Replace *in $O(\frac{\lg n}{\lg\lg n})$ amortized time.*

3.1 Proof of Theorem 3

Two-level solution. We partition the string S into chunks of $m=\sigma\lg^2 n$ symbols each, and we use $M=n/m$ to denote the number of these chunks ($M=1$ if $\sigma=\Omega(n/\lg^2 n)$). The first level dynamically maintains the cumulative number of occurrences of each symbol in each chunk while the second level builds a static instance of $\mathcal{I}$ on each chunk together with additional data structures to correct its potentially wrong answers. The first level routes a Rank or Select query to the appropriate chunk; the second level handles the query for that chunk.

First level. For each symbol c, we maintain an array $A_c[1,M]$, where $A_c[i]$ stores the number of occurrences of symbol c in the ith chunk, for $1\le i\le M$. Every time we replace a symbol c in the ith chunk with a new symbol c', we increase $A_{c'}[i]$ and decrease $A_c[i]$, both by 1. We then construct an instance of the dynamic partial-sums data structure in [16] on each of these arrays. This data structure answers prefix sum queries, predecessor queries on the prefix sums and ±1 updates on the arrays in $O(\lg n/\lg\lg n)$ worst-case time. For each symbol c, it is easy to see that these queries suffice to reduce Rank and Select on S to those on a chunk. As the data structures use a linear number of words of memory, the overall space usage for all A_c's is $O(M\sigma\lg n)=O(n/\lg n)$ bits overall.

Second level. The second level supports Rank, Select and Replace on each chunk C (of size m). The key idea in our solution is to build a static index $\mathcal{I}_C$ on C, whose local operations are denoted by Static_Rank and Static_Select to distinguish them from Rank and Select that we want to support dynamically. The index $\mathcal{I}_C$ is rebuilt from scratch as soon as $\Theta(m \lg\lg n / \lg n)$ updates occur in C: since rebuilding takes $O(m)$ time, the rebuilding phase costs $O(\lg n / \lg\lg n)$ amortized time per update. Note that while Static_Rank and Static_Select need to probe the original content of C, Replace operations could have changed C meanwhile. In the full version, we will show how to solve this problem.

Now, consider Rank and Select queries on the current content of C. They are solved by first querying the static index $\mathcal{I}_C$ with Static_Rank and Static_Select. However, they may report incorrect answers since $\mathcal{I}_C$ is built on the original content of C. We give two solutions to correct these potentially incorrect answers, depending on the alphabet size σ. The solution for small alphabets, namely, when $\lg\sigma = O(\lg n / \lg\lg n)$, is quite standard and will be given in the full version of this paper. Below we discuss the solution for large alphabets, namely, when $\lg\sigma = \omega(\lg n / \lg\lg n)$ and so we can use $O(n)$ additional bits without increasing the space complexity in Theorem 3.

Operations for large alphabets. Here we implement the operations as follows.

Rank is supported by indexing two dynamic sequences $A[1,m]$ and $D[1,m]$ that keep track of the modifications occurred in the chunk C. Initially, when there are no updates, $A = \perp^m$ and $D = \perp^m$ for a special symbol $\perp$ that denotes no symbol change. An update that modifies the i-th position in C with symbol c is recorded by setting $A[i] = c$ and $D[i] = C[i]$: subsequent modifications of the i-th position change only $A[i] = c$. Supporting Rank on A and D suffices for correcting the (potentially) wrong results of Static_Rank. Indeed, we have $\mathsf{Rank}(c,i) = \mathsf{Static\_Rank}(c,i) + \mathsf{Rank}_A(c,i) - \mathsf{Rank}_D(c,i)$. We observe that A and D contain only $O(m \lg\lg n / \lg n) = O(\sigma \lg n \lg\lg n)$ occurrences of symbols different from $\perp$, before the rebuilding. This sparseness allows us to remain within the space bound of Theorem 3 by using the solution of Navarro and Nekrich [13] on A and D. Indeed, it requires $mH_0(A) + O(m + \sigma \lg^{1+\epsilon} m) = O(m)$ bits for A, as $m = \sigma \lg^2 n$, and an analogous argument holds for D.

Select is supported by maintaining a dynamic ternary vector T_c for each symbol c over the alphabet $\{\perp, +, -\}$ as follows. Initially, we start with the sequence $T_c = \perp^{n_c}$, where n_c is the number of occurrences of c in C. When the j-th occurrence of c in the sequence is replaced by some other symbol, we change the j-th occurrence of a symbol from $\{\perp, +\}$ in T_c to a $-$. When a symbol other than c at position i is replaced by a c, then first we count the number of occurrences j of c before the position j, using a Rank operation, and then insert a $+$ after the j-th occurrence of a symbol from $\{\perp, +\}$ in T_c. We use the index in [21] for supporting $\mathsf{Select}_{T_c}(\perp/+, i)$ and $\mathsf{Rank}_{T_c}(\perp/+, j)$ in $O(\lg n / \lg\lg n)$ time. In this case the space usage is: $\sum_{c \in [\sigma]} O(n_c \lg 3) = O(m)$ bits. Operation $\mathsf{Select}(c,i)$ is then the following one:

1. Set $j = \mathsf{Select}_{T_c}(\perp/+, i)$ and $k = \mathsf{Rank}_{T_c}(+, j)$.
2. If $T_c[j] = \perp$, return $\mathsf{Static\_Select}(c, j-k)$; else, return $\mathsf{Select}_A(c,k)$.

Finally, Replace is supported by changing the content of A, D and T_c's, as discussed above.

References

1. Barbay, J., He, M., Munro, J.I., Satti, S.R.: Succinct indexes for strings, binary relations and multilabeled trees. ACM Transactions on Algorithms 7, 52 (2011)
2. Brodnik, A., Carlsson, S., Demaine, E.D., Munro, J.I., Sedgewick, R.: Resizable arrays in optimal time and space. In: Dehne, F., Gupta, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 37–48. Springer, Heidelberg (1999)
3. Dietzfelbinger, M., Karlin, A.R., Mehlhorn, K., Meyer auf der Heide, F., Rohnert, H., Tarjan, R.E.: Dynamic perfect hashing: Upper and lower bounds. SIAM J. Comput. 23, 738–761 (1994)
4. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and indexing labeled trees, with applications. J. ACM 57 (2009)
5. Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM 52, 552–581 (2005)
6. Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. Theor. Comput. Sci. 372, 115–121 (2007)
7. Fredman, M.L., Saks, M.E.: The cell probe complexity of dynamic data structures. In: STOC, pp. 345–354 (1989)
8. González, R., Navarro, G.: Compressed text indexes with fast locate. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 216–227. Springer, Heidelberg (2007)
9. Grossi, R., Orlandi, A., Raman, R.: Optimal trade-offs for succinct string indexes. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 678–689. Springer, Heidelberg (2010)
10. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J. Computing 35, 378–407 (2005)
11. Jansson, J., Sadakane, K., Sung, W.-K.: CRAM: Compressed random access memory. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 510–521. Springer, Heidelberg (2012)
12. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM 48, 407–430 (2001)
13. Navarro, G., Nekrich, Y.: Optimal dynamic sequence representations. In: Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA (2013)
14. Pagh, R.: A new trade-off for deterministic dictionaries. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 22–31. Springer, Heidelberg (2000)
15. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms 51, 122–144 (2004)
16. Pătraşcu, M., Demaine, E.D.: Logarithmic lower bounds in the cell-probe model. SIAM J. Comput. 35, 932–963 (2006)
17. Patrascu, M., Viola, E.: Cell-probe lower bounds for succinct partial sums. In: Charikar, M. (ed.) SODA, pp. 117–122. SIAM (2010)
18. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM TALG 3 (2007)
19. Raman, R., Raman, V., Rao, S.S.: Succinct Dynamic Data Structures. In: Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 2001. LNCS, vol. 2125, pp. 426–437. Springer, Heidelberg (2001)
20. Sadakane, K., Grossi, R.: Squeezing succinct data structures into entropy bounds. In: SODA, pp. 1230–1239. ACM Press (2006)
21. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: SODA, pp. 134–149 (2010)

The Complexity of Planar Boolean #CSP with Complex Weights*

Heng Guo and Tyson Williams

University of Wisconsin-Madison, Madison, WI, USA
{hguo,tdw}@cs.wisc.edu

Abstract. We prove a complexity dichotomy theorem for symmetric complex-weighted Boolean #CSP when the constraint graph of the input must be planar. The problems that are #P-hard over general graphs but tractable over planar graphs are precisely those with a holographic reduction to matchgates. This generalizes a theorem of Cai, Lu, and Xia for the case of real weights. We also obtain a dichotomy theorem for a symmetric arity 4 signature with complex weights in the planar Holant framework, which we use in the proof of our #CSP dichotomy. In particular, we reduce the problem of evaluating the Tutte polynomial of a planar graph at the point $(3, 3)$ to counting the number of Eulerian orientations over planar 4-regular graphs to show the latter is #P-hard. This strengthens a theorem by Huang and Lu to the planar setting.

1 Introduction

In 1979, Valiant [2] defined the class #P to explain the apparent intractability of counting the number of perfect matchings in a graph. Yet over a decade earlier, Kasteleyn [3] gave a polynomial-time algorithm to compute this quantity for planar graphs. This was an important milestone in a decades-long research program by physicists in statistical mechanics to determine what problems the restriction to the planar setting renders tractable [4–10, 3, 11–13]. More recently, Valiant introduced matchgates [14, 15] and *holographic* algorithms [16, 17] that rely on Kasteleyn's algorithm to solve certain counting problems over planar graphs. In a series of papers [18–21], Cai et al. characterized the local constraint functions (which define counting problems) that are representable by matchgates in a holographic algorithm.

From the viewpoint of computational complexity, we seek to understand exactly which intractable problems the planarity restriction enable us to efficiently compute. Partial answers to this question have been given in the context of various counting frameworks [22–25]. In every case, the problems that are #P-hard over general graphs but tractable over planar graphs are essentially those characterized by Cai et al. In this paper, we give more evidence for this phenomenon by extending the results of [23] to the setting of complex-valued constraint functions. This provides the most natural setting to express holographic algorithms and transformations.

* Full version with proofs available at [1].

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 516–527, 2013.

Our main result is a dichotomy theorem for the framework of counting Constraint Satisfaction Problems (#CSP), but our proof is in a generalized framework called Holant problems [26–29]. We briefly introduce the Holant framework and then explain its main advantages. A set of functions $\mathcal{F}$ defines the problem Holant($\mathcal{F}$). An instance of this problem is a tuple $\Omega = (G, \mathcal{F}, \pi)$ called a *signature grid*, where $G = (V, E)$ is a graph, π labels each $v \in V$ with a function $f_v \in \mathcal{F}$, and f_v maps $\{0,1\}^{\deg(v)}$ to $\mathbb{C}$. We also call the functions in $\mathcal{F}$ *signatures*. An assignment σ for every $e \in E$ gives an evaluation $\prod_{v \in V} f_v(\sigma \mid_{E(v)})$, where $E(v)$ denotes the incident edges of v and $\sigma \mid_{E(v)}$ denotes the restriction of σ to $E(v)$. The counting problem on the instance Ω is to compute

$$\text{Holant}_\Omega = \sum_{\sigma: E \to \{0,1\}} \prod_{v \in V} f_v \left(\sigma \mid_{E(v)}\right). \tag{1}$$

Counting the number of perfect matchings in G corresponds to attaching the Exact-One signature at every vertex of G. A function or signature is called *symmetric* if its output depends only on the Hamming weight of the input. We often denote a symmetric signature by the list of its outputs sorted by input Hamming weight in ascending order. For example, $[0, 1, 0, 0]$ is the Exact-One function on three bits. The output is 1 if and only if the input is 001, 010, or 100, and 0 otherwise.

We consider #CSP, which are also parametrized by a set of functions $\mathcal{F}$. The problem #CSP($\mathcal{F}$) is equivalent to Holant($\mathcal{F} \cup \mathcal{EQ}$), where $\mathcal{EQ} = \{=_1, =_2, \dots\}$ and $(=_k) = [1, 0, \dots, 0, 1]$ is the equality signature of arity k. This explicit role of equality signatures permits a finer classification of problems. For a direct definition of #CSP, see [30].

We often consider a Holant problem over bipartite graphs, which is denoted by Holant($\mathcal{F} \mid \mathcal{G}$), where the sets $\mathcal{F}$ and $\mathcal{G}$ contain the signatures available for assignment to the vertices in each partition. Considering the edge-vertex incidence graph, one can see that Holant($\mathcal{F}$) is equivalent to Holant($=_2 \mid \mathcal{F}$). One powerful tool in this setting is the holographic transformation. Let T be a nonsingular 2-by-2 matrix and define $T\mathcal{F} = \{T^{\otimes \text{arity}(f)} f \mid f \in \mathcal{F}\}$, where $T^{\otimes k}$ is the tensor product of k factors of T. Here we view f as a column vector by listing its values in lexicographical order as in a truth table. Similarly $\mathcal{F}T$ is defined by viewing $f \in \mathcal{F}$ as a row vector. Valiant's Holant theorem [16] states that Holant($\mathcal{F} \mid \mathcal{G}$) is equivalent to Holant($\mathcal{F}T^{-1} \mid T\mathcal{G}$).

Cai, Lu, and Xia gave a dichotomy for complex-weighted Boolean #CSP($\mathcal{F}$) in [28]. Let Pl-#CSP($\mathcal{F}$) (resp. Pl-Holant($\mathcal{F}$)) denote the #CSP (resp. Holant problem) defined by $\mathcal{F}$ when the inputs are restricted to planar graphs. In this paper, we investigate the complexity of Pl-#CSP($\mathcal{F}$) for a set $\mathcal{F}$ of symmetric complex-weighted functions. In particular, we would like to determine which sets become tractable under this planarity restriction. Holographic algorithms with matchgates provide planar tractable problems for sets that are matchgate realizable after a holographic transformation. From the Holant perspective, the signatures in $\mathcal{EQ}$ are always available in #CSP($\mathcal{F}$). By the signature theory of Cai

and Lu [21], the Hadamard matrix $H = \left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$ essentially defines the only[1] holographic transformation under which $\mathcal{EQ}$ becomes matchgate realizable. Let $\widehat{\mathcal{F}}$ denote $H\mathcal{F}$ for any set of signatures $\mathcal{F}$. Then $\widehat{\mathcal{EQ}}$ is $\{[1,0],[1,0,1],[1,0,1,0],\dots\}$ while $(=_2)(H^{-1})^{\otimes 2}$ is still $=_2$. Therefore #CSP($\mathcal{F}$) and Holant($\mathcal{F} \cup \mathcal{EQ}$) are equivalent to Holant($\widehat{\mathcal{F}} \cup \widehat{\mathcal{EQ}}$) by Valiant's Holant theorem.

Our main dichotomy theorem is stated as follows.

Theorem 1. *Let $\mathcal{F}$ be a set of symmetric, complex-valued signatures in Boolean variables. Then* Pl-#CSP($\mathcal{F}$) *is* #P*-hard unless $\mathcal{F}$ satisfies one of the following conditions, in which case it is tractable:*

1. #CSP($\mathcal{F}$) *is tractable (cf. [28]); or*
2. $\widehat{\mathcal{F}}$ *is realizable by matchgates (cf. [21]).*

A more explicit description of the tractable cases can be found in Theorem 19.

In many previous dichotomy theorems for Boolean #CSP($\mathcal{F}$), the proof of hardness began by pinning. The goal of this technique is to realize the constant functions $[1,0]$ and $[0,1]$ and was always achieved by a *nonplanar* reduction. This does not imply the collapse of any complexity classes because the tractable sets for #CSP($\mathcal{F}$) include $[1,0]$ and $[0,1]$. However, $\mathcal{EQ}$ with $\{[1,0],[0,1]\}$ are not simultaneously realizable as matchgates. Therefore, according to our main result, if pinning were possible for Pl-#CSP($\mathcal{F}$), then #P collapses to P! Instead, apply the Hadamard transformation and consider Pl-Holant($\widehat{\mathcal{F}} \cup \widehat{\mathcal{EQ}}$). In this Hadamard basis, pinning becomes possible again since $[1,0]$ and $[0,1]$ are included in every tractable set. Indeed, we prove our pinning result in this Hadamard basis, which is discussed in Section 4.

For Holant problems, it is often important to understand the complexity of the small arity cases first [23, 31, 32]. In [23], Cai, Lu, and Xia gave a dichotomy for Pl-Holant(f) when f is a symmetric arity 3 signature while a dichotomy for Holant(f) when f is a symmetric arity 4 signature was shown in [32]. In the proof of the latter result, most of the reductions were planar. However, the crucial starting point for hardness, namely counting Eulerian orientations (#EO) over 4-regular graphs, was not known to be #P-hard under the planarity restriction. Huang and Lu [31] had recently proved that #EO is #P-hard over 4-regular graphs but left open its complexity when the input is also planar. We show that #EO remains #P-hard over planar 4-regular graphs. The problem we reduce from is the evaluation of the Tutte polynomial of a planar graph at the point (3,3), which has a natural expression in the Holant framework. In addition, we determine the complexity of counting complex-weighted matchings over planar 4-regular graphs. The problem is #P-hard except for the tractable case of counting perfect matchings. With these two ingredients, we obtain a dichotomy for Pl-Holant(f) when f is a symmetric arity 4 signature.

Our main result is a generalization of the dichotomy by Cai, Lu, and Xia [23] for Pl-#CSP($\mathcal{F}$) when $\mathcal{F}$ contains symmetric real-weighted Boolean functions. It is natural to consider complex weights in the Holant framework because surprising equivalences between problems are often discovered via complex holographic

[1] Up to transformations under which matchgates are closed.

transformations, sometimes even between problems using only rational weights. Our proof of hardness for #EO over planar 4-regular graphs in Section 3 is a prime example of this. Extending the range from $\mathbb{R}$ to $\mathbb{C}$ also enlarges the set of problems that can be transformed into the framework.

However, a dichotomy for complex weights is more technically challenging. The proof technique of polynomial interpolation often has infinitely many failure cases in $\mathbb{C}$ corresponding to the infinitely many roots of unity, which prevents a brute force analysis of failure cases as was done in [23]. This increased difficulty requires us to develop new ideas to bypass previous interpolation proofs. In particular, we perform a planar interpolation with a rotationally invariant signature to prove the #P-hardness of #EO over planar 4-regular graphs. For the complexity of counting complex-weighted matchings over planar 4-regular graphs, we introduce the notion of planar pairings to build reductions. We show that every planar 3-regular graph has a planar pairing and that it can be efficiently computed. We also refine and extend existing techniques for application in the new setting, including the recursive unary construction, the anti-gadget technique, compressed matrix criteria, and domain pairing.

2 Preliminaries

The framework of Holant problems is defined for functions mapping any $[q]^k \to \mathbb{F}$ for a finite q and some field $\mathbb{F}$. In this paper, we investigate the complex-weighted Boolean Holant problems, that is, all functions are $[2]^k \to \mathbb{C}$. Technically, functions must take complex algebraic numbers for issues of computability.

A *signature grid* $\Omega = (G, \mathcal{F}, \pi)$ consists of a graph $G = (V, E)$, where each vertex is labeled by a function $f_v \in \mathcal{F}$, and $\pi : V \to \mathcal{F}$ is the labeling. If the graph G is planar, then we call Ω a *planar signature grid*. The Holant problem on instance Ω is to evaluate $\text{Holant}_\Omega = \sum_\sigma \prod_{v \in V} f_v(\sigma \mid_{E(v)})$, a sum over all edge assignments $\sigma : E \to \{0, 1\}$.

A function f_v can be represented by listing its values in lexicographical order as in a truth table, which is a vector in $\mathbb{C}^{2^{\deg(v)}}$, or as a tensor in $(\mathbb{C}^2)^{\otimes \deg(v)}$. We also use f^α to denote the value $f(\alpha)$, where α is a binary string. A function $f \in \mathcal{F}$ is also called a *signature*. A symmetric signature f on k Boolean variables can be expressed as $[f_0, f_1, \ldots, f_k]$, where f_w is the value of f on inputs of Hamming weight w. In this paper, we consider symmetric signatures. Since a signature of arity k must be placed on a vertex of degree k, we can represent a signature of arity k by a labeled vertex with k ordered dangling edges. Throughout this paper, we do not distinguish between these two views.

A Holant problem is parametrized by a set of signatures.

Definition 2. *For a signature set $\mathcal{F}$, define the counting problem* Holant($\mathcal{F}$) *as:*
Input: A signature grid $\Omega = (G, \mathcal{F}, \pi)$*;*
Output: Holant_Ω*.*

The problem Pl-Holant($\mathcal{F}$) is defined similarly using a planar signature grid. The Holant^c framework is the special case of Holant problems when the constant

signatures of the domain are freely available. In the Boolean domain, the constant signatures are $[1, 0]$ and $[0, 1]$.

Definition 3. *For signature set* $\mathcal{F}$, $\text{Holant}^c(\mathcal{F})$ *denotes* $\text{Holant}(\mathcal{F} \cup \{[0, 1], [1, 0]\})$.

The problem Pl-Holant$^c(\mathcal{F})$ is defined similarly. A symmetric signature f of arity n is *degenerate* if there exists a unary signature u such that $f = u^{\otimes n}$. Replacing a signature $f \in \mathcal{F}$ by a constant multiple cf, where $c \neq 0$, does not change the complexity of Holant$(\mathcal{F})$. It introduces a global factor to Holant$_\Omega$. Hence, for two signatures f, g of the same arity, we use $f \neq g$ to mean that these signatures are not equal in the projective space sense, i.e. not equal up to any nonzero constant factor. We denote polynomial time Turing equivalence by $\equiv_T$.

An instance of #CSP$(\mathcal{F})$ has the following bipartite view. Create a node for each variable and each constraint. Connect a variable node to a constraint node if the variable appears in the constraint function. This bipartite graph is also known as the *constraint graph*. Under this view, we can see that #CSP$(\mathcal{F}) \equiv_T$ Holant$(\mathcal{F} \mid \mathcal{EQ}) \equiv_T$ Holant$(\mathcal{F} \cup \mathcal{EQ})$, where $\mathcal{EQ} = \{=_1, =_2, =_3, \dots\}$ is the set of equality signatures of all arities. This equivalence also holds for the planar versions of these frameworks. For the #CSP framework, the following two signature sets are tractable [28].

Definition 4. *A* k*-ary function* $f(x_1, \dots, x_k)$ *is* affine *if it has the form* $\lambda \chi_{Ax=0} \cdot \sqrt{-1}^{\sum_{j=1}^{n} \langle \alpha_j, x \rangle}$, *where* $\lambda \in \mathbb{C}$, $x = (x_1, x_2, \dots, x_k, 1)^T$, A *is a matrix over* $\mathbb{F}_2$, α_j *is a vector over* $\mathbb{F}_2$, *and* χ *is a 0-1 indicator function such that* $\chi_{Ax=0}$ *is 1 iff* $Ax = 0$. *Note that the dot product* $\langle \alpha_j, x \rangle$ *is calculated over* $\mathbb{F}_2$, *while the summation* $\sum_{j=1}^{n}$ *on the exponent of* $i = \sqrt{-1}$ *is evaluated as a sum mod 4 of 0-1 terms. We use* $\mathscr{A}$ *to denote the set of all affine functions.*

Definition 5. *A function is of* product type *if it can be expressed as a product of unary functions, binary equality functions* $([1, 0, 1])$, *and binary disequality functions* $([0, 1, 0])$. *We use* $\mathscr{P}$ *to denote the set of product type functions.*

In the Holant framework, there are two corresponding signature sets that are tractable. A signature f is $\mathscr{A}$-transformable if there exists a holographic transformation T such that $f \in T\mathscr{A}$ and $[1, 0, 1]T^{\otimes 2} \in \mathscr{A}$. Similarly, a signature f is $\mathscr{P}$-transformable if there exists a holographic transformation T such that $f \in T\mathscr{P}$ and $[1, 0, 1]T^{\otimes 2} \in \mathscr{P}$. These two families are tractable because after a transformation by T, it is a tractable #CSP instance.

Matchgates were introduced by Valiant [14, 15] and are combinatorial in nature. They encode computation as a sum of weighted perfect matchings, which has a polynomial-time algorithm by the work of Kasteleyn [3].

We say a signature is a *matchgate signature* if there is some matchgate that realizes this signature and use $\mathscr{M}$ to denote the set of all matchgate signatures. Lemmas 6.2 and 6.3 in [18] (and the paragraph the follows them) characterize the symmetric signatures in $\mathscr{M}$. Instead of formally stating these two lemmas, we explicitly list all the symmetric signatures in $\mathscr{M}$: For any $\alpha, \beta \in \mathbb{C}$,

1. $[\alpha^n, 0, \alpha^{n-1}\beta, 0, \ldots, 0, \alpha\beta^{n-1}, 0, \beta^n]$;
2. $[\alpha^n, 0, \alpha^{n-1}\beta, 0, \ldots, 0, \alpha\beta^{n-1}, 0, \beta^n, 0]$;
3. $[0, \alpha^n, 0, \alpha^{n-1}\beta, 0, \ldots, 0, \alpha\beta^{n-1}, 0, \beta^n]$;
4. $[0, \alpha^n, 0, \alpha^{n-1}\beta, 0, \ldots, 0, \alpha\beta^{n-1}, 0, \beta^n, 0]$.

Roughly speaking, the symmetric matchgate signatures have 0 for every other entry (which is called the *parity condition*), and form a geometric progression with the remaining entries. We also say a signature f is $\mathscr{M}$-transformable if there exists a holographic transformation T such that $f \in T\mathscr{M}$ and $[1, 0, 1]T^{\otimes 2} \in \mathscr{M}$.

3 Pl-Holant Dichotomy for a Symmetric 4-ary Signature

One of our main results is a dichotomy theorem for Pl-Holant(f) when f is a symmetric arity 4 signature with complex weights, which uses the #P-hardness of counting Eulerian orientations over planar 4-regular graphs in a crucial way.

Recall that an orientation of the edges of a graph G is an *Eulerian orientation* if for each vertex v of G, the number of incoming edges of v equals the number of outgoing edges of v.

Counting the number of (unweighted) Eulerian orientations over 4-regular graphs was shown to be #P-hard in Theorem V.10 of [31]. We improve this result by showing that this problem remains #P-hard when the input is also planar. We reduce from the problem of counting weighted Eulerian orientations over medial graphs, which are planar 4-regular graphs (see Section 2 in [33] for a definition). Las Vergnas [34] showed that this problem is equivalent to evaluating the Tutte polynomial at the point (3,3), which is #P-hard for planar graphs [22].

Theorem 6 (Theorem 2.1 in [34]). *Let G be a connected plane graph and let $\mathscr{O}(H)$ be the set of all Eulerian orientations of the medial graph H of G. Then*

$$2 \cdot \mathrm{T}(G; 3, 3) = \sum_{O \in \mathscr{O}(H)} 2^{\beta(O)}, \tag{2}$$

where $\beta(O)$ is the number of saddle vertices in the orientation O, i.e. the number of vertices in which the edges are oriented "in, out, in, out" in cyclic order.

Our proof also uses two notions from [32].

Definition 7. *The matrix* $M_g = \begin{bmatrix} g^{0000} & g^{0010} & g^{0001} & g^{0011} \\ g^{0100} & g^{0110} & g^{0101} & g^{0111} \\ g^{1000} & g^{1010} & g^{1001} & g^{1011} \\ g^{1100} & g^{1110} & g^{1101} & g^{1111} \end{bmatrix}$ *is the* signature matrix *of an arity 4 signature g. When we present g pictorially, we order the four external edges $ABCD$ counterclockwise. In M_g, the row index bits are ordered AB and the column index bits are ordered DC, in a reverse way. This is for convenience so that the signature matrix of the linking of two arity 4 signatures is the matrix product of the signature matrices of the two signatures.*

Now we can prove our hardness result.

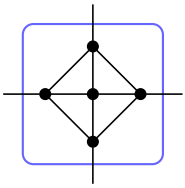


Fig. 1. The planar tetrahedron gadget. Each vertex is assigned $[3, 0, 1, 0, 3]$.

Theorem 8. #Eulerian-Orientations *is* #P*-hard on planar 4-regular graphs.*

Proof. We reduce from calculating the right-hand side of (2) to Pl-Holant($\neq_2$ | $[0, 0, 1, 0, 0]$). The bipartite Holant problem Pl-Holant($\neq_2$ | f) expresses the right-hand side of (2), where the signature matrix of f is $M_f = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$. A holographic transformation by $Z = \begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix}$ transforms Pl-Holant($\neq_2$ | f) to Pl-Holant($\hat{f}$), where the signature matrix of $\hat{f}$ is $M_{\hat{f}} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix}$. We also perform a holographic transformation by Z on our target problem Pl-Holant($\neq_2$ | $[0, 0, 1, 0, 0]$) to get Pl-Holant($[3, 0, 1, 0, 3]$). Using the planar tetrahedron gadget in Figure 1, we assign $[3, 0, 1, 0, 3]$ to every vertex and obtain a gadget with signature $32\hat{g}$, where the signature matrix of $\hat{g}$ is $M_{\hat{g}} = \frac{1}{2} \begin{bmatrix} 19 & 0 & 0 & 7 \\ 0 & 7 & 5 & 0 \\ 0 & 5 & 7 & 0 \\ 7 & 0 & 0 & 19 \end{bmatrix}$.

Now we show how to reduce Pl-Holant($\hat{f}$) to Pl-Holant($\hat{g}$) by interpolation. Consider an instance Ω of Pl-Holant($\hat{f}$). Suppose that $\hat{f}$ appears n times in Ω. We construct from Ω a sequence of instances Ω_s of Holant($\hat{g}$) indexed by $s \geq 1$. We obtain Ω_s from Ω by replacing each occurrence of $\hat{f}$ with the gadget N_s in Figure 2 with $\hat{g}$ assigned to all vertices. Although $\hat{f}$ and $\hat{g}$ are asymmetric signatures, they are invariant under a cyclic permutation of their inputs. Thus, it is unnecessary to specify which edge corresponds to which input. We call such signatures *rotationally symmetric*.

To obtain Ω_s from Ω, we effectively replace $M_{\hat{f}}$ with $M_{N_s} = (M_{\hat{g}})^s$, the sth power of the signature matrix $M_{\hat{g}}$. Let $T = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$. Then $M_{\hat{f}} = T\Lambda_{\hat{f}}T^{-1} = T \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} T^{-1}$ and $M_{\hat{g}} = T\Lambda_{\hat{g}}T^{-1} = T \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 13 \end{bmatrix} T^{-1}$. We can view our con-

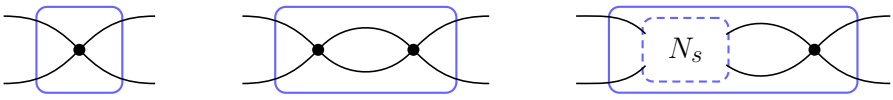


Fig. 2. Recursive construction to interpolate $\hat{f}$. The vertices are assigned $\hat{g}$.

struction of Ω_s as first replacing each $M_{\hat{f}}$ by $T\Lambda_{\hat{f}}T^{-1}$, which does not change the Holant value, and then replacing each $\Lambda_{\hat{f}}$ with $\Lambda_{\hat{g}}^s$. We stratify the assignments in Ω based on the assignment to $\Lambda_{\hat{f}}$. We only need to consider the assignments to $\Lambda_{\hat{f}}$ that assign 0000 j many times, 0110 or 1001 k many times, and 1111 ℓ many times. Let $c_{jk\ell}$ be the sum over all such assignments of the products of evaluations (including the contributions from T and T^{-1}) on Ω. Then Pl-Holant$_\Omega$ and Pl-Holant$_{\Omega_s}$, for $s \geq 1$, can be expressed as Pl-Holant$_\Omega = \sum_{j+k+\ell=n} 3^\ell c_{jk\ell}$ and Pl-Holant$_{\Omega_s} = \sum_{j+k+\ell=n} (6^k 13^\ell)^s c_{jk\ell}$. This coefficient matrix in the linear system involving Pl-Holant$_{\Omega_s}$ is Vandermonde and of full rank since for any $0 \leq k+\ell \leq n$ and $0 \leq k'+\ell' \leq n$ such that $(k,\ell) \neq (k',\ell')$, $6^k 13^\ell \neq 6^{k'} 13^{\ell'}$. Therefore, we can solve the linear system for the unknown $c_{jk\ell}$'s and obtain the value of Holant$_\Omega$. □

The previous proof can be easily modified to reduce from #EO over 4-regular graphs by interpolating the so-called crossover signature. Conceptually, the current proof is simpler because the #P-hardness proof for #EO over 4-regular graphs in [31] reduces from the same starting point as our current proof.

It was shown that any symmetric signature with a rank 3 signature matrix defines a #P-hard Holant problem (see Corollary 5.7 [32]). The only nonplanar part of the proof is that the initial problem in the reduction, counting Eulerian orientations over 4-regular graphs, was not known to be #P-hard when the input is also input planar. The reductions themselves were all planar. Here we have shown the #P-hardness of this problem under the planarity restriction in Theorem 8, and therefore obtain the planar version of Corollary 5.7 in [32].

Corollary 9. *For a symmetric arity 4 signature* $[f_0, f_1, f_2, f_3, f_4]$ *with complex weights, if there does* not *exist* $a, b, c \in \mathbb{C}$*, not all zero, such that for all* $k \in \{0,1,2\}$*,* $af_k + bf_{k+1} + cf_{k+2} = 0$*, then* Pl-Holant$(f)$ *is* #P*-hard.*

With Corollary 9, only one obstacle remains in proving a dichotomy for a symmetric arity 4 signature in the Pl-Holant framework: the case $[v, 1, 0, 0, 0]$ when $v \neq 0$. We handle this by a reduction from Pl-Holant$([v, 1, 0, 0])$, which is #P-hard over planar graphs for $v \neq 0$. These problems are the weighted versions of counting matchings over planar k-regular graphs for $k = 4, 3$ respectively. This proof uses a refined interpolation technique. A planar $\mathcal{F}$-gate is essentially a planar graph gadget where the vertices are assigned signatures in $\mathcal{F}$.

Lemma 10. *Let* $\mathcal{F}$ *be a set of signatures. If there exists a planar* $\mathcal{F}$*-gate with signature matrix* $M \in \mathbb{C}^{2\times 2}$ *and a planar* $\mathcal{F}$*-gate with signature* $s \in \mathbb{C}^{2\times 1}$ *such that (1)* $\det(M) \neq 0$*, (2)* $\det([s\ Ms]) \neq 0$*, and (3)* M *has infinite order modulo a scalar, then* Pl-Holant$(\mathcal{F} \cup \{[a, b]\}) \leq_T$ Pl-Holant$(\mathcal{F})$ *for any* $a, b \in \mathbb{C}$*.*

The refinement is in the third condition. Previous work [35, 27, 29] used a stronger third condition: *the ratio of the eigenvalues of* M *is not a root of unity.* The first two conditions of Lemma 10 are easy to check. Our third condition holds in one of these two cases: either the eigenvalues are the same but M is not a multiple of the identity matrix, or the eigenvalues are different but their

ratio is not a root of unity. The power of this lemma is that when our third condition fails to hold, we can construct M^{-1} by some constant number of copies of M and use this in other gadget constructions. This is called the anti-gadget technique [25]. We use this interpolation lemma or the anti-gadget technique to realize $[1, 0, 0]$.

To effectively use $[1, 0, 0]$, we introduce the notion of a planar pairing.

Definition 11 (Planar pairing). *A* planar pairing *in a graph $G = (V, E)$ is a set of edges $P \subset V \times V$ such that P is a perfect matching in the graph $(V, V \times V)$, and the graph $(V, E \cup P)$ is planar.*

Lemma 12. *For any planar 3-regular graph G, there exists a planar pairing that can be computed in polynomial time.*

With this lemma, we may use $[1, 0, 0]$ as $[1, 0]$ on every vertex of a planar 3-regular graph, and obtain the hardness of the weighted versions of counting matchings over planar 4-regular graphs.

Lemma 13. *If $v \in \mathbb{C} - \{0\}$, then* Pl-Holant$([v, 1, 0, 0, 0])$ *is* #P*-hard.*

Combining Corollary 9 and Lemma 12 with Theorem 22 in [36], we can prove our Pl-Holant dichotomy for a symmetric arity 4 signature. A signature is vanishing if the Holant is always 0 [32].

Theorem 14. *If f is a non-degenerate, symmetric, complex-valued signature of arity 4 in Boolean variables, then* Pl-Holant(f) *is* #P*-hard unless f is $\mathscr{A}$-transformable or $\mathscr{P}$-transformable or vanishing or $\mathscr{M}$-transformable, in which case the problem is in* P*.*

4 Pinning for Planar Graphs

The idea of "pinning" is a common reduction technique between counting problems. For the #CSP framework, pinning fixes some variables to specific values of the domain by means of the constant functions [37, 30, 38, 31]. For counting graph homomorphisms, pinning is used when the input graph is connected and the target graph is disconnected. Pinning a vertex of the input graph to a vertex of the target graph forces all the vertices of the input graph to map to the same connected component of the target graph [39–42]. In the Boolean domain, the constant 0 and 1 functions are the signatures $[1, 0]$ and $[0, 1]$ respectively.

From these works, the most relevant pinning lemma for the Pl-#CSP framework is by Dyer, Goldberg, and Jerrum in [30], where they show how to pin in the #CSP framework. However, the proof of this pinning lemma is highly nonplanar. Cai, Lu, and Xia [23] overcame this difficultly in the proof of their dichotomy theorem for the real-weighted Pl-#CSP framework by first undergoing a holographic transformation by the Hadamard matrix $H = \left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$ and then pinning in this Hadamard basis.[2] We stress that this holographic transformation is necessary. Indeed, if one were able to pin in the standard basis of the

[2] The pinning in [23], which is accomplished in Section IV, is not summarized in a single statement but is implied by the combination of all the results in that section.

Pl-#CSP framework, then P = #P would follow since Pl-#CSP($\widehat{\mathscr{M}}$) is tractable but Pl-#CSP($\widehat{\mathscr{M}} \cup \{[1,0],[0,1]\}$) is #P-hard by our main result, Theorem 19.

Since Pl-#CSP($\mathcal{F}$) is Turing equivalent to Pl-Holant($\mathcal{F} \cup \mathcal{EQ}$), its expression in the Hadamard basis is Pl-Holant($H\mathcal{F} \cup \widehat{\mathcal{EQ}}$). As $[1,0] \in \widehat{\mathcal{EQ}}$, pinning in this Hadamard basis amounts to obtaining the missing signature $[0,1]$.

Theorem 15 (Pinning). *Let $\mathcal{F}$ be any set of complex-weighted symmetric signatures. Then* Pl-Holant$^c(\mathcal{F} \cup \widehat{\mathcal{EQ}})$ *is* #P*-hard (or in* P*) iff* Pl-Holant$(\mathcal{F} \cup \widehat{\mathcal{EQ}})$ *is* #P*-hard (or in* P*).*

This theorem does not exclude the possibility that either framework can express a problem of intermediate complexity. It merely says that if one framework does not contain a problem of intermediate complexity, then neither does the other. Our goal is to prove a dichotomy for Pl-Holant($\mathcal{F} \cup \widehat{\mathcal{EQ}}$). By Theorem 15, this is equivalent to proving a dichotomy for Pl-Holant$^c(\mathcal{F} \cup \widehat{\mathcal{EQ}})$.

In Theorem 15, the difference between the two counting problems is the presence of $[0,1]$ in the first problem. The proof is quite involved and can be found in the full version of this paper [1]. It is proved in several steps under various assumptions on $\mathcal{F}$. Each of these steps is proved in one of three ways:

1. either the first problem is tractable (so the second problem is as well);
2. or the second problem is #P-hard (so the first problem is as well);
3. or the first problem reduces to the second problem by constructing $[0,1]$ using the signatures from the second problem.

5 Main Dichotomy

Our main dichotomy theorem relies on a dichotomy for a single signature.

Theorem 16. *If f is a non-degenerate symmetric signature of arity at least 2 with complex weights in Boolean variables, then* Pl-Holant$(\{f\} \cup \widehat{\mathcal{EQ}})$ *is* #P*-hard unless $f \in \mathscr{A} \cup \widehat{\mathscr{P}} \cup \mathscr{M}$, in which case the problem is in* P*.*

We also prove a useful result that we call the Mixing Theorem.

Theorem 17 (Mixing). *Let $\mathcal{F}$ be any set of symmetric, complex-valued signatures. If $\mathcal{F} \subseteq \mathscr{A} \cup \widehat{\mathscr{P}} \cup \mathscr{M}$, then* Pl-Holant$(\mathcal{F} \cup \widehat{\mathcal{EQ}})$ *is* #P*-hard unless $\mathcal{F} \subseteq \mathscr{A}$, $\mathcal{F} \subseteq \widehat{\mathscr{P}}$, or $\mathcal{F} \subseteq \mathscr{M}$.*

By Theorems 15, 16, and 17, the proof of our main theorem is straightforward.

Theorem 18. *Let $\mathcal{F}$ be any set of symmetric, complex-valued signatures in Boolean variables. Then* Pl-Holant$(\mathcal{F} \cup \widehat{\mathcal{EQ}})$ *is* #P*-hard unless $\mathcal{F} \subseteq \mathscr{A}$, $\mathcal{F} \subseteq \widehat{\mathscr{P}}$, or $\mathcal{F} \subseteq \mathscr{M}$, in which case the problem is in* P*.*

We also have the corresponding theorem for the Pl-#CSP framework in the standard basis, which is equivalent to Theorem 1.

Theorem 19. *Let $\mathcal{F}$ be any set of symmetric, complex-valued signatures in Boolean variables. Then* Pl-#CSP$(\mathcal{F})$ *is* #P*-hard unless $\mathcal{F} \subseteq \mathscr{A}$, $\mathcal{F} \subseteq \mathscr{P}$, or $\mathcal{F} \subseteq \widehat{\mathscr{M}}$, in which case the problem is in* P*.*

Acknowledgements. Both authors were supported by NSF CCF-0914969 and NSF CCF-1217549. We are very grateful to Jin-Yi Cai for his support, guidance, and discussions. We also thank him for his careful reading and insightful comments on a draft of this work. We thank Peter Bürgisser, Leslie Ann Goldberg, Mark Jerrum, and Pascal Koiran for the invitation to present this work at the Dagstuhl seminar on computational counting as well as all those at the seminar for their interest. We also thank the anonymous referees for their helpful comments.

References

1. Guo, H., Williams, T.: The complexity of planar Boolean #CSP with complex weights. CoRR abs/1212.2284 (2012)
2. Valiant, L.G.: The complexity of computing the permanent. TCS 8(2), 189–201 (1979)
3. Kasteleyn, P.W.: Graph theory and crystal physics. In: Harary, F. (ed.) Graph Theory and Theoretical Physics, pp. 43–110. Academic Press, London (1967)
4. Ising, E.: Beitrag zür theorie des ferromagnetismus. Zeitschrift für Physik 31(1), 253–258 (1925)
5. Onsager, L.: Crystal statistics. I. A two-dimensional model with an order-disorder transition. Phys. Rev. 65(3-4), 117–149 (1944)
6. Yang, C.N.: The spontaneous magnetization of a two-dimensional Ising model. Phys. Rev. 85(5), 808–816 (1952)
7. Yang, C.N., Lee, T.D.: Statistical theory of equations of state and phase transitions. I. Theory of condensation. Phys. Rev. 87(3), 404–409 (1952)
8. Lee, T.D., Yang, C.N.: Statistical theory of equations of state and phase transitions. II. Lattice gas and Ising model. Phys. Rev. 87(3), 410–419 (1952)
9. Temperley, H.N.V., Fisher, M.E.: Dimer problem in statistical mechanics—an exact result. Philosophical Magazine 6(68), 1061–1063 (1961)
10. Kasteleyn, P.W.: The statistics of dimers on a lattice. Physica 27(12), 1209–1225 (1961)
11. Baxter, R.J.: Exactly solved models in statistical mechanics. Academic Press, London (1982)
12. Lieb, E.H., Sokal, A.D.: A general Lee-Yang theorem for one-component and multicomponent ferromagnets. Comm. Math. Phys. 80(2), 153–179 (1981)
13. Welsh, D.: Complexity: Knots, Colourings and Countings. London Mathematical Society Lecture Note Series. Cambridge University Press (1993)
14. Valiant, L.G.: Quantum circuits that can be simulated classically in polynomial time. SIAM J. Comput. 31(4), 1229–1254 (2002)
15. Valiant, L.G.: Expressiveness of matchgates. TCS 289(1), 457–471 (2002)
16. Valiant, L.G.: Holographic algorithms. SIAM J. Comput. 37(5), 1565–1594 (2008)
17. Valiant, L.G.: Accidental algorthims. In: FOCS, pp. 509–517. IEEE Computer Society (2006)
18. Cai, J.Y., Choudhary, V.: Some results on matchgates and holographic algorithms. Int. J. Software and Informatics 1(1), 3–36 (2007)

19. Cai, J.Y., Choudhary, V., Lu, P.: On the theory of matchgate computations. Theory of Computing Systems 45(1), 108–132 (2009)
20. Cai, J.Y., Lu, P.: On symmetric signatures in holographic algorithms. Theory of Computing Systems 46(3), 398–415 (2010)
21. Cai, J.Y., Lu, P.: Holographic algorithms: From art to science. J. Comput. Syst. Sci. 77(1), 41–61 (2011)
22. Vertigan, D.: The computational complexity of Tutte invariants for planar graphs. SIAM J. Comput. 35(3), 690–712 (2005)
23. Cai, J.Y., Lu, P., Xia, M.: Holographic algorithms with matchgates capture precisely tractable planar #CSP. In: FOCS, pp. 427–436. IEEE Computer Society (2010)
24. Cai, J.Y., Kowalczyk, M.: Spin systems on k-regular graphs with complex edge functions. TCS 461, 2–16 (2012)
25. Cai, J.Y., Kowalczyk, M., Williams, T.: Gadgets and anti-gadgets leading to a complexity dichotomy. In: ITCS, pp. 452–467. ACM (2012)
26. Cai, J.Y., Lu, P., Xia, M.: Holographic algorithms by Fibonacci gates. Linear Algebra and its Applications 438(2), 690–707 (2013)
27. Cai, J.Y., Lu, P., Xia, M.: Holographic reduction, interpolation and hardness. Computational Complexity 21(4), 573–604 (2012)
28. Cai, J.Y., Lu, P., Xia, M.: Holant problems and counting CSP. In: STOC, pp. 715–724. ACM (2009)
29. Cai, J.Y., Lu, P., Xia, M.: Computational complexity of Holant problems. SIAM J. Comput. 40(4), 1101–1132 (2011)
30. Dyer, M., Goldberg, L.A., Jerrum, M.: The complexity of weighted Boolean CSP. SIAM J. Comput. 38(5), 1970–1986 (2009)
31. Huang, S., Lu, P.: A dichotomy for real weighted Holant problems. In: IEEE Conference on Computational Complexity, pp. 96–106. IEEE Computer Society (2012)
32. Cai, J.Y., Guo, H., Williams, T.: A complete dichotomy rises from the capture of vanishing signatures. CoRR abs/1204.6445 (2012); STOC 2013 (to appear)
33. Las Vergnas, M.: Eulerian circuits of 4-valent graphs imbedded in surfaces. In: Lovász, L., Sós, V.T. (eds.) Algebraic Methods in Graph Theory. Colloq. Math. Soc. János Bolyai, pp. 451–477. North-Holland (1981)
34. Las Vergnas, M.: On the evaluation at (3, 3) of the Tutte polynomial of a graph. J. Comb. Theory, Ser. B 45(3), 367–372 (1988)
35. Vadhan, S.P.: The complexity of counting in sparse, regular, and planar graphs. SIAM J. Comput. 31(2), 398–427 (2001)
36. Kowalczyk, M.: Dichotomy theorems for Holant problems. PhD thesis, University of Wisconsin—Madison (2010)
37. Bulatov, A.A., Dalmau, V.: Towards a dichotomy theorem for the counting constraint satisfaction problem. Information and Computation 205(5), 651–678 (2007)
38. Bulatov, A., Dyer, M., Goldberg, L.A., Jalsenius, M., Richerby, D.: The complexity of weighted boolean #CSP with mixed signs. TCS 410(38-40), 3949–3961 (2009)
39. Dyer, M., Greenhill, C.: The complexity of counting graph homomorphisms. Random Struct. Algorithms 17(3-4), 260–289 (2000)
40. Bulatov, A., Grohe, M.: The complexity of partition functions. TCS 348(2), 148–186 (2005)
41. Goldberg, L.A., Grohe, M., Jerrum, M., Thurley, M.: A complexity dichotomy for partition functions with mixed signs. SIAM J. Comput. 39(7), 3336–3402 (2010)
42. Cai, J.Y., Chen, X., Lu, P.: Graph homomorphisms with complex values: A dichotomy theorem. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 275–286. Springer, Heidelberg (2010)

Arthur-Merlin Streaming Complexity*

Tom Gur and Ran Raz**

Weizmann Institute of Science

Abstract. We study the power of Arthur-Merlin probabilistic proof systems in the data stream model. We show a canonical $\mathcal{AM}$ streaming algorithm for a wide class of data stream problems. The algorithm offers a tradeoff between the length of the proof and the space complexity that is needed to verify it.

As an application, we give an $\mathcal{AM}$ streaming algorithm for the *Distinct Elements* problem. Given a data stream of length m over alphabet of size n, the algorithm uses $\tilde{O}(s)$ space and a proof of size $\tilde{O}(w)$, for every s, w such that $s \cdot w \geq n$ (where $\tilde{O}$ hides a polylog(m, n) factor). We also prove a lower bound, showing that every $\mathcal{MA}$ streaming algorithm for the *Distinct Elements* problem that uses s bits of space and a proof of size w, satisfies $s \cdot w = \Omega(n)$.

As a part of the proof of the lower bound for the *Distinct Elements* problem, we show a new lower bound of $\Omega(\sqrt{n})$ on the $\mathcal{MA}$ communication complexity of the *Gap Hamming Distance* problem, and prove its tightness.

Keywords: Probabilistic Proof Systems, Data Streams, Communication Complexity.

1 Introduction

The data stream computational model is an abstraction commonly used for algorithms that process network traffic using sublinear space [2,14,5]. In the settings of this model, we have an algorithm that gets a sequence of elements (typically, each element is an integer) as input. This sequence of elements is called a *data stream* and is usually denoted by $\sigma = (a_1, \ldots, a_m)$; where a_1 is the first element, a_2 is the second element, and so forth. The algorithm receives its input (a data stream) element-by-element. After it sees each a_i, it no longer has an access to elements with index that is smaller than i. The algorithm is required to compute a function of the data stream, using as little space as possible.

Among the most fundamental problems in the data stream model is the problem of *Distinct Elements*, i.e., the problem of computing the number of distinct elements in a given data stream. The problem has been studied extensively in the

* This paper is an extended abstract of [13].

** Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: {tom.gur, ran.raz}@weizmann.ac.il. Research supported by an Israel Science Foundation grant and by the I-CORE Program of the Planning and Budgeting Committee and the Israel Science Foundation.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 528–539, 2013.

last two decades (see, for example, [2,14,15]). Its significance stems both from the vast variety of applications that it spans (covering IP routing, database operations and text compression, e.g., [16,2,11]), and due to the theoretical insight that it gives on the nature of computation in the data stream model.

Alon et al. [2] have shown a lower bound of $\Omega(n)$ (where n is the size of the alphabet from which the elements are taken) on the streaming complexity of the computation of the *exact* number of distinct elements in a sufficiently long data stream (i.e., where the length of the data stream is at least proportional to n). The goal of reducing the space complexity of the *Distinct Elements* problem has led to a long line of research of approximation algorithms for the problem, starting with the seminal paper [10] by Flajolet and Martin. Recently, Kane at el. [15] gave the first optimal approximation algorithm for estimating the number of distinct elements in a data stream; for a data stream with alphabet of size n, given $\epsilon > 0$ their algorithm computes a $(1 \pm \epsilon)$ multiplicative approximation using $O(\epsilon^{-2} + \log n)$ bits of space, with $2/3$ success probability.

A natural approach for reducing the space complexity of streaming algorithms, *without* settling on an approximation, is by considering a probabilistic proof system. Chakrabarti at el. [5] have shown *data stream with annotations* algorithms for several data stream problems, using a probabilistic proof system that is very similar to $\mathcal{MA}$. This line of work continued in [8], wherein a probabilistic proof system was used in order to reduce the streaming complexity of numerous graph problems. In a subsequent work [9], Cormode at el. provided a practical instantiation of one of the most efficient general-purpose construction of an interactive proof for arbitrary computations, due to Goldwasser et al. [12].

In this work, we study the power of Arthur-Merlin probabilistic proof systems in the data stream model. We show a canonical $\mathcal{AM}$ streaming algorithm for a wide class of data stream problems. The algorithm offers a tradeoff between the length of the proof and the space complexity that is needed to verify it. We show that the problem of *Distinct Elements* falls within the class of problems that our canonical algorithm can handle. Thus, we give an $\mathcal{AM}$ streaming algorithm for the *Distinct Elements* problem. Given a data stream of length m over alphabet of size n, the algorithm uses $\tilde{O}(s)$ space and a proof of size $\tilde{O}(w)$, for every s, w such that $s \cdot w \geq n$ (where throughout the paper $\tilde{O}$ hides a polylog(m, n) factor).

In addition, we give a lower bound on the $\mathcal{MA}$ streaming complexity of the *Distinct Elements* problem. Our lower bound for *Distinct Elements* relies on a new lower bound that we prove on the $\mathcal{MA}$ communication complexity of the *Gap Hamming Distance* problem.

1.1 Arthur-Merlin Probabilistic Proof Systems

An $\mathcal{MA}$ (Merlin-Arthur) proof is a probabilistic extension of the notion of proof in complexity theory. Proofs of this type are commonly described as an interaction between two players, usually referred to as Merlin and Arthur. We think of Merlin as an omniscient prover, and of Arthur as a computationally bounded verifier. Merlin is supposed to send Arthur a valid proof for the correctness of a certain statement. After seeing both the input and Merlin's proof, with high

probability Arthur can verify a valid proof for a correct statement, and reject every possible alleged proof for a wrong statement.

An $\mathcal{AM}$ proof is defined almost the same as an $\mathcal{MA}$ proof, except that in $\mathcal{AM}$ proof systems we assume that both the prover and the verifier have access to a common source of randomness (alternatively, $\mathcal{AM}$ proof systems can be described as $\mathcal{MA}$ proof systems that start with an extra round, wherein Arthur sends Merlin a random string).

The notion of $\mathcal{AM}$ and $\mathcal{MA}$ proof systems can be extended to many computational models. In this work we consider both the communication complexity analogue of $\mathcal{MA}$, wherein Alice and Bob receive a proof that they use in order to save communication, and the data stream analogues of $\mathcal{MA}$ and $\mathcal{AM}$, wherein the data stream algorithm receives a proof and uses it in order to reduce the required resources for solving a data stream problem.

Recently, probabilistic proof systems for streaming algorithms have been used to provide an abstraction of the notion of *delegation of computation* to a cloud (see [8,9,7]). In the context of cloud computing, a common scenario is one where a user receives or generates a massive amount of data, which he cannot afford to store locally. The user can stream the data he receives to the cloud, keeping only a short certificate of the data he streamed. Later, when the user wants to calculate a function of that data, the cloud can perform the calculations and send the result to the user. However, the user cannot automatically trust the cloud (as an error could occur during the computation, or the service provider might not be honest). Thus the user would like to use the short certificate that he saved in order to verify the answer that he gets from the cloud.

1.2 Communication Complexity and the *Gap Hamming Distance* Problem

Communication complexity is a central model in computational complexity. In its basic setup, we have two computationally unbounded players, Alice and Bob, holding (respectively) binary strings x, y of length n each. The players need to compute a function of both of the inputs, using the least amount of communication between them.

In this work we examine the well known communication complexity problem of *Gap Hamming Distance* (GHD), wherein each of the two parties gets an n bit binary string, and together the parties need to tell whether the Hamming distance of the strings is larger than $\frac{n}{2} + \sqrt{n}$ or smaller than $\frac{n}{2} - \sqrt{n}$ (assuming that one of the possibilities occurs). In [6] a tight linear lower bound was proven on the communication complexity of a randomized communication complexity protocol for GHD. Following [6], a couple of other proofs ([21,19]) were given for the aforementioned lower bound. Relying on [19], in this work we give a tight lower bound of $\Omega(\sqrt{n})$ on the $\mathcal{MA}$ communication complexity of GHD.

1.3 Our Results

The main contributions in this work are:

1. A canonical $\mathcal{AM}$ streaming algorithm for a wide class of data stream problems, including the *Distinct Elements* problem.
2. A lower bound on the $\mathcal{MA}$ streaming complexity of the *Distinct Elements* problem.
3. A tight lower bound on the $\mathcal{MA}$ communication complexity of the *Gap Hamming Distance* problem.

In order to state the results precisely, we first introduce the following notations: given a data stream $\sigma = (a_1, \ldots, a_m)$ (over alphabet $[n]$), the *element indicator* $\chi_i : [n] \to \{0,1\}$ of the i'th element ($i \in [m]$) of the stream σ, is the function that indicates whether a given element is in position $i \in [m]$ of σ, i.e., $\chi_i(j) = 1$ if and only if $a_i = j$. Furthermore, let $\chi : [n] \to \{0,1\}^m$ be the *element indicator* of σ, defined by

$$\chi(j) = \big(\chi_1(j), \ldots, \chi_m(j)\big).$$

In addition, given $m \in \mathbb{N}$ we define a *clause* over m variables $x_1, \ldots, x_m$ as a function $C : \{0,1\}^m \to \{0,1\}$ of the form $(y_1 \vee y_2 \vee \ldots \vee y_m)$, where for every $i \in [m]$ the literal y_i is either a variable (x_j), a negation of a variable ($\neg x_j$), or one of the constants $\{0,1\}$.

Equipped with the notations above, we formally state our results. Let $0 \leq \epsilon < 1/2$. Let $\mathcal{P}$ be a data stream problem such that for every $m, n \in \mathbb{N}$ there exists a set of $k = k(m,n)$ clauses $\{C_t\}_{t\in[k]}$ over m variables, and a function $\psi : \{0,1\}^k \to \mathbb{Z}$, such that for every data stream $\sigma = (a_1, \ldots, a_m)$ with alphabet $[n]$,

$$(1-\epsilon)\mathcal{P}(\sigma) \leq \sum_{j=1}^{n} \psi\big(C_1 \circ \chi(j), \ldots, C_k \circ \chi(j)\big) \leq (1+\epsilon)\mathcal{P}(\sigma).$$

Moreover, we assume that ψ and $\{C_t\}_{t\in[k]}$ are known to the verifier[1], and that there exists $B \leq \text{poly}(m,n)$ such that $\psi(x) < B$ for every $x \in \{0,1\}^k$. Given such $\mathcal{P}$, for every $0 < \delta \leq 1$ and every $s, w \in \mathbb{N}$ such that $s \cdot w \geq n$, we give an $\mathcal{AM}$ streaming algorithm, with error probability δ, for approximating $\mathcal{P}(\sigma)$ within a multiplicative factor of $1 \pm \epsilon$. The algorithm uses space $O\big(sk \cdot \text{polylog}(m, n, \delta^{-1})\big)$, a proof of size $W = O\big(wk \cdot \text{polylog}(m, n, \delta^{-1})\big)$, and randomness complexity $\text{polylog}(m, n, \delta^{-1})$.

We show that the aforementioned algorithm, when applied to the *Distinct Elements* problem with parameters s, w such that $s \cdot w \geq n$, yields an $\mathcal{AM}$ streaming algorithm for the problem. The algorithm computes, with probability at least $2/3$, the exact number of distinct elements in a data stream of length m over alphabet $[n]$, using space $\tilde{O}(s)$ and a proof of size $\tilde{O}(w)$. For example, by fixing $w = n$, we get an $\mathcal{AM}$ streaming algorithm for the *Distinct Elements* problem that uses only polylogarithmic space.

[1] For example, ψ and $\{C_t\}_{t\in[k]}$ can be $\text{polylog}(m,n)$-space uniform; that is, the *description* of ψ and $\{C_t\}_{t\in[k]}$ can be computed by a deterministic Turing machine that runs in $\text{polylog}(m,n)$ space.

We note that an interesting special case of the class of problems that our canonical $\mathcal{AM}$ streaming algorithm handles can also be stated in terms of Boolean circuits, instead of clauses. That is, given $0 \leq \epsilon < 1/2$ and a data stream problem $\mathcal{P}$ such that for every $m, n \in \mathbb{N}$ there exists an unbounded fan-in Boolean circuit $C : \{0,1\}^m \to \{0,1\}$ with $k = k(m,n)$ non-input gates, such that for every data stream $\sigma = (a_1, \ldots, a_m)$ with alphabet $[n]$,

$$(1-\epsilon)\mathcal{P}(\sigma) \leq \sum_{j=1}^{n} C\big(\chi_1(j), \ldots, \chi_m(j)\big) \leq (1+\epsilon)\mathcal{P}(\sigma).$$

Assuming that C is known to the verifier, we get an $\mathcal{AM}$ streaming algorithm for $\mathcal{P}$ with the same parameters as in the original formulation of the canonical $\mathcal{AM}$ algorithm above.

Our next result is a lower bound on the $\mathcal{MA}$ streaming complexity of the *Distinct Elements* problem. We show that every $\mathcal{MA}$ streaming algorithm that approximates, within a multiplicative factor of $1 \pm 1/\sqrt{n}$, the number of distinct elements in a data stream of length m over alphabet $[n]$ (where $m > n$), using s bits of space and a proof of size w, must satisfy $s \cdot w = \Omega(n)$.

Last, we show a tight (up to a logarithmic factor) lower bound on the $\mathcal{MA}$ communication complexity of the *Gap Hamming Distance* problem. For every $\mathcal{MA}$ communication complexity protocol for GHD that communicates t bits and uses a proof of size w, we have $t \cdot w = \Omega(n)$. We prove the tightness of the lower bound by giving, for every $t, w \in \mathbb{N}$ such that $t \cdot w \geq n$, an $\mathcal{MA}$ communication complexity protocol for GHD, which communicates $O(t \log n)$ bits and uses a proof of size $O(w \log n)$.

For the full formal proofs, we refer the reader to the full version [13].

1.4 Related Work

The data stream model has gained a great deal of attention after the publication of the seminal paper by Alon, Matias and Szegedy [2]. In the scope of that work, the authors have shown a lower bound of $\Omega(n)$ (where n is the size of the alphabet) on the streaming complexity of *Distinct Elements* (i.e., the computation of the *exact* number of distinct elements in a data stream) where the length of the input is at least proportional to n.

Following [2] there was a long line of theoretical research on the approximation of the *Distinct Element* problem ([14,3,4,15], see [16] for a survey of earlier results). Finally, Kane et al. [15] gave the first optimal approximation algorithm for estimating the number of distinct elements in a data stream; for a data stream with alphabet of size n, given $\epsilon > 0$ their algorithm computes a $(1 \pm \epsilon)$ multiplicative approximation using $O(\epsilon^{-2} + \log n)$ bits of space, with $2/3$ success probability. This result matches the tight lower bound of Indyk and Woodruff [14].

In a recent sequence of works, the data stream model was extended to support several interactive and non-interactive proof systems [5,8,7]. The model of streaming algorithms with non-interactive proofs was first introduced in [5] and

extended in [8,9]. In [5] the authors gave an optimal (up to polylogarithmic factors) *data stream with annotations* algorithm for computing the k'th frequency moment exactly, for every integer $k \geq 1$.

2 Preliminaries

2.1 MA Communication Complexity

In $\mathcal{MA}$ *communication complexity protocols*, we have a (possibly partial) function $f : X \times Y \to \{0, 1\}$ (for some finite sets X, Y), and three computationally unbounded parties: Merlin, Alice, and Bob. The function f is known to all parties. Alice gets as an input $x \in X$. Bob gets as an input $y \in Y$. Merlin sees both x and y. We think of Merlin as a *prover*, and think of Alice and Bob as *verifiers*. We assume that Alice and Bob share a private random string that Merlin cannot see.

At the beginning of an $\mathcal{MA}$ *communication complexity protocol*, Merlin sends a proof string w to both Alice and Bob, so both players have a free access to w. The players proceed as before. In each step of the protocol, one of the players sends one bit to the other player. At the end of the protocol, both players have to know an answer z. Hence, the answer depends on the input (x, y) as well as on the proof w. For a protocol P, denote by $P\big((x, y), w\big)$ the probabilistic answer z given by the protocol on input (x, y) and proof w.

An $\mathcal{MA}$ *communication complexity protocol* has three parameters: a limit on the probability of error of the protocol, denoted by ϵ; a limit on the number of bits of communication between Alice and Bob, denoted by T; and a limit on the length of Merlin's proof string, denoted by W.

With the above in mind, we can now define $\mathcal{MA}_\epsilon(T, W)$ communication complexity as follows:

Definition 1. *An $\mathcal{MA}_\epsilon(T, W)$-communication complexity protocol for f is a probabilistic communication complexity protocol P, as above (i.e., with an additional proof string w presented to the players). During the protocol, Alice and Bob communicate at most T bits. The protocol satisfies,*

1. ***Completeness**: for all $(x, y) \in f^{-1}(1)$, there exists a string w such that $|w| < W$, that satisfies*

$$\Pr\big[P\big((x, y), w\big) = 1\big] > 1 - \epsilon.$$

2. ***Soundness**: for all $(x, y) \in f^{-1}(0)$ and for any string w such that $|w| < W$, we have*

$$\Pr\big[P\big((x, y), w\big) = 1\big] < \epsilon.$$

2.2 Streaming Complexity

Let $\epsilon \geq 0$, $\delta > 0$. Let $m, n \in \mathbb{N}$. A *data stream* $\sigma = (a_1, \ldots, a_m)$ is a sequence of elements, each from $[n] = \{1, \ldots, n\}$. We say that the *length* of the stream is m, and the *alphabet size* is n.

A *streaming algorithm* is a space-bounded probabilistic algorithm that gets an element-by-element access to a *data stream*. After each element arrives, the algorithm can no longer access the elements that precede it. At the end of its run, the *streaming algorithm* is required to output (with high probability) a certain function of the *data stream* that it read. When dealing with *streaming algorithms*, the main resource we are concerned with is the size of the space that the algorithm uses.

Formally, a *data stream problem* $\mathcal{P}$ is a collection of functions $\{f_{m,n} : [n]^m \to \mathbb{R}\}_{m,n \in \mathbb{N}}$. That is, a function for every combination of *length* and *alphabet size* of a *data stream*. However, slightly abusing notation for the sake of brevity, we will define each *data stream problem* by a single function (which in fact depends on the length m and alphabet size n of the *data stream*). A δ-error, ϵ-approximation data stream algorithm $\mathcal{A}_{\epsilon,\delta}$ for $\mathcal{P}$ is a probabilistic algorithm that gets a sequential, one pass access to a *data stream* $\sigma = (a_1, \ldots, a_m)$ (where each a_i is a member of $[n]$), and satisfies:

$$\Pr\left[\left|\frac{\mathcal{A}_{\epsilon,\delta}(\sigma)}{f_{m,n}(\sigma)} - 1\right| > \epsilon\right] < \delta.$$

If $\epsilon = 0$ we say that the *streaming algorithm* is exact.

Last, given a data stream problem $\mathcal{P} = \{f_{m,n} : [n]^m \to \mathbb{R}\}_{m,n \in \mathbb{N}}$ and a data stream $\sigma = (a_1, \ldots, a_m)$ (with alphabet $[n]$) we denote by $\mathcal{P}(\sigma)$ the output of $f_{m,n}(\sigma)$, for the $f_{m,n} \in \mathcal{P}$ that matches the length and alphabet size of σ. Similarly, when applying a family of functions to σ, we in fact apply a specific function in the family, according to the parameters m, n of σ.

The *Distinct Elements* Problem The *Distinct Elements* problem is the problem of computing the exact number of distinct elements that appear in a data stream, denoted by $F_0(\sigma)$. Formally, we define:

Definition 2. *The* Distinct Elements *problem is the data stream problem of computing the exact number of distinct elements in a given data stream* $\sigma = (a_1, \ldots, a_m)$ *(where* $a_i \in [n]$ *for every* i*), i.e., computing (exactly):*

$$F_0(\sigma) = \left|\, \{i \in \mathbb{N} \,:\, \exists j \in [m] \;\; a_j = i\} \,\right|.$$

Note that if we define $0^0 = 0$ then this is exactly the 0'th frequency moment of the stream. Hence the notation F_0.

3 Streaming Algorithms with Probabilistic Proof Systems

In this section we extend the data stream computational model in order to support two types of probabilistic proof systems: $\mathcal{MA}$ algorithms, wherein the

streaming algorithm gets a proof that it probabilistically verifies, and $\mathcal{AM}$ algorithms that extend $\mathcal{MA}$ algorithms by adding shared randomness. We study both of these probabilistic proof systems in two variations: in the first, the proof is also being streamed to the verifier, and in the second, the verifier has a free access to the proof. Formal definitions follow.

3.1 MA Streaming Algorithms

Similarly to the way $\mathcal{MA}$ communication complexity protocols are defined, in $\mathcal{MA}$ streaming algorithms we have an omniscient prover (Merlin) who sends a proof to a verifier (Arthur), which is in fact a streaming algorithm that gets both the input stream and the proof (either by a free access or by a one-pass, sequential access). The streaming algorithm computes a function of the input stream. Using the proof we hope to achieve a better space complexity than what the regular streaming model allows.

We start with $\mathcal{MA}$ proofs wherein the proof is being streamed to the verifier. Formally, we define

Definition 3. *Let $\epsilon \geq 0$, $\delta > 0$, and let $\mathcal{P} = \{f_{m,n} : [n]^m \to \mathbb{R}\}_{m,n\in\mathbb{N}}$ be a data stream problem. An $\mathcal{MA}$ streaming algorithm for $\mathcal{P}$ is a probabilistic data stream algorithm $\mathcal{A}$, which simultaneously gets two streams: an input stream $\sigma = (a_1, \ldots, a_m)$ (where $a_i \in [n]$ for every i) and a proof stream ω; to both it has a sequential, one pass access.*[2] *Given two functions $S, W : \mathbb{N}^2 \to \mathbb{N}$, we say that an $\mathcal{MA}$ streaming algorithm is $\mathcal{MA}_{\epsilon,\delta}\big(S(m,n), W(m,n)\big)$ if it uses at most $S(m,n)$ bits of space, and satisfies:*

1. ***Completeness**: for every $\sigma = (a_1, \ldots, a_m)$ (with* alphabet $[n]$*) there exists a non empty set $\mathcal{W}_\sigma$ of proof streams of length at most $W(m,n)$, such that for every $\omega \in \mathcal{W}_\sigma$ we have,*

$$\Pr\left[\left|\frac{\mathcal{A}(\sigma,\omega)}{f_{m,n}(\sigma)} - 1\right| \leq \epsilon\right] > 1 - \delta$$

2. ***Soundness**: for every $\sigma = (a_1, \ldots, a_m)$ (with* alphabet $[n]$*), and for every $\omega \notin \mathcal{W}_\sigma$ we have*

$$\Pr[\mathcal{A}(\sigma,\omega) \neq \perp] < \delta$$

where $\perp \notin \mathbb{R}$ is a symbol that represents that the algorithm could not verify the correctness of the proof.

The second natural way to define an $\mathcal{MA}$ probabilistic proof system for the data stream model, is by allowing the algorithm a free access to the proof. We denote

[2] We note that one may consider several possible ways to implement the mechanism of accessing two streams simultaneously. However, since for all of our algorithms the order does not matter — for simplicity, we assume that first the proof stream is being received, and then the input stream is being received.

this model by $\widehat{\mathcal{MA}}$ streaming complexity. Note that by definition, the model of $\mathcal{MA}$ streaming with a free access to the proof is stronger than the model of $\mathcal{MA}$ streaming with a proof stream. In this work we prove lower bounds on the $\widehat{\mathcal{MA}}$ streaming complexity, hence it also implies lower bounds on the $\mathcal{MA}$ streaming complexity.

3.2 AM Streaming Algorithms

We can further extend the data stream model to support an $\mathcal{AM}$ probabilistic proof system. Similarly to the case of $\mathcal{MA}$ proofs, an $\mathcal{AM}$ streaming algorithm receives a proof stream and an input stream, to which it has a sequential, one pass access; except that in $\mathcal{AM}$ proof systems the prover and verifier also share a common random string. Formally, we define

Definition 4. *Let $\epsilon \geq 0$, $\delta > 0$, and let $\mathcal{P} = \{f_{m,n} : [n]^m \to \mathbb{R}\}_{m,n\in\mathbb{N}}$ be a data stream problem. An $\mathcal{AM}$ streaming algorithm for $\mathcal{P}$ is a probabilistic data stream algorithm $\mathcal{A}^r$ that has an oracle access to a common random string r, and that is also allowed to make private random coin tosses. The algorithm simultaneously gets two streams: an input stream $\sigma = (a_1, \ldots, a_m)$ (where $a_i \in [n]$ for every i) and a proof stream ω, to both it has a sequential, one pass access. Given two functions $S, W : \mathbb{N}^2 \to \mathbb{N}$, we say that an $\mathcal{AM}$ streaming algorithm is $\mathcal{AM}_{\epsilon,\delta}\big(S(m,n), W(m,n)\big)$ if it uses at most $S(m,n)$ bits of space, and satisfies that for every $\sigma = (a_1, \ldots, a_m)$ (over* alphabet $[n]$*), with probability at least $1 - \delta/2$ (over r) there exists a non empty set $\mathcal{W}_\sigma(r)$ of proof streams of length at most $W(m,n)$, such that:*

1. ***Completeness**: For every $\omega \in \mathcal{W}_\sigma(r)$*

$$\Pr\left[\left|\frac{\mathcal{A}^r(\sigma,\omega)}{f_{m,n}(\sigma)} - 1\right| \leq \epsilon\right] > 1 - \frac{\delta}{2},$$

   where the probability is taken over the private random coin tosses of $\mathcal{A}^r$.
2. ***Soundness**: For $\omega \notin \mathcal{W}_\sigma(r)$*

$$\Pr\left[\mathcal{A}^r(\sigma,\omega) = \perp\right] > 1 - \frac{\delta}{2},$$

   where the probability is taken over the private random coin tosses of $\mathcal{A}^r$, and $\perp \notin \mathbb{R}$ is a symbol that represents that the algorithm could not verify the correctness of the proof.

The randomness complexity *of the algorithm is the total size of the common random string r, and the number of private random coin tosses that the algorithms performs.*

Note that we slightly deviate from the standard definition of an $\mathcal{AM}$ algorithm, by allowing $\mathcal{A}$ to be a probabilistic algorithm with a private random string.

Just as with the $\mathcal{MA}$ streaming model, we can define $\widehat{\mathcal{AM}}$ streaming algorithms by allowing a free access to the proof. Again, by definition the model

of $\mathcal{AM}$ streaming with a free access to the proof is stronger than the model of $\mathcal{AM}$ streaming with a proof stream. Our canonical $\mathcal{AM}$ algorithm works for the weaker model, wherein the proof is being streamed, thus our $\mathcal{AM}$ upper bounds also implies $\widehat{\mathcal{AM}}$ upper bounds.

4 Techniques

The main intuition behind our canonical $\mathcal{AM}$ streaming algorithm is based on the "algebrization" inspired communication complexity protocol of Aaronson and Wigderson [1]. However our proof is much more technically involved.

In general, say we have a data stream problem $\mathcal{P}$ and two integers s, w such that $s \cdot w \geq n$. If there exists a low degree polynomial $g(x, y) : \mathbb{Z}^2 \to \mathbb{Z}$ (that depends on the input stream σ) and two domains $\mathcal{D}_w, \mathcal{D}_s \subseteq \mathbb{Z}$ of cardinality w, s (respectively) such that

$$\mathcal{P}(\sigma) = \sum_{x \in \mathcal{D}_w} \sum_{y \in \mathcal{D}_s} g(x, y),$$

then assuming we can efficiently evaluate g at a random point, by a straightforward adaptation of the [1] protocol to the settings of streaming algorithms, we obtain a simple $\mathcal{MA}$ streaming algorithm for $\mathcal{P}$.

However, in our case we can only express $\mathcal{P}(\sigma)$ as

$$\sum_{x \in \mathcal{D}_w} \sum_{y \in \mathcal{D}_s} \psi\big(C_1 \circ \tilde{\chi}(x, y), \ldots, C_k \circ \tilde{\chi}(x, y)\big),$$

where k is a natural number, $\{C_t\}_{t \in [k]}$ are clauses over m variables, $\psi : \{0,1\}^k \to \mathbb{Z}$ is a function over the hypercube, $\tilde{\chi} : \mathcal{D}_w \times \mathcal{D}_s \to \{0,1\}^m$ is the bivariate equivalent of the element indicator $\chi : [n] \to \{0,1\}^m$, and $\mathcal{D}_w, \mathcal{D}_s \subseteq \mathbb{Z}$ are domains of cardinality w, s (respectively).

The function $\psi\big(C_1 \circ \tilde{\chi}(x, y), \ldots, C_k \circ \tilde{\chi}(x, y)\big)$ is not a low degree polynomial. We would have liked to overcome this difficulty by using the approximation method of [18,20]. The latter allows us to have a low degree approximation of the clauses $\{C_t\}_{t \in [k]}$, such that with high probability (over the construction of the approximation polynomials) we can replace the clauses with low degree polynomials, without changing the output. The aforementioned randomized procedure comes at a cost of turning the $\mathcal{MA}$ streaming algorithm to an $\mathcal{AM}$ streaming algorithm.

Yet, the above does not sufficiently reduces the degree of

$$\psi\big(C_1 \circ \tilde{\chi}(x, y), \ldots, C_k \circ \tilde{\chi}(x, y)\big).$$

This is due to the fact that the method of [18,20] results with approximation polynomials over a finite field of cardinality that is larger than $\mathcal{P}(\sigma)$. The degree of the approximation polynomials is close to the cardinality of the finite field, which in our case can be a large number ($\mathrm{poly}(m, n)$).

Instead we aim to apply the method of [18,20] to approximate

$$\{\mathcal{P}(\sigma) \pmod q\}_{q\in Q}$$

for a set Q of $\mathrm{polylog}(m,n)$ primes, each of size at most $\mathrm{polylog}(m,n)$. This way, each approximation polynomial that we get is over a finite field of cardinality $\mathrm{polylog}(m,n)$, and of sufficiently low degree. Then, we use the *Chinese Remainder Theorem* to extract the value of $\mathcal{P}(\sigma)$ from $\{\mathcal{P}(\sigma) \pmod q\}_{q\in Q}$.

Nonetheless, this is still not enough, as for every $q \in Q$ we want the answer to be the summation of the polynomial approximation of $\psi\big(C_1 \circ \tilde{\chi}(x,y), \ldots, C_k \circ \tilde{\chi}(x,y)\big) \pmod q$ over some domain $\mathcal{D}_w \times \mathcal{D}_s \subseteq \mathbb{Z}^2$ (where $|\mathcal{D}_w| = w$ and $|\mathcal{D}_s| = s$). Since the cardinality of the field $\mathbb{F}_q$ is typically smaller than w and s, we use an extension (of sufficient cardinality) of the field $\mathbb{F}_q$.

At each step of the construction, we make sure that we perserve both the restrictions that are imposed by the data stream model, and the conditions that are needed to ensure an efficient verification of the proof.

The idea behind our $\mathcal{AM}$ streaming algorithm for *Distinct Elements* is simply noting that we can indicate whether an element j appears in the data stream, by the disjunction of the element indicators of $j \in [n]$ in all of the positions of the stream (i.e., $\chi_1(j), \ldots, \chi_m(j)$). Then we can represent the number of distinct elements as a sum of disjunctions, and use the canonical $\mathcal{AM}$ streaming algorithm in order to solve the *Distinct Elements* problem.

As for the lower bound on the $\mathcal{MA}$ streaming complexity of the *Distinct Elements* problem, we start by establishing a lower bound on the $\mathcal{MA}$ communication complexity of the *Gap Hamming Distance* problem (GHD). A key element in the proof of the latter is based on Sherstov's recent result [19] on the *Gap Orthogonality* problem (ORT) and its relation to GHD. Sherstov observed that the problem of *Gap Orthogonality* readily reduces to *Gap Hamming Distance* problem. Although at first glance it seems that the transition to ORT is of little substance, it turns out that Yao's corruption bound [22] suits it perfectly. In fact, the corruption property for ORT is equivalent to the anti-concentration property of orthogonal vectors in the Boolean cube. Using this observation, we prove a lower bound on the $\mathcal{MA}$ communication complexity of ORT (following the method of [17]), which in turn, by the reduction from ORT to GHD, implies a lower bound on the $\mathcal{MA}$ communication complexity of GHD. Next we adapt the reduction that was implicitly stated in [14], and reduce the $\mathcal{MA}$ communication complexity problem of GHD to the $\mathcal{MA}$ problem of calculating the exact number of *Distinct Elements*.

For the full formal proofs, we refer the reader to the full version [13].

References

1. Aaronson, S., Wigderson, A.: Algebrization: A new barrier in complexity theory. ACM Trans. Comput. Theory 1, 2:1–2:54 (2009)
2. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC 1996, pp. 20–29. ACM, New York (1996)

3. Beyer, K.S., Haas, P.J., Reinwald, B., Sismanis, Y., Gemulla, R.: On synopses for distinct-value estimation under multiset operations. In: SIGMOD Conference, pp. 199–210 (2007)
4. Brody, J., Chakrabarti, A.: A multi-round communication lower bound for gap hamming and some consequences. In: IEEE Conference on Computational Complexity, pp. 358–368 (2009)
5. Chakrabarti, A., Cormode, G., Mcgregor, A.: Annotations in data streams. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 222–234. Springer, Heidelberg (2009)
6. Chakrabarti, A., Regev, O.: An optimal lower bound on the communication complexity of gap-hamming-distance. In: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing, STOC 2011, pp. 51–60. ACM, New York (2011)
7. Chung, K.-M., Kalai, Y.T., Liu, F.-H., Raz, R.: Memory delegation. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 151–168. Springer, Heidelberg (2011)
8. Cormode, G., Mitzenmacher, M., Thaler, J.: Streaming graph computations with a helpful advisor. CoRR, abs/1004.2899 (2010)
9. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. CoRR (May 2011)
10. Flajolet, P., Martin, G.N.: Probabilistic counting. In: FOCS, pp. 76–82 (1983)
11. Ganguly, S., Kanpur, I., Garofalakis, M.: Join-distinct aggregate estimation over update streams. In: Proc. ACM PODS (2005)
12. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: STOC, pp. 113–122 (2008)
13. Gur, T., Raz, R.: Arthur-merlin streaming complexity. Electronic Colloquium on Computational Complexity (ECCC) 20, 20 (2013)
14. Indyk, P., Woodruff, D.P.: Tight lower bounds for the distinct elements problem. In: FOCS, pp. 283–289 (2003)
15. Kane, D.M., Nelson, J., Woodruff, D.P.: An optimal algorithm for the distinct elements problem. In: PODS, pp. 41–52 (2010)
16. Muthukrishnan, S.: Data streams: algorithms and applications. Now Publishers (2005)
17. Raz, R., Shpilka, A.: On the power of quantum proofs. In: Annual IEEE Conference on Computational Complexity, pp. 260–274 (2004)
18. Razborov, A.: Lower bounds for the size of circuits of bounded depth with basis $\{\wedge, \oplus\}$. Notes of the Academy of Science of the USSR 41(4), 333–338 (1987)
19. Sherstov, A.A.: The communication complexity of gap hamming distance. Electronic Colloquium on Computational Complexity (ECCC) 18, 63 (2011)
20. Smolensky, R.: Algebraic methods in the theory of lower bounds for boolean circuit complexity. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC 1987, pp. 77–82. ACM, New York (1987)
21. Vidick, T.: A concentration inequality for the overlap of a vector on a large set, with application to the communication complexity of the gap-hamming-distance problem. Electronic Colloquium on Computational Complexity (ECCC) 18, 51 (2011)
22. Yao, A.C.: Lower bounds by probabilistic arguments. In: Proceedings of the 24th Annual Symposium on Foundations of Computer Science, pp. 420–428. IEEE Computer Society, Washington, DC (1983)

Local Correctability of Expander Codes

Brett Hemenway[1], Rafail Ostrovsky[2,⋆], and Mary Wootters[1,⋆⋆]

[1] University of Michigan
{bhemen,wootters}@umich.edu
[2] Department of Computer Science and Department of Mathematics, UCLA
rafail@cs.ucla.edu

Abstract. In this work, we present the first local-decoding algorithm for expander codes. This yields a new family of constant-rate codes that can recover from a constant fraction of errors in the codeword symbols, and where any symbol of the codeword can be recovered with high probability by reading N^{ε} symbols from the corrupted codeword, where N is the block-length of the code.

Expander codes, introduced by Sipser and Spielman, are formed from an expander graph $G = (V, E)$ of degree d, and an inner code of block-length d over an alphabet Σ. Each edge of the expander graph is associated with a symbol in Σ. A string in Σ^E will be a codeword if for each vertex in V, the symbols on the adjacent edges form a codeword in the inner code.

We show that if the inner code has a smooth reconstruction algorithm in the noiseless setting, then the corresponding expander code has an efficient local-correction algorithm in the noisy setting. Instantiating our construction with inner codes based on finite geometries, we obtain a novel locally decodable codes with constant rate. This provides an alternative to the multiplicity codes of Kopparty, Saraf and Yekhanin (STOC '11) and the lifted codes of Guo, Kopparty and Sudan (ITCS '13).

Keywords: error correcting codes, expander codes, locally decodable codes.

1 Introduction

Expander codes, introduced in [29], are linear codes which are notable for their efficient decoding algorithms. In this paper, we show that when appropriately

⋆ 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.

⋆⋆ Research supported in part by NSF grant CCF-1161233.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 540–551, 2013.

instantiated, expander codes are also *locally decodable,* and we give a sublinear time local-decoding algorithm.

In standard error correction, a sender encodes a message $x \in \{0,1\}^k$ as a codeword $c \in \{0,1\}^N$, and transmits it to a receiver across a noisy channel. The receiver's goal is to recover x from the corrupted codeword w. Decoding algorithms typically process all of w and in turn recover all of x. The goal of local decoding is to recover only few bits of x, with the benefit of querying only a few bits of w. The number of bits, q, of w needed to recover a single bit x is known as the *query complexity,* and the important trade-off in local decoding is between this quantity and the rate $r = k/N$ of the code. When q is constant or even logarithmic in k, the best known codes have rates which tend to zero as N grows. Until recently, there were no known locally decodable codes with rate approaching 1 and sublinear locality; to date, there are only two constructions known [18,22]. In this work, we show that expander codes provide a new construction of efficiently locally decodable codes with constant rate.

1.1 Notation and Preliminaries

We will construct linear codes $\mathcal{C}$ of length N and message length k, over an alphabet $\Sigma = \mathbb{F}$, for some finite field $\mathbb{F}$. That is, $\mathcal{C} \subset \mathbb{F}^N$ is a linear subspace of dimension k. The *rate* of $\mathcal{C}$ is the ratio $r = k/N$. We say a d-regular graph G is a *spectral expander* with parameter λ, if λ is the second-largest eigenvalue of the normalized adjacency matrix of G. Intuitively, the smaller λ is, the better connected G is—see [19] for a survey of expanders and their applications. For $n \in \mathbb{Z}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. For an event $\mathcal{E}$, $\mathbf{1}_{\mathcal{E}}$ is the indicator variable which is 1 if $\mathcal{E}$ occurs and 0 otherwise. For $x, y \in \Sigma^N$, $\Delta(x, y)$ denotes relative Hamming distance, $x[i]$ denotes the i^{th} symbol of x, and $x|_S$ denotes x restricted to symbols indexed by $S \subset [N]$.

A code is *locally decodable* if there is a decoding algorithm that recovers any symbol, $x[i]$, of the message, making only a few queries to received word.

Definition 1 (Locally Decodable Codes (LDCs)). *Let $\mathcal{C} \subset \Sigma^N$ be a code of size $|\Sigma|^k$, and let $E : \Sigma^k \to \Sigma^N$ be an encoding map. Then $(\mathcal{C}, E)$ is (q, ρ)-locally decodable with error probability η if there is a randomized algorithm R, so that for any $w \in \Sigma^N$ with $\Delta(w, E(x)) < \rho$, for each $i \in [k]$,*

$$\mathbb{P}\{R(w, i) = x[i]\} \geq 1 - \eta,$$

and further R accesses at most q symbols of w.

We actually construct a stronger primitive, *locally correctable codes.*

Definition 2 (Locally Correctable Codes (LCCs)). *Let $\mathcal{C} \subset \Sigma^N$ be a code. Then $\mathcal{C}$ is (q, ρ)-locally correctable with error probability η if there is a randomized algorithm, R, so that for any $w \in \mathbb{F}^N$ with $\Delta(w, E(x)) < \rho$, for each $j \in [N]$,*

$$\mathbb{P}\{R(w, j) = w[j]\} \geq 1 - \eta,$$

and further R accesses at most q symbols of w.

When there is a constant $\rho > 0$ and a failure probability $\eta = o(1)$ so that $\mathcal{C}$ is (q, ρ)-locally correctable with error probability η, we will simply say that $\mathcal{C}$ is locally correctable with query complexity q (and similarly for locally decodable).

When $\mathcal{C}$ is a linear code, writing the generator matrix in systematic form gives an encoding function $E : \mathbb{F}^k \to \mathbb{F}^N$ so that for every $x \in \mathbb{F}^k$ and for all $i \in [k]$, $E(x)[i] = x[i]$. If $\mathcal{C}$ is a (q, ρ) linear LCC, then $(E, \mathcal{C})$ is a (q, ρ) LDC. Thus we will focus our attention on creating locally correctable linear codes.

Many LCCs work on the following principle: suppose, for each $i \in [N]$, there is a set of q query positions $Q(i)$, which are *smooth*—that is, each query is almost uniformly distributed within the codeword—and a method to determine $c[i]$ from $\{c[j] : j \in Q(i)\}$ for any uncorrupted codeword $c \in \mathcal{C}$. If q is constant, this *smooth local reconstruction algorithm* yields a local correction algorithm: with high probability none of the locations queried are corrupted. When q is merely sublinear in N, as is the case in this work, this reasoning fails. This work demonstrates how to turn codes which only possess a local reconstruction procedure (in the noiseless setting) into LCCs with constant rate and sublinear query complexity.

Definition 3 (Smooth reconstruction). *For a code $\mathcal{C} \subset \Sigma^N$, consider a pair of algorithms (Q, A), where Q is a randomized query algorithm with inputs in $[N]$ and outputs in 2^N, and $A : \Sigma^q \times [N] \to \Sigma$ is a deterministic reconstruction algorithm. We say that (Q, A) is a* s-smooth local reconstruction algorithm *with query complexity q if the following hold.*

1. *For each $i \in [N]$, the query set $Q(i)$ has $|Q(i)| \leq q$.*
2. *For each $i \in [N]$, there is some set $S \subset [N]$ of size s, so that each query in $Q(i)$ is uniformly distributed in S.*
3. *For all $i \in [N]$ and for all codewords $c \in \mathcal{C}$, $A(c|_{Q(i)}, i) = c[i]$.*

If $s = N$, then we say the reconstruction is perfectly smooth, since all symbols are equally likely to be queried (though the queries need not be independent). The inner codes we consider decode a symbol indexed by $x \in \mathbb{F}^m$ by querying random subspaces through x (but not x itself), and thus will have $s = N - 1$.

1.2 Related Work

The first local-decoding procedure for an error-correcting code was the majority-logic decoder for Reed-Muller codes proposed by Reed [28]. Local-decoding procedures have found many applications in theoretical computer science including proof-checking [3, 23, 27] and self-testing [9, 10, 16, 17]. While these applications implicitly used local-decoding procedures, the first explicit definition of locally decodable codes did not appear until later [21]. An excellent survey is available [33]. The study of locally decodable codes focuses on the trade-off between rate (the ratio of message length to codeword length) and query complexity (the number of queries made by the decoder). Research in this on locally decodable codes is separated into two distinct areas: the first seeks to minimize

the query complexity, while the second seeks to maximize the rate. In the low-query-complexity regime, Yekhanin was the first to exhibit codes with a constant number of queries and a subexponential rate [32]. Following Yekhanin's work, there has been significant progress in constructing locally decodable codes with constant query-complexity [7, 8, 11–14, 20, 32]. On the other hand, in the high-rate regime, there has been less progress. In 2011, Kopparty, Saraf and Yekhanin introduced *multiplicity codes*, the first codes with a sublinear local-decoding algorithm [22] and rate greater than one half. Like Reed-Muller codes, multiplicity codes treat the message as a multivariate polynomial, and create codewords by evaluating the polynomial at a sequence of points. Multiplicity codes improve on the performance of Reed-Muller codes by including evaluations of the partial derivatives of the message polynomial in the codeword. A separate line of work has developed high-rate locally decodable codes by "lifting" shorter codes [18]. The work of Guo, Kopparty and Sudan takes a short code $\mathcal{C}_0$ of length q^t, and lifts it to a longer code $\mathcal{C}$, of length $N > q^t$ over $\mathbb{F}_q$, such that every restriction of a codeword in $\mathcal{C}$ to an affine subspace of dimension t yields a codeword in $\mathcal{C}_0$. The definition provides a natural local-correcting procedure for the outer code: to decode a symbol of the outer code, pick a random affine subspace of dimension t that contains the symbol, read the q^t coordinates and decode the resulting codeword using the code $\mathcal{C}_0$. By lifting explicit inner codes the outer code can achieve constant rate and query complexity N^ε.

In this work, we show that *expander codes* can also give constant-rate locally decodable codes with query complexity N^ε. Expander codes, introduced by Sipser and Spielman [29], are formed by choosing a d-regular expander graph, G, on n vertices, and a code $\mathcal{C}_0$ of length d (called the *inner code*), and defining the codeword to be all assignments of symbols to the edges of G so that for every vertex in G, its edges form a codeword in $\mathcal{C}_0$. The connection between error-correcting codes and graphs was first noticed by Gallager [15] who showed that a random bipartite graph induces a good error-correcting code. Gallager's construction was refined by Tanner [31], who suggested the use of an inner code. Sipser and Spielman [29] were the first to consider this type of code with an expander graph, and Spielman [30] showed that these *expander codes* could be encoded and decoded in linear time. Spielman's work provided the first family of error-correcting code with linear-time encoding and decoding procedures. The decoding procedure has since been improved by Barg and Zemor [4–6, 34].

1.3 Our Approach and Contributions

We show that certain expander codes can be efficiently locally decoded, and we instantiate our results to obtain novel families of (N^ε, ρ)-LCCs of rate $1 - \alpha$ for arbitrary $\alpha, \varepsilon > 0$ and some positive constant ρ. Our decoding algorithm runs in time linear in the number of queries, and hence sublinear in the length of the message. We provide a general method for turning codes with smooth local reconstruction algorithms into LCCs: our main result, Theorem 2, states that as long as the inner code $\mathcal{C}_0$ has rate at least $1/2$ and possesses a smooth local reconstruction algorithm, then the corresponding family of expander codes are

constant rate LCCs. In Section 3, we give examples of inner codes, leading to the parameters claimed above.

Our approach (and the resulting codes) are very different from earlier approaches. Both multiplicity codes and lifted Reed-Solomon codes use the same basic principle, also at work in Reed-Muller codes: in these schemes, for any two codewords c_1 and c_2 which differ at index i, the corresponding queries $c_1|_{Q(i)}$ and $c_2|_{Q(i)}$ differ in many places. Thus, if the queries are smooth, with high probability they will not have too many errors, and the correct symbol can be recovered. In contrast, our decoder works differently: while our queries are smooth, they will not have this distance property. In fact, we will see in the proof of Theorem 2 that changing a mere $\log(q)$ out of our q queries may change the correct answer. The trick is that these problematic error patterns must have a lot of structure, and we will show that they are unlikely to occur.

Finally, our results port a typical argument from the low-query regime to the high-rate regime. As mentioned above, when the query complexity, q, is constant, a smooth local reconstruction algorithm is sufficient for local correctability, but this reasoning fails when q grows with N. Our work overcomes this problem: Theorem 2 shows that any family of codes, $\mathcal{C}_0$, with good rate and smooth local reconstruction can be used to obtain a family of LCCs with similar parameters.

2 Local Correctability of Expander Codes

In this section, we give an efficient local correction algorithm for expander codes with appropriate inner codes. We use a formulation of expander codes due to [34]. Let G be a d-regular expander graph on n vertices with parameter λ. We will take G to be a *Ramanujan graph,* so that $\lambda \leq \frac{2\sqrt{d-1}}{d}$; explicit constructions of Ramanujan graphs are known [24–26] for arbitrarily large values of d. Let H be the double cover of G. That is, H is a bipartite graph whose vertices $V(H)$ are two disjoint copies V_0 and V_1 of $V(G)$, and so that

$$E(H) = \{(u_0, v_1) \,:\, (u, v) \in E(G)\},$$

where u_i denotes the copy of u in V_i. Fix a linear inner code $\mathcal{C}_0$ over Σ of rate r_0 and relative distance δ_0. Let $N = nd$. For $v_i \in V(H)$, let $E(v_i)$ denote the edges attached to v. The expander code $\mathcal{C} \subset \Sigma^N$ arising from G and $\mathcal{C}_0$ is given by

$$\mathcal{C} = \mathcal{C}_N(\mathcal{C}_0, G) = \left\{x \in \Sigma^N \,:\, x|_{E(v_i)} \in \mathcal{C}_0 \text{ for all } v_i \in V(H)\right\} \tag{1}$$

The following theorem shows that as long as the inner code $\mathcal{C}_0$ has good rate and distance, so does the resulting code $\mathcal{C}$.

Theorem 1 ([29,31]). *The code $\mathcal{C}$ has rate $r = 2r_0 - 1$, and as long as $2\lambda \leq \delta_0$, the relative distance of $\mathcal{C}$ is at least $\delta_0^2/2$.*

2.1 Local Correction

If the inner code $\mathcal{C}_0$ has a smooth local reconstruction procedure, then not only does $\mathcal{C}$ have good distance, but in fact it is also efficiently locally decodable. Our main result is the following theorem.

Theorem 2. *Let $\mathcal{C}_0$ be a linear code over Σ of length d and rate $r_0 > 1/2$. Suppose that $\mathcal{C}_0$ has a s_0-smooth local reconstruction procedure with query complexity q_0. Let $\mathcal{C} = \mathcal{C}_N(\mathcal{C}_0, G)$ be the expander code of length N arising from the inner code $\mathcal{C}_0$ and a Ramanujan graph G. Choose any $\gamma < 1/2$ and any $\zeta > 0$ satisfying $\gamma \left(e^{\zeta} q_0\right)^{-1/\gamma} > 8\lambda$. Then $\mathcal{C}$ is (q, ρ)-locally correctable, for any error rate ρ, with $\rho < \gamma \left(e^{\zeta} q_0\right)^{-1/\gamma} - 2\lambda$. The success probability is*

$$1 - \left(\frac{N}{d}\right)^{-1/\ln(d/4)}$$

and the query complexity is

$$q = \left(\frac{N}{d}\right)^{\varepsilon} \qquad \text{where} \qquad \varepsilon = \left(1 + \frac{\ln(q_0') + 1}{\zeta}\right) \cdot \frac{\ln(q_0')}{\ln(d/4)}.$$

Further, when the length of the inner code, d, is constant, the correction algorithm runs in time $O(|\Sigma|^{q_0'+1} q)$, where $q_0' = q_0 + (d - s_0)$.

Remark 1. We will choose d (and hence $q_0' < d$) and $|\Sigma|$ to be constant. Thus, the rate of $\mathcal{C}$, as well as the parameters ρ and ε, will be constants independent of the block length N. The parameter ζ trades off between the query complexity and the allowable error rate. When q_0 is much smaller than d (for example, $q_0 = 3$ and d is reasonably large), we will take $\zeta = O(1)$. On the other hand, if $q_0 = d^{\varepsilon}$ and d is chosen to be a sufficiently large constant, we should take $\zeta \approx \ln(q_0)$.

Here, we give the correction algorithm and sketch our argument, the detailed proof can be found in the full version of this paper. First, we observe that it suffices to consider the case when Q_0 is perfectly smooth: that is, the queries of the inner code are uniformly random. Otherwise, if Q_0 is s_0-smooth with q_0 queries, we may modify it so that it is d-smooth with $q_0 + (d - s_0)$ queries, by having it query extra points and then ignore them. Thus, we assume in the following that Q_0 makes q_0 perfectly smooth queries.

Now, suppose that $\mathcal{C}_0$ has smooth local reconstruction algorithm (Q_0, A_0), and we receive a corrupted codeword, w, which differs from a correct codeword c^* in at most a ρ fraction of the entries. Say we wish to determine $c^*[(u_0, v_1)]$, for $(u_0, v_1) \in E(H)$. The algorithm proceeds in two steps. First, we find a set of about $N^{\varepsilon/2}$ query positions which are nearly uniform in $[N]$, and whose correct values together determine $c^*[(u_0, v_1)]$. Next, we correct each of *these* queries with very high probability—for each, we will make another $N^{\varepsilon/2}$ or so queries.

Step 1. By construction, $c^*[(u_0, v_1)]$ is a symbol in a codeword of the inner code, $\mathcal{C}_0$, which lies on the edges emanating from u_0. By applying Q_0, we may choose q_0 of these edges, $S = \left\{(u_0, s_1^{(i)}) : i \in [q_0]\right\}$, so that $A_0\left(c^*|_S, (u_0, v_1)\right) = c[(u_0, v_1)]$. Now we repeat on each of these edges: each $(u_0, s_1^{(i)})$ is part of a codeword emanating from $s_1^{(i)}$, and so q_0 more queries determine each of those, and so on. Repeating this L_1 times yields a q_0-ary tree T of depth L_1, whose nodes are labeled by of edges of H. This tree-making procedure is given more precisely below in Algorithm 2. Because the queries are smooth, each path down this tree is a random walk in H; because G is an expander, this means that the leaves themselves, while not independent, are each close to uniform on $E(H)$. Note that at this point, we have not made any queries, merely documented a tree, T, of edges we could query.

Step 2. Our next step is to actually make queries to determine the correct values on the edges represented in the leaves of T. By construction, these values determine $c^*[(u_0, v_1)]$. Unfortunately, in expectation a ρ fraction of the leaves are corrupted, and we need them all to be correct. We will use the fact that each leaf is nearly uniform in $E(H)$ to argue that we can correct it with very high probability—large enough to tolerate a union bound over all of the leaves. For each edge, e, of H that shows up on a leaf of T, we repeat the tree-making process beginning at this edge, resulting in new q_0-ary trees T_e of depth L_2. This time, we make *all* the queries along the way, resulting in an evaluated tree τ_e, whose nodes are labeled by elements of Σ; the root of τ_e is the e-th position in the corrupted codeword, $w[e]$, and we hope to correct it to $c^*[e]$, which we do in Algorithm 3. The intuition is as follows: suppose that we (the decoder) are fooled by some errors in τ into making an incorrect guess at the root. Then these errors must have a very particular structure: if the root is incorrect, then one of its children must be, and so on. Thus, the errors disproportionately affect some path from the root to a leaf of τ. To be slightly more precise, for two labelings σ and ν of the same tree by elements of Σ, consider the distance

$$D(\sigma, \nu) = \max_P \Delta\left(\sigma|_P, \nu|_P\right), \tag{2}$$

where the maximum is over all paths P from the root to a leaf, and $\sigma|_P$ denotes the restriction of σ to P. Because G is an expander, it is very unlikely that τ contains a path from the root to a leaf with more than a constant fraction $\eta < 1/2$ of errors. In the favorable case, the distance between the correct tree τ^* arising from c^* and the observed tree τ is at most $D(\tau^*, \tau) \leq \eta$. In contrast, we also show that if σ^* and τ^* are both trees arising from legitimate codewords with distinct roots, then σ^* and τ^* must differ on an entire path P, and so $D(\sigma^*, \tau) > 1 - \eta$. To take advantage of this, we must efficiently decide which symbol $a \in \Sigma$ has a tree σ^* with root labelled a, so that $D(\sigma^*, \tau) < \eta$, and then $c^*[e] = a$ will be our answer. Algorithm 3 does precisely this, with one pass over τ. Loosely, we assume recursively that each node y at level $\ell - 1$ knows how far the subtree below it is from a tree σ^* with root a—this is captured in the

quantity $\mathsf{best}_a(y)$. At level ℓ, a node x determines $\mathsf{best}_a(x)$ by assigning labels to its children so that A_0 returns a, and using the values from the previous level.

Once all the leaves of T are correctly evaluated, we may use A_0 to work our way back up T and determine the correct symbol corresponding to the edge at the root of T. The complete correction algorithm is given below in Algorithm 1.

Algorithm 1. correct: Local correcting protocol.

Input: An index $e_0 \in E(H)$, and a corrupted codeword $w \in \Sigma^{E(H)}$.
Output: With high probability, the correct value of the e_0'th symbol.
Set $L_1 = \log(q_0)/\log(d/4)$ and a parameter L_2
$T = \mathsf{makeTree}(e_0, L_1)$
for *each edge e of H that showed up on a leaf of T* **do**
  $T_e = \mathsf{makeTree}(e, L_2)$
  Let $\tau_e = T_e|_w$ be the tree of symbols from w
  $w^*[e] = \mathsf{correctSubtree}(\tau_e)$
Initialize a q_0-ary tree τ^* of depth L_1
Label the leaves of τ^* according to T and w^*: if a leaf of T is labeled e, label the corresponding leaf of τ^* with $w^*[e]$.
Use the local reconstruction algorithm A_0 of $\mathcal{C}_0$ to label all the nodes in τ^*
return The label on the root of τ^*

Algorithm 2. makeTree: Uses the local correction property of $\mathcal{C}_0$ to construct a tree of indices.

Input: An initial edge $e_0 = (u_0, v_1) \in E(H)$, and a depth L.
Output: A q_0-ary tree T of depth L, whose nodes are indexed by edges of H, with root e_0
Initialize a tree T with a single node labeled e_0
$s = 0$
for $\ell \in [L]$ **do**
  Let leaves be the current leaves of T
  for $e = (u_s, v_{1-s}) \in$ *leaves* **do**
    Let $\left\{v_{1-s}^{(i)} : i \in [d]\right\}$ be the neighbors of u_s in H
    Choose queries $Q_0(e) \subset \left\{(u_s, v_{1-s}^{(i)}) : i \in [d]\right\}$, and add each query in T as a child at e.
  $s = 1 - s$
return T

The number of queries made by Algorithm 1 is $q = q_0^{L_1+L_2}$ and the running time is $O(t_d|\Sigma|^{q_0+1}q)$, where t_d is the time required to run the local correction algorithm of $\mathcal{C}_0$. When d and $|\Sigma|$ are constant, and the running time is $O(q)$.

Algorithm 3. correctSubtree: Correct the root of a fully evaluated tree τ.

Input: τ, a q_0-ary tree of depth L whose nodes are labeled with elements of Σ.
Output: A guess at the root of the correct tree τ.
For a node x of τ, let $\tau[x]$ denote the label on x.
for *leaves x of τ and $a \in \Sigma$* **do**
  $\mathsf{best}_a(x) = \begin{cases} 1 & \tau[x] \neq a \\ 0 & \tau[x] = a \end{cases}$
for $\ell = L-1, L-2, \ldots, 0$ **do**
  for *nodes x at level ℓ in τ and $a \in \Sigma$* **do**
    Let $y_1, \ldots, y_{q_0}$ be the children of x
    Let $S_a \subset \Sigma^{q_0}$ be the set of query responses for the children of x so that A_0 returns a on those responses
    $\mathsf{best}_a(x) = \min_{(a_0,\ldots,a_{q_0}) \in S_a} \max_{r \in [q_0]} \left(\mathsf{best}_{a_r}(y_r) + \mathbf{1}_{\tau(y_r) \neq a_r} \right)$
Let r be the root of τ
for $a \in \Sigma$ **do**
  $$\mathsf{Score}(a) = \frac{\mathsf{best}_a(r) + \mathbf{1}_{\tau(r) \neq a}}{L}$$
return *$a \in \Sigma$ with the smallest $\mathsf{Score}(a)$*

3 Examples

In this section, we provide two examples of choices for $\mathcal{C}_0$, both of which result in (N^ε, ρ)-LCCs of rate $1 - \alpha$ for any sufficiently small constants $\varepsilon, \alpha > 0$ and for some constant $\rho > 0$. Our main example is a generalization of Reed-Muller codes, based on affine geometries. With these codes as $\mathcal{C}_0$, we provide LCCs over $\mathbb{F}_p$—unlike multiplicity codes, these codes work naturally over small fields.

Our second example comes from the observation that if the $\mathcal{C}_0$ is itself an LCC (of a fixed length) our construction provides a new family of (N^ε, ρ)-LCCs. In particular, plugging the multiplicity codes of [22] into our construction yields a novel family of LCCs. This new family of LCCs has a very different structure than the underlying multiplicity codes, but achieves roughly the same rate and locality. Due to space constraints, the reader is referred to the full paper for the full details of both examples.

Codes from Affine Geometries. One advantage of our construction is that the inner code $\mathcal{C}_0$ need not actually be a good locally decodable or correctable code. Rather, we only need a smooth reconstruction procedure, which is easier to come by. One example comes from affine geometries.

For a prime power $h = p^\ell$ and parameters r and m, consider the r-dimensional affine subspaces $L_1, \ldots, L_t$ of the vector space $\mathbb{F}_h^m$. let H be the $t \times h^m$ incidence matrix of the L_i and the points of $\mathbb{F}_h^m$, and let $\mathcal{A}^*(r, m, h)$ be the code over $\mathbb{F}_p$ whose parity check matrix is H. These *finite geometry codes* are well-studied, and their ranks can be exactly computed—see [1, 2] for an overview.

The definition of $\mathcal{A}^*(r, m, h)$ gives a reconstruction procedure: we may query all the points in a random r-dimensional affine subspace, and use the corresponding parity check. The locality of $\mathcal{A}^*(r, m, h)$ has been noticed before, for example in [18], where it was observed that these codes could be viewed as lifted parity check codes. However, as they note, these codes do not themselves make good LCCs—the reconstruction procedure cannot tolerate any errors in the chosen subspace, and thus the error rate ρ must tend to zero as the block length grows. Even though these codes are not good LCCs, we can use them in Theorem 2 to obtain good LCCs with sublinear query complexity, which can correct a constant fraction of errors.

For any sufficiently small constants ε, α, and any prime p, we show in the full version of this paper how to pick r, m, and h, so that Theorem 2 applies, and so that the resulting expander code $\mathcal{C}$ of block-length N is a linear p-ary (N^ε, ρ) LCC with rate $1 - \alpha$ and query complexity N^ε, where ρ is a constant independent of N.

Multiplicity codes. Multiplicity codes [22] are themselves a family of constant-rate locally decodable codes. We can, however, use a multiplicity code of constant length as the inner code $\mathcal{C}_0$ in our construction. This results in a new family of constant-rate locally decodable codes. The parameters we obtain from this construction are slightly worse than the original multiplicity codes, and the main reason we include this example is novelty—these new codes have a very different structure than the original multiplicity codes. We show in the full version that for an arbitrarily small constant β, the multiplicity codes from [22] with block-length d, locality on the order of $d^{(1-\beta)\varepsilon}$, and rate $1 - \alpha$ result via Theorem 2 in a family of expander codes $\mathcal{C}_N$ of block-length N, with locality $O(N^\varepsilon)$ and rate $1 - 2\alpha$.

4 Conclusion

In the constant-rate regime, all known LDCs work by using a smooth local reconstruction algorithm. When the locality is, say, three, then with very high probability none of the queried positions will be corrupted. This reasoning fails for constant rate codes, which have larger query complexity: we expect a ρ fraction of errors in our queries, and this is often difficult to deal with. In this work, we have shown how to make the low-query argument valid in a high-rate setting—any code with large enough rate and with a good local reconstruction algorithm can be used to make a full-blown locally correctable code.

This work presented the first sublinear time algorithm for locally correcting expander codes. More precisely, we have shown that as long as the inner code $\mathcal{C}_0$ admits a smooth local reconstruction algorithm with appropriate parameters, then the resulting expander code $\mathcal{C}$ is a (N^ε, ρ)-LCC with rate $1 - \alpha$, for any $\alpha, \varepsilon > 0$ and some constant ρ. Further, we presented a correction algorithm with runtime linear in the number of queries. Expander codes are a natural construction, and it is our hope that the additional structure of our codes, as well as the extremely fast decoding time, will lead to new applications of local decodability.

References

1. Assmus, E.F., Key, J.D.: Designs and their Codes, vol. 103. Cambridge University Press (1994)
2. Assmus, E.F., Key, J.D.: Polynomial codes and finite geometries. In: Handbook of Coding Theory, vol. 2(part 2), pp. 1269–1343 (1998)
3. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, STOC 1991, pp. 21–32. ACM, New York (1991)
4. Barg, A., Zemor, G.: Error exponents of expander codes. IEEE Transactions on Information Theory 48(6), 1725–1729 (2002)
5. Barg, A., Zemor, G.: Concatenated codes: serial and parallel. IEEE Trans. Inf. Theor. 51(5), 1625–1634 (2005)
6. Barg, A., Zemor, G.: Distance properties of expander codes. IEEE Transactions on Information Theory 52(1), 78–90 (2006)
7. Beimel, A., Ishai, Y., Kushilevitz, E., Orlov, I.: Share Conversion and Private Information Retrieval. In: CCC 2012, pp. 258–268. IEEE Computer Society, Los Alamitos (2012)
8. Ben-Aroya, A., Efremenko, K., Ta-Shma, A.: Local List Decoding with a Constant Number of Queries. In: 2010 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 715–722. IEEE (October 2010)
9. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, STOC 1990, pp. 73–83. ACM, New York (1990)
10. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. Journal of Computer and System Sciences 47(3), 549–595 (1993)
11. Chee, Y.M., Feng, T., Ling, S., Wang, H., Zhang, L.F.: Query-Efficient Locally Decodable Codes of Subexponential Length. In: Computational Complexity, pp. 1–31 (August 2011)
12. Dvir, Z., Gopalan, P., Yekhanin, S.: Matching Vector Codes. SIAM Journal on Computing 40(4), 1154–1178 (2011)
13. Efremenko, K.: 3-query locally decodable codes of subexponential length. In: STOC 2009, pp. 39–44. ACM (2009)
14. Efremenko, K.: From irreducible representations to locally decodable codes. In: Proceedings of the 44th Symposium on Theory of Computing, STOC 2012, pp. 327–338. ACM, New York (2012)
15. Gallager, R.G.: Low Density Parity-Check Codes. Technical report. MIT (1963)
16. Gemmell, P., Lipton, R.J., Rubinfeld, R., Sudan, M., Wigderson, A.: Self-testing/correcting for polynomials and for approximate functions. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, STOC 1991, pp. 33–42. ACM, New York (1991)
17. Gemmell, P., Sudan, M.: Highly resilient correctors for polynomials. Information Processing Letters 43(4), 169–174 (1992)
18. Guo, A., Kopparty, S., Sudan, M.: New affine-invariant codes from lifting. In: ITCS (2013)
19. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin of the American Mathematical Society 43(4), 439–562 (2006)
20. Itoh, T., Suzuki, Y.: New Constructions for Query-Efficient Locally Decodable Codes of Subexponential Length. IEICE Transactions on Information and Systems E93-D(2), 263–270 (2010)

21. Katz, J., Trevisan, L.: On the efficiency of local decoding procedures for error-correcting codes. In: STOC 2000: Proceedings of the 32nd Annual Symposium on the Theory of Computing, pp. 80–86 (2000)
22. Kopparty, S., Saraf, S., Yekhanin, S.: High-rate codes with sublinear-time decoding. In: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing, pp. 167–176. ACM (2011)
23. Lipton, R.J.: Efficient checking of computations. In: Choffrut, C., Lengauer, T. (eds.) STACS 1990. LNCS, vol. 415, pp. 207–215. Springer, Heidelberg (1990)
24. Lubotzky, A., Phillips, R., Sarnak, P.: Ramanujan graphs. Combinatorica 8(3), 261–277 (1988)
25. Margulis, G.A.: Explicit group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. Problems of Information Transmission 9(1), 39–46 (1988)
26. Morgenstern, M.: Existence and explicit constructions of $q + 1$ regular ramanujan graphs for every prime power q. Journal of Combinatorial Theory, Series B 62(1), 44–62 (1994)
27. Polishchuk, A., Spielman, D.A.: Nearly-linear size holographic proofs. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, STOC 1994, pp. 194–203. ACM, New York (1994)
28. Reed, I.: A class of multiple-error-correcting codes and the decoding scheme. Transactions of the IRE Professional Group on Information Theory 4(4), 38–49 (1954)
29. Sipser, M., Spielman, D.A.: Expander codes. IEEE Transactions on Information Theory 42(6), 1710–1722 (1996)
30. Spielman, D.A.: Linear-time encodable and decodable error-correcting codes. IEEE Transactions on Information Theory 42(6), 1723–1731 (1996)
31. Tanner, R.: A recursive approach to low complexity codes. IEEE Transactions on Information Theory 27(5), 533–547 (1981)
32. Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. J. ACM 55(1) (2008)
33. Yekhanin, S.: Locally Decodable Codes. Foundations and Trends in Theoretical Computer Science (2010)
34. Zemor, G.: On expander codes. IEEE Transactions on Information Theory 47(2), 835–837 (2001)

On the Complexity of Broadcast Setup*

Martin Hirt and Pavel Raykov

ETH Zurich, Switzerland
{hirt,raykovp}@inf.ethz.ch

Abstract. Byzantine broadcast is a distributed primitive that allows a specific party (called "sender") to consistently distribute a value v among n parties in the presence of potential misbehavior of up to t of the parties. Broadcast requires that correct parties always agree on the same value and if the sender is correct, then the agreed value is v. Broadcast without a setup (i.e., from scratch) is achievable from point-to-point channels if and only if $t < n/3$. In case $t \geq n/3$ a trusted setup is required. The setup may be assumed to be given initially or generated by the parties in a setup phase.

It is known that generating setup for protocols with cryptographic security is relatively simple and only consists of setting up a public-key infrastructure. However, generating setup for information-theoretically secure protocols is much more involved. In this paper we study the complexity of setup generation for information-theoretic protocols using point-to-point channels and temporarily available broadcast channels. We optimize the number of rounds in which the temporary broadcast channels are used while minimizing the number of bits broadcast with them. We give the first information-theoretically secure broadcast protocol tolerating $t < n/2$ that uses the temporary broadcast channels during only 1 round in the setup phase. Furthermore, only $\mathcal{O}(n^3)$ bits need to be broadcast with the temporary broadcast channels during that round, independently of the security parameter employed. The broadcast protocol presented in this paper allows to construct the first information-theoretically secure MPC protocol which uses a broadcast channel during only one round. Additionally, the presented broadcast protocol supports refreshing, which allows to broadcast an a priori unknown number of times given a fixed-size setup.

1 Introduction

1.1 Byzantine Broadcast

The Byzantine broadcast problem (aka Byzantine generals) is stated as follows [PSL80]: A specific party (the sender) wants to distribute a message among n parties in such a way that all correct parties obtain the same message, even when some of the parties are malicious. The malicious misbehavior is modeled by a central adversary who corrupts up to t parties and takes full control of their actions. Corrupted parties are called *Byzantine* and the remaining parties

* The full version of this paper can be found here [HR13].

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 552–563, 2013.

are called *correct*. Broadcast requires that all correct parties agree on the same value v, and if the sender is correct, then v is the value proposed by the sender. Broadcast is one of the most fundamental primitives in distributed computing. It is used to implement various protocols like voting, bidding, collective contract signing, etc. Basically, this list can be continued with all protocols for secure multi-party computation (MPC) [GMW87].

There exist various implementations of Byzantine broadcast from synchronous point-to-point communication channels with different security guarantees. In the model without trusted setup, perfectly-secure Byzantine broadcast is achievable when $t < n/3$ [PSL80, BGP92, CW92]. In the model with trusted setup, cryptographically or information-theoretically secure Byzantine broadcast is achievable for any $t < n$ [DS83, PW96].

Closely related to the broadcast problem is the consensus problem. In consensus each party holds a value as an input, and then parties agree on a common value as an output of consensus. Consensus and broadcast are reducible to each other with the help of point-to-point channels in case $t < n/2$.

1.2 Model and Definitions

Parties. We consider a setting consisting of n parties (players) $\mathcal{P} = \{P_1, \ldots, P_n\}$ with some designated party called the sender, which we denote with P_s for some $s \in \{1, \ldots, n\}$. We assume that each pair of parties is connected with a secure synchronous channel, where synchronous means that the parties share a common clock and that the message delay is bounded by a constant.

Broadcast definition. A broadcast protocol allows the sender P_s to distribute a value v_s among a set of parties $\mathcal{P}$ such that:

Validity: If the sender P_s is correct, then every correct party $P_i \in \mathcal{P}$ decides on the value proposed by the sender $v_i = v_s$.

Consistency: All correct parties in $\mathcal{P}$ decide on the same value.

Termination: Every correct party in $\mathcal{P}$ terminates.

Adversary. The faultiness of parties is modeled in terms of a central adversary corrupting some of the parties. The adversary can corrupt up to $t < n/2$ parties, making corrupted parties deviate from the protocol in any desired manner.
We consider *information-theoretic* security which captures the fact that the protocol may fail only with some negligible probability even in the settings where the adversary has unbounded computing power.

1.3 Broadcast with a Trusted Setup

Broadcast is achievable from point-to-point channels if and only if $t < n/3$. In order to tolerate $t \geq n/3$ a broadcast protocol must additionally use a trusted setup which is provided to the parties beforehand. Such a trusted setup may be assumed to be a part of the model or to be distributed by a temporarily available trusted party. In this paper we consider the alternative case where

parties themselves generate the setup using point-to-point communication and temporarily available during a setup phase broadcast channels.

Formally, a *broadcast scheme* is a pair of protocols (Setup, Broadcast), where Setup generates the parties' secret states with which they start the execution of Broadcast. The Setup protocol makes uses of *temporary* broadcast and point-to-point communication channels, while Broadcast employs point-to-point channels only. It is required that the combination of Setup and Broadcast achieves broadcast defined above and Setup is independent of the sender's input v_s provided in Broadcast.

In this paper we study the efficiency of broadcast schemes, in particular the *temporary broadcast-efficiency* of the setup protocol. We employ two measures of efficiency for the temporary broadcast used in the setup protocol: *round complexity* and *bit complexity*. Round complexity denotes the number of rounds in which temporary broadcast is used and bit complexity denotes the number of bits broadcast by it.

1.4 Contributions

If computational security suffices, then in the setup phase it is enough to consistently generate a *Public-Key Infrastructure* (PKI) and then employ [DS83] to broadcast. Generating PKI requires only 1 round of temporary broadcast, in which each party broadcasts his public key. In case of information-theoretic security known solutions require $\Omega(n^2)$ rounds of temporary broadcast while broadcasting $\Omega(n^8\kappa)$ bits [PW96], where κ is a security parameter. Motivated by the gap between computational and information-theoretically secure protocols, Garay et al. [GGO12] initiated the study of information-theoretically secure broadcast schemes for $t < n/2$.[1] They focused on optimizing the temporary broadcast round complexity in the setup phase and gave a broadcast scheme which requires only 3 rounds of the temporary broadcast. The number of bits broadcast by their protocol in the setup phase is $\Omega(n^6\kappa)$. Another construction by Hirt et al. [BHR07] yields a broadcast scheme for $t < n/2$ which needs $\Omega(n)$ rounds of temporary broadcast with $\Omega(n\kappa)$ bits broadcast in the setup phase.[2]

This paper gives the first information-theoretically secure broadcast scheme that requires only 1 round of the temporary broadcast in the setup phase for $t < n/2$, which is trivially optimal. This result not only improves the broadcast round complexity of all existing broadcast schemes, but allows to construct an MPC protocol which uses only one round of broadcast (existence of such schemes was unresolved before [KK07, GGO12]).[3] Furthermore, our protocol needs to broadcast

[1] Their original motivation was to optimize the number of broadcast rounds needed for information-theoretically secure MPC. The broadcast scheme they give is used as a core component of an MPC protocol.

[2] One can view this construction as a broadcast scheme by interpreting the refresh protocol presented in this paper as the setup protocol.

[3] Additionally, this shows that the lower bound of at least 2 broadcast rounds for MPC protocols given in [GGO12] is wrong.

only $\mathcal{O}(n^3)$ bits in the setup phase *regardless* of how long the security parameter κ is. To our knowledge, this is the first protocol with such a property. Additionally, we give an efficient refresh protocol which allows to broadcast many values given a fixed setup. The table below summarizes the existing broadcast schemes:

Security	**Threshold**	**BC rounds**	**BC bits**	**Ref.**
comp.	$t < n$	1	$\Omega(n\kappa)$	[DS83]
inf.-theor.	$t < n$	$\Omega(n^2)$	$\Omega(n^8\kappa)$	[PW96]
	$t < n/2$	$\Omega(n)$	$\Omega(n\kappa)$	[BHR07]
		3	$\Omega(n^6\kappa)$	[GGO12]
		1	$\mathcal{O}(n^3)$	This paper

1.5 Organization of the Paper

First, we give a broadcast scheme for three players ($n = 3$) which tolerates any $t \leq 3$ number of corruptions (Section 2). Then, in Section 3 we show how to use the scheme for three players to obtain a broadcast scheme for arbitrary number of players n tolerating $t < n/2$ corruptions. Most proofs appear only in the full version of this paper [HR13].

2 The Broadcast Scheme for $n = 3$ and $t \leq 3$

In this section we consider a setting consisting of only *three* players and show how one can prepare a setup that allows a designated player to broadcast one bit. The broadcast scheme is based on a series of reductions among modifications of well known cryptographic primitives. On the highest level we show that: (1) temporary broadcast allows to construct information checking, (2) information checking allows to construct verifiable secret sharing, and (3) verifiable secret sharing allows to construct a setup protocol.

Additionally, we present an optimization which allows players to efficiently generate many setups in parallel. This reduces the number of bits broadcast in the setup phase while still requiring only one round of a temporary broadcast.

2.1 Detectable Information Checking

Information checking (IC) is an information-theoretically secure method for authenticating data among three players D, I, R. The dealer D holds a secret value s from some finite field $\mathbb{F}$ which he sends to an intermediary I together with an authentication information. The intended recipient R gets a verification information. Later I sends s' and some authentication information to R, who uses

the verification information to check whether $s = s'$. We propose a non-robust modification of IC which we call *Detectable Information Checking* (DIC), where the authentication phase may either *succeed* or *abort*. If it aborts then the parties output a *dispute* Δ, which is a pair of players (i.e., $\Delta \subseteq \{D, I, R\}$), at least one of them being Byzantine.

Formally, a DIC scheme is a pair[4] of protocols (ICSetup, ICReveal), where in ICSetup D inputs s and then parties either abort with a dispute or succeed by saving a local state. If ICSetup succeeds then parties invoke ICReveal such that R outputs s' or $\perp$. A DIC scheme must satisfy the following security properties:

Completeness: If D, I and R are correct, then ICSetup succeeds and R will output $s' = s$ in ICReveal.

Non-Forgery: If D and R are correct and ICSetup succeeds, then R will output $s' = s$ or $s' = \perp$ in ICReveal.

Commitment: If I and R are correct and ICSetup succeeds, then at the end of ICSetup I knows a value s' such that R will output s' in ICReveal.

Privacy: If D and I are correct, then R obtains no information on s during ICSetup.

Detection: In case ICSetup aborts every correct party outputs the same dispute Δ.

Termination: Every correct party terminates ICSetup, respectively ICReveal.

We say that a DIC scheme is information-theoretically secure if the properties above are guaranteed with overwhelming probability.

The Protocol. In this section we present a protocol for DIC based on [CDD+99]. The secret and the verification information will lie on a line (a polynomial of degree 1), where the secret will be the value in 0. Intermediary I gets to know the line, while the recipient R learns a value on this line in some secret point α unknown to I. In the reveal phase, R accepts a line from I only if the evaluation of this line in α is correct. In order to ensure commitment property I verifies whether D distributed consistent information to him and to R. This is done during one broadcast round in ICSetup. As an outcome of the broadcast parties may succeed in ICSetup or abort with a dispute.

Let $s, y, z, \alpha \in \mathbb{F}$. We say that the triple (s, y, z) is 1_α-consistent provided that three points $(0, s)$, $(1, y)$ and (α, z) lie on a line in $\mathbb{F}$ (that is, for some $L(x) = bx + c$ over $\mathbb{F}$, we have $L(0) = s$, $L(1) = y$ and $L(\alpha) = z$).

Lemma 1. *The pair of protocols* (ICSetup, ICReveal) *achieves DIC (except with probability* $\mathcal{O}(1/|\mathbb{F}|)$*). Furthermore,* ICSetup *uses the underlying broadcast channel during one predetermined round (where each player broadcasts* $\mathcal{O}(\log |\mathbb{F}|)$ *bits) and* ICReveal *does not use broadcast at all.*

[4] Sometimes (as in [CDD+99, GGO12]) IC is presented as a triple of protocols Distr, AuthVal, RevealVal where Distr and AuthVal is a more fine-grained representation of ICSetup.

Protocol ICSetup(s):

1. Dealer D chooses a random value $\alpha \in \mathbb{F} \setminus \{0,1\}$ and additional random $y, z \in \mathbb{F}$ such that (s, y, z) is 1_α-consistent. In addition it chooses a random 1_α-consistent vector (s', y', z'). D sends s, s', y, y' to I and α, z, z' to R.
2. Intermediary I chooses a random $d \in \mathbb{F}$. I sends d to D and $d, s' + ds, y' + dy$ to R. Let d_1 denote the value received by D and d_2, s'', y'' the values received by R.
3. Every player **broadcasts** the following (in parallel):
   3.1 Dealer broadcasts triple $T_D = (d_1, s' + d_1 s, y' + d_1 y)$.
   3.2 Intermediary broadcasts triple $T_I = (d, s' + ds, y' + dy)$.
   3.3 Recipient broadcasts triple $T_R = (d_2, s'', y'')$ and a bit b, where b is 1 if $(s'', y'', z' + d_2 z)$ is 1_α-consistent and 0 otherwise.
4. Every player checks for abortion:
   If $T_D \neq T_I$ then abort with $\Delta = \{D, I\}$.
   If $T_R \neq T_I$ then abort with $\Delta = \{R, I\}$.
   If $T_D = T_R$ and $b = 0$ then abort with $\Delta = \{D, R\}$.
   Otherwise, the protocol succeeds and D stores nothing, I stores (s, y) and R stores (α, z).

Protocol ICReveal(s):

1. Intermediary I sends s, y to R. If (s, y, z) is 1_α-consistent then R decides on s, otherwise on $\perp$.

2.2 Detectable Verifiable Secret Sharing

Verifiable secret sharing (VSS) is a classical cryptographic primitive for secure sharing of a secret. It lies in a core of many protocols for multi-party computation and is used in various applications. In this section we consider a very restricted setting for VSS consisting of only three players D, P_1, P_2. The dealer D holds a secret value s from some finite field $\mathbb{F}$ and shares it among two recipients P_1, P_2, such that individually they have no information about s. Later in the reconstruction phase all correct recipients reconstruct the same s' which equals s if the dealer is correct. We consider a non-robust version of VSS which we call *Detectable Verifiable Secret Sharing* (or short DVSS), where the sharing phase can *abort* in the presence of malicious behavior.

Formally, a DVSS scheme is a pair of protocols (VSS-Share, VSS-Rec), where in VSS-Share D inputs s and then parties either abort with a dispute Δ or succeed by saving a local state. If VSS-Share succeeds then the parties invoke VSS-Rec such that the recipients reconstruct the shared secret. A DVSS scheme must satisfy the following security properties:

Correctness: If VSS-Share succeeds, then there exists a fixed value $s' \in \mathbb{F}$ which will be reconstructed as a result of VSS-Rec by every correct recipient. If D is correct then $s' = s$.

Privacy: If D is correct, then corrupted parties obtain no information on s in VSS-Share.

Detection: If D, P_1 and P_2 are correct then VSS-Share always succeeds. In case VSS-Share aborts every correct party outputs the same dispute Δ.

Termination: Every correct party terminates VSS-Share, respectively VSS-Rec.

We say that a DVSS scheme is information-theoretically secure if the properties above are guaranteed with overwhelming probability. Furthermore, by t-DVSS

we denote a DVSS scheme tolerating $\leq t$ corruptions, and by t-DVSS$^+$ we denote a scheme which is t-DVSS but Detection and Termination hold for arbitrary number of corruptions.

The Protocol. In this section we present an implementation of 1-DVSS$^+$ based on DIC. In order to share a secret s the dealer generates two random values s_1, s_2 such that $s_1 + s_2 = s$. Then the dealer authenticates s_1 by invoking $\mathsf{ICSetup}(s_1)$ where P_1 acts as intermediary and P_2 as recipient. In parallel, the dealer authenticates s_2 by invoking $\mathsf{ICSetup}(s_2)$ where P_2 acts as intermediary and P_1 as recipient. If one of the ICSetup invocations aborts then VSS-Share aborts as well. The reconstruction consists of running ICReveal among P_1, P_2 and the dealer sending the secret s to both recipients. If a correct recipient obtains non-$\perp$ value in ICReveal, then it reconstructs $s' = s_1 + s_2$, otherwise it outputs $s' = s$ received from the dealer.

Protocol VSS-Share(s):

1. The dealer D chooses random $s_1, s_2 \in \mathbb{F}$ such that $s = s_1 + s_2$.
2. Then D, P_1, P_2 execute in parallel $\mathsf{ICSetup}(s_1)$ where P_1 is intermediary and P_2 is recipient and $\mathsf{ICSetup}(s_2)$ where P_2 is intermediary and P_1 is recipient.
3. Every player checks for abortion:
   If any of ICSetup aborts, then VSS-Share aborts as well with a dispute Δ output by one of ICSetup (in case both aborts, then output the dispute from the first).
   Otherwise, the protocol succeeds and each player saves the states obtained from both ICSetup invocations.

Protocol VSS-Rec(s):

1. The dealer D sends s to both recipients.
2. Players P_1, P_2 invoke $\mathsf{ICReveal}(s_1)$ and $\mathsf{ICReveal}(s_2)$. If P_i obtains a share $s_j \neq \perp$ from the other recipient P_j then it decides on $s_i + s_j$, otherwise it decides on a value s received from the dealer.

Lemma 2. *The pair of protocols* (VSS-Share, VSS-Rec) *achieves* 1-*DVSS*$^+$ *(except with probability* $\mathcal{O}(1/|\mathbb{F}|)$*). Furthermore,* VSS-Share *uses the underlying broadcast channel in only one predetermined round (where each player broadcasts* $\mathcal{O}(\log |\mathbb{F}|)$ *bits) and* VSS-Rec *does not use broadcast at all.*

Proof. First, we show that each 1-DVSS$^+$ property is satisfied:

Correctness: We prove that Correctness holds when the number of corrupted parties $t \leq 1$. If all parties are correct ($t = 0$) then, due to the Completeness property of DIC, correct recipients will output $s_1 + s_2 = s$ in VSS-Rec. Assume now only one party is corrupted ($t = 1$). Consider two cases: (1) *the dealer is corrupted* and (2) *one of the recipients is corrupted*. In case (1), due to the Commitment property of DIC, player P_1 knows s_1 such that P_2 outputs s_1 in ICReveal, and P_2 knows s_2 such that P_1 outputs s_1 in ICReveal (except with probability $\mathcal{O}(1/|\mathbb{F}|)$). Hence, since both P_1 and P_2 are correct they will both output $s' = s_1 + s_2$ in VSS-Rec. In case (2), wlog assume that a correct recipient is P_1. Due to the Non-Forgery property of DIC, player P_1 may obtain in ICReveal only s_2 or $\perp$ (except with probability $\mathcal{O}(1/|\mathbb{F}|)$). In both cases P_1 outputs $s' = s_1 + s_2$ or $s' = s$, which is the same for a correct D.

PRIVACY: Wlog assume that a corrupted recipient is P_2 and P_1 is correct. Due to the Privacy property of DIC, player P_2 who acts as a recipient in $\mathsf{ICSetup}(s_1)$ has no information about s_1 in the end of $\mathsf{ICSetup}$. Hence, after $\mathsf{VSS\text{-}Share}$ P_2's view contains only s_2 which is independent of s.

DETECTION: If all parties are correct then, due to the Completeness property of DIC, both $\mathsf{ICReveal}(s_1)$ and $\mathsf{ICReveal}(s_2)$ succeed and hence $\mathsf{VSS\text{-}Share}$ succeeds. Parties abort if and only if one of $\mathsf{ICSetup}$ aborts. Due to the Detection property of $\mathsf{ICSetup}$ parties output the same dispute Δ.

TERMINATION: Due to their specifications, protocols $\mathsf{VSS\text{-}Share}$ and $\mathsf{VSS\text{-}Rec}$ always terminate.

Finally, the protocol $\mathsf{VSS\text{-}Share}$ uses the underlying broadcast channel only at Step 2 where $\mathsf{ICSetup}$ uses it, while the protocol $\mathsf{VSS\text{-}Rec}$ does not broadcast. Since two instances of $\mathsf{ICSetup}$ are invoked in parallel, according to Lemma 1 there is only one predetermined round where the underlying broadcast channel is invoked and each player broadcasts $\mathcal{O}(\log |\mathbb{F}|)$ bits.

2.3 Broadcast Scheme

We consider three players $\{D, P_1, P_2\}$, where D is the sender (also called the dealer) and P_1, P_2 are the recipients. In the setup phase parties execute protocol Setup_3 which makes use of a temporary broadcast channel. The setup generation may abort by outputting a dispute Δ. Later the dealer D can broadcast a bit value with the protocol $\mathsf{Broadcast}_3$ using the setup created. This is done differently depending on whether the preceding Setup_3 aborted or succeeded.

The Protocol. The protocol for generating a setup uses 1-DVSS$^+$ together with *Message Authentication Codes* (MACs). We employ a MAC scheme which uses a preshared key. In the setup phase, the dealer shares a random key from some finite field $\mathbb{F}$ using $\mathsf{VSS\text{-}Share}$. In order to broadcast a message later, the dealer sends the message together with its authentication information to recipients. Then the recipients exchange the messages they received from the dealer. It is guaranteed that if both recipients are correct then they hold the same set of at most two messages. Then the parties invoke $\mathsf{VSS\text{-}Rec}$ and learn the key for the MAC scheme used. Recipients decide on the message with a valid MAC, respectively on $\perp$ if no/both messages match.

In order to construct an information-theoretically secure MAC scheme, we employ a trivially unforgeable authentication scheme for bits. To authenticate a message consisting of one bit b the dealer sends to the recipients P_1, P_2 either the left part of the preshared key ($b = 0$) or its right part ($b = 1$).

For the simplicity of the presentation, let $\overline{P_1}$ denote P_2 and $\overline{P_2}$ denote P_1. Additionally, let $\mathbb{F}$ be a finite field of $2^{2\kappa}$ elements interpreted as bit strings of length 2κ.

Protocol Setup_3:

1. The dealer generates random $k_0||k_1 \in \{0,1\}^{2\kappa}$.
   Parties execute $\mathsf{VSS\text{-}Share}(k_0||k_1)$. If $\mathsf{VSS\text{-}Share}$ aborts, then abort as well and output the dispute Δ.

Protocol $\mathsf{Broadcast}_3(b)$:

1. If Setup_3 succeeded
   - 1.1 The dealer sends k_b to both P_1, P_2. Denote the values received by the players P_1 and P_2 with a_1 and a_2, respectively.
   - 1.2 The recipients exchange a_1 and a_2 and form a set of authenticators $A = \{a_1, a_2\}$.
   - 1.3 The players execute $\mathsf{VSS\text{-}Rec}$ and get $k_0||k_1$ as an output.
   - 1.4 The recipients decide on 0 if $k_0 \in A$ and on 1 otherwise. The dealer decides on b.
2. else Setup_3 aborted with Δ then
   - 2.1 If $\Delta = \{D, P_i\}$ then the dealer sends b to $\overline{P_i}$ who forwards it to P_i. All parties decide on the values received. The dealer decides on b.
   - 2.2 If $\Delta = \{P_1, P_2\}$ then the dealer sends b to P_1 and P_2. All parties decide on the values received. The dealer decides on b.

Lemma 3. *The protocol* $\mathsf{Broadcast}_3$ *achieves broadcast (of b) given that* Setup_3 *has been executed before (except with probability* $\mathcal{O}(2^{-\kappa})$*). Furthermore,* Setup_3 *is independent of b and uses the temporary available broadcast channel in only one predetermined round (where each party broadcasts* $\mathcal{O}(\kappa)$ *bits).*

2.4 Efficient Parallel Setup

In the previous section we have shown how to generate a setup for one bit broadcast. An obvious approach for generating a setup for an ℓ bit message is to prepare ℓ such setups. In this section we show how to efficiently parallelize ℓ setup invocations such that the temporary broadcast is used only small number of times. As a key ingredient of such a parallelization, we present the protocol $\mathsf{BCFromTenBits}_3$ which given an opportunity to broadcast 10 bits allows to broadcast a message of arbitrary fixed length. The protocol is perfectly secure, i.e., its security depends only on the security of the underlying broadcast.

The protocol $\mathsf{BCFromTenBits}_3$ works recursively. At each iteration the parties reduce the broadcast of a long message to broadcasting a shorter message. The recursion stops once the dealer is supposed to broadcast 10 bits only. Each recursive iteration has the same structure: first the dealer distributes the long message using point-to-point channels, then the recipients exchange the messages received from the dealer, and afterwards each recipient forwards to the dealer the message received from the other recipient. Finally, the dealer broadcasts a hint that allows each correct recipient to decide on one of the messages he has received. The hint consists of a key k for a special *c-identifying predicate* (see later). The task of broadcasting a long message is now reduced to broadcasting the key k which has a smaller bit-length than the message.

Identifying Predicates. An identifying predicate allows to identify a specific element v from some small subset $S \subseteq \mathcal{D}$ where $\mathcal{D}$ is a potentially large domain. More formally, a c-identifying predicate is a function $Q : \mathcal{D} \times \mathcal{K} \to \{0, 1\}$ such that for any $S \subseteq \mathcal{D}$ with $|S| \leq c$ and any value $v \in S$ there exists a key $k \in \mathcal{K}$ with $Q(v, k) = 1$ and $Q(v', k) = 0$ for all $v' \in S \setminus \{v\}$. The goal of constructing such a function Q is to have $|\mathcal{K}|$ as small as possible given c and $|\mathcal{D}|$.

Assume now $\mathcal{D} = \{0,1\}^\ell$, then Q can be constructed as following: for a set S and a value v, we find $c-1$ bit positions $p_1, \ldots, p_{c-1}$ such that v differs from any other value $v' \in S$ in some position p_i. Then the key k for a set S and a value v is defined by positions $p_1, \ldots, p_{c-1}$ and bits $b_1, \ldots, b_{c-1}$ which v has at these positions. Hence, for this Q the key space $\mathcal{K} = \{0,1\}^{(c-1)(\lceil \log \ell \rceil + 1)}$.

Protocol $\mathsf{BCFromTenBits}_3(v \in \{0,1\}^\ell)$:

1. If $\ell \leq 10$ then the parties use the underlying broadcast given in order to broadcast v.
2. Otherwise:
   - 2.1 The dealer sends v to P_1 and P_2. Denote the values received by v_1 and v_2.
   - 2.2 P_1 sends v_1 to P_2, denoted by v_{12}, and P_2 sends v_2 to P_1, denoted by v_{21}. P_1 forms the set $V_1 = \{v_1, v_{21}\}$ and P_2 forms the set $V_2 = \{v_2, v_{12}\}$.
   - 2.3 The recipients send v_{21} and v_{12} to the dealer. Denote received values by v_{210} and v_{120}. Let $S = \{v, v_{120}, v_{120}\}$.
   - 2.4 The dealer chooses a key k for the 3-identifying predicate Q with the domain $\{0,1\}^\ell$, the set of values S and the value v. The parties invoke $\mathsf{BCFromTenBits}_3(k)$ recursively. Let k' denote the result of the broadcast.
   - 2.5 Each recipient P_i decides on a unique $v' \in V_i$ such that $Q(v', k') = 1$ (if $k' = \perp$ or none/both values in V_i have Q equal to 1, then decide on $\perp$). The dealer decides on v.

Lemma 4. *The protocol* $\mathsf{BCFromTenBits}_3$ *perfectly secure achieves broadcast (of* v *from a predetermined domain* $\{0,1\}^\ell$*) given that the dealer can broadcast 10 bits. Furthermore, the dealer needs to broadcast 10 bits only in one predetermined round which index depends only on* ℓ*.*

Parallelizing Setup_3**.** We run many instances of Setup_3 in parallel such that each player consolidates the values it needs to broadcast in one string which is broadcast with $\mathsf{BCFromTenBits}_3$. The protocol $\mathsf{BCFromTenBits}_3$ uses temporary broadcast as its underlying primitive for broadcasting 10 bits. The following lemma summarizes the properties achieved by this construction:

Lemma 5. *The broadcast scheme described above generates* ℓ *setups. Furthermore, the temporary broadcast is used in one predetermined round only where each player broadcasts at most 10 bits.*

3 Broadcast Scheme for any n and $t < n/2$

Broadcast is achievable from point-to-point channels without a trusted setup if and only if $t < n/3$. Fitzi and Maurer [FM00] proposed a construction of a broadcast protocol for $t < n/2$ from the broadcast channels among every triple of players. In this paper we use following theorem from [FM00]:

Theorem 1. *In the model where broadcast is available among every triple of players there is a protocol that implements broadcast among* n *players tolerating* $t < n/2$ *corruptions. This protocol invokes underlying triple broadcast channels at most* n *times for each triple* (P_i, P_j, P_k)*, where* P_i *is the sender and* P_j, P_k *are the recipients.*

The protocol we present generates n setups for each triple of players where P_i is the sender and P_j, P_k are the recipients. It is done by invoking in parallel procedure Setup_3.

Protocol Setup_n:
1. For each possible sender P_i and recipients P_j, P_k (i, j, k are all different): Parties P_i, P_j, P_k invoke Setup_3 n times in parallel.

Protocol $\mathsf{Broadcast}_n(b)$:
1. Use protocol by [FM00], whenever P_i needs to broadcast 1 bit among P_j, P_k use the protocol $\mathsf{Broadcast}_3$ with the prepared setup.

Theorem 2. *The protocol* $\mathsf{Broadcast}_n$ *achieves broadcast (of b) for* $t < n/2$ *given that* Setup_n *has been executed before (except with probability* $\mathcal{O}(n^4 2^{-\kappa})$*). Furthermore,* Setup_n *is independent of b and uses temporary broadcast procedure in only one predetermined round (where players needs to broadcast* $\mathcal{O}(n^3)$ *bits).*

4 Conclusions

In this paper we study the efficiency of information-theoretically secure broadcast schemes in terms of the temporary broadcast usage during the setup phase. All known schemes [PW96, BHR07, GGO12] use temporary broadcast in strictly more than one round in the setup phase. We give a broadcast scheme for $t < n/2$ which requires only 1 round of the temporary broadcast. Furthermore, the presented scheme requires only $\mathcal{O}(n^3)$ bits to be broadcast in the setup phase.

One of the most important applications of broadcast schemes is a secure MPC [GMW87]. In the settings with $t < n/2$ MPC is achievable from point-to-point communication only when a broadcast channel is available additionally. Such a broadcast channel is usually simulated with a broadcast scheme which tolerates $t < n/2$. The broadcast scheme presented in this paper shows that there is an information-theoretically secure MPC protocol which uses a broadcast channel during only one round. This is achieved by running Setup_n sufficiently many times in parallel (this uses one round of temporary broadcast) to prepare enough setups, and simulating all invocations to broadcast in the MPC protocol by $\mathsf{Broadcast}_n$. Furthermore, if it is not known beforehand how many times the broadcast channel will be used, a *refresh* protocol is used. It allows to generate arbitrary many setups from a given fixed-size setup. We describe a refresh protocol for our scheme in the full version of this paper [HR13].

References

[BGP92] Berman, P., Garay, J.A., Perry, K.J.: Bit optimal distributed consensus. In: Comp. Sc. Res., pp. 313–322. Plenum Publ. Corp., NY (1992)

[BHR07] Beerliová-Trubíniová, Z., Hirt, M., Riser, M.: Efficient Byzantine agreement with faulty minority. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 393–409. Springer, Heidelberg (2007)

[CDD+99] Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)

[CW92] Coan, B.A., Welch, J.L.: Modular construction of a byzantine agreement protocol with optimal message bit complexity. Inf. and C. 97, 61–85 (1992)

[DS83] Dolev, D., Strong, H.R.: Authenticated algorithms for Byzantine agreement. SIAM Journal on Computing 12(4), 656–666 (1983)

[FM00] Fitzi, M., Maurer, U.: From partial consistency to global broadcast. In: STOC 2000, pp. 494–503. ACM, New York (2000)

[GGO12] Garay, J.A., Givens, C., Ostrovsky, R.: Broadcast-efficient secure multiparty computation. IACR Cryptology ePrint Archive, 2012:130 (2012)

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC 1987, pp. 218–229. ACM, New York (1987)

[HR13] Hirt, M., Raykov, P.: On the complexity of broadcast setup (2013), http://eprint.iacr.org/2013/103

[KK07] Katz, J., Koo, C.-Y.: Round-efficient secure computation in point-to-point networks. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 311–328. Springer, Heidelberg (2007)

[PSL80] Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27(2), 228–234 (1980)

[PW96] Pfitzmann, B., Waidner, M.: Information-theoretic pseudosignatures and Byzantine agreement for $t \geq n/3$. Technical report, IBM Research (1996)

On Model-Based RIP-1 Matrices*

Piotr Indyk and Ilya Razenshteyn

CSAIL, MIT

Abstract. The Restricted Isometry Property (RIP) is a fundamental property of a matrix enabling sparse recovery [5]. Informally, an $m \times n$ matrix satisfies RIP of order k in the ℓ_p norm if $\|Ax\|_p \approx \|x\|_p$ for any vector x that is k-sparse, i.e., that has at most k non-zeros. The minimal number of rows m necessary for the property to hold has been extensively investigated, and tight bounds are known. Motivated by signal processing models, a recent work of Baraniuk et al [3] has generalized this notion to the case where the support of x must belong to a given *model*, i.e., a given family of supports. This more general notion is much less understood, especially for norms other than ℓ_2.

In this paper we present tight bounds for the model-based RIP property in the ℓ_1 norm. Our bounds hold for the two most frequently investigated models: tree-sparsity and block-sparsity. We also show implications of our results to sparse recovery problems.

1 Introduction

In recent years, a new "linear" approach for obtaining a succinct approximate representation of n-dimensional vectors (or signals) has been discovered. For any signal x, the representation is equal to Ax, where A is an $m \times n$ matrix, or possibly a random variable chosen from some distribution over such matrices. The vector Ax is often referred to as the *measurement vector* or *linear sketch* of x. Although m is typically much smaller than n, the sketch Ax often contains plenty of useful information about the signal x.

A particularly useful and well-studied problem is that of *stable sparse recovery*. We say that a vector x' is k-sparse if it has at most k non-zero coordinates. The sparse recovery problem is typically defined as follows: for some norm parameters p and q and an approximation factor $C > 0$, given Ax, recover an "approximation" vector x^* such that

$$\|x - x^*\|_p \leq C \min_{k\text{-sparse } x'} \|x - x'\|_q \tag{1}$$

(this inequality is often referred to as ℓ_p/ℓ_q *guarantee*). Sparse recovery has a tremendous number of applications in areas such as compressive sensing of signals [5,6], genetic data acquisition and analysis and data stream algorithms [14,10].

* The full version of this paper is available at `http://arxiv.org/abs/1304.3604`

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 564–575, 2013.

It is known [5] that there exist matrices A and associated recovery algorithms that produce approximations x^* satisfying Equation (1) with $p = q = 1$[1] constant approximation factor C, and sketch length

$$m = O(k \log(n/k)) \tag{2}$$

This result was proved by showing that there exist matrices A with $m = O(k \log(n/k))$ rows that satisfy the *Restricted Isometry Property (RIP)*. Formally, we say that A is *a (k, ε)-RIP-p matrix*, if for every $x \in \mathbb{R}^n$ with at most k non-zero coordinates we have

$$(1 - \varepsilon)\|x\|_p \leq \|Ax\|_p \leq (1 + \varepsilon)\|x\|_p.$$

The proof of [5] proceeds by showing that (i) there exist matrices with $m = O(k \log(n/k))$ rows that satisfy (k, ε)-RIP-2 for some constant $\varepsilon > 0$ and (ii) for such matrices there exist a polynomial time recovery algorithm that given Ax produces x^* satisfying Equation 1. Similar results were obtained for RIP-1 matrices [4]. The latter matrices are closely connected to hashing-based streaming algorithms for heavy-hitter problems, see [10] for an overview.

It is known that the bound on the number of measurements in Equation (2) is asymptotically optimal for some constant C and $p = q = 1$, see [2] and [8] (building on [6,9,11,13]). The necessity of the "extra" logarithmic factor multiplying k is quite unfortunate: the sketch length determines the "compression rate", and for large n any logarithmic factor can worsen that rate tenfold. Fortunately, a more careful modeling offers a way to overcome the aforementioned limitation. In particular, after decades of research in signal modeling, signal processing researchers know that not all supports (i.e., sets of non-zero coordinates) are equally common . For example, if a signal is a function of time, large coefficients of the signal tend to occur consecutively. This phenomenon can be exploited by searching for the best k-sparse approximation x^* whose support belongs to a given "model" family of supports $\mathcal{M}_k$ (i.e., x^* is *$\mathcal{M}_k$-sparse*). Formally, we seek x^* such that

$$\|x - x^*\|_p \leq C \cdot \min_{\substack{\mathrm{supp}(x') \subseteq T \\ T \in \mathcal{M}_k}} \|x - x'\|_q \tag{3}$$

for some family $\mathcal{M}_k$ of k-subsets of $[n]$. Clearly, the original k-sparse recovery problem corresponds to the case, when $\mathcal{M}_k$ is a family of all k-subsets of $[n]$.

A prototypical example of a sparsity model is *block sparsity* [7]. Here the signal is divided into blocks of size b, and the non-zero coefficients belong to at most k/b blocks. This model is particularly useful for bursty time signals, where the "activity" occurs during a limited time period, and is therefore contained in a few blocks. Another example is *tree sparsity* [16] which models the structure of wavelet coefficients. Here the non-zero coefficients form a rooted subtree in a full binary tree defined over the coordinates.[2]. For many such scenarios the size of

[1] In fact, one can prove a somewhat stronger guarantee, referred to as the ℓ_2/ℓ_1 guarantee.

[2] See Section 2 for formal definitions of the two models.

the family $\mathcal{M}_k$ is much smaller than $\binom{n}{k}$, which in principle makes it possible to recover an approximation from fewer measurements.

An elegant and very general model-based sparse recovery scheme was recently provided in a seminal work of Baraniuk et al [3]. The scheme has the property that, for any "computationally tractable" family of supports of "small" size, it guarantees a near-optimal sketch length $m = O(k)$, i.e., without any logarithmic factors. This is achieved by showing the existence of matrices A satisfying the *model-based* variant of RIP. Formally, we say that A satisfies ε-$\mathcal{M}_k$-RIP-p if

$$(1-\varepsilon)\|x\|_p \leq \|Ax\|_p \leq (1+\varepsilon)\|x\|_p \tag{4}$$

for any $\mathcal{M}_k$-sparse vector $x \in \mathbb{R}^n$.

In [3] it was shown that there exist matrices with $m = O(k)$ rows that satisfy ε-$\mathcal{M}_k$-RIP-2 as long as (i) either $\mathcal{M}_k$ is the block-sparse model and $b = \Omega(\log n)$ or (ii) $\mathcal{M}_k$ is the tree-sparse model. This property can be then used to give an efficient algorithm that, given Ax, finds x^* satisfying a variant of the guarantee of Equation 3. However, the guarantees offered in [3], when phrased in the ℓ_1/ℓ_1 framework, results in a super-constant approximation factor $C = \Theta(\sqrt{\log n})$ [12]. The question of whether this bound can be improved has attracted considerable attention in signal processing and streaming communities. In particular, one of the problems[3] listed in the Bertinoro workshop open problem list [1] asks whether there exist matrices A with $m = O(k)$ rows that provide the ℓ_1/ℓ_1 guarantee for the tree-sparse model with some constant approximation factor C.

Our results In this paper we make a substantial progress on this question. In particular:

1. For both block-sparse and tree-sparse models, we show that there exist $m \times n$ matrices A that provide the ℓ_1/ℓ_1 guarantee for some constant approximation factor C, such that the number of measurements improves over the bound of Equation 2 for a wide range of parameters k and b. In particular we show that for the block-sparse model we can achieve $m = O(k \log_k n)$ as long as $b = \omega(\log n)$ and $k \geq 2b$. This improves over the $O(k \log(n/k))$ bound of Equation 2 for any k in this range. In particular, if $k = n^{\Omega(1)}$, we obtain $m = O(k)$. For the tree-sparse model we achieve $m = O(k \log(n/k)/\log\log(n/k))$ as long as $k = \omega(\log n)$. This also improves over the $O(k \log(n/k))$ bound of Equation 2.
   We note, however, that our results are not accompanied by efficient recovery algorithms. Instead, we show the existence of model-based RIP-1 matrices with the given number of rows. This implies that Ax contains enough information to recover the desired approximation x^* (see Section A for more details).
2. We complement the aforementioned results by showing that the measurement bounds achievable for a matrix satisfying block-sparse or tree-sparse

[3] See Question 15: Sparse Recovery for Tree Models. The question was posed by the first author.

RIP-1 property cannot be improved (i.e., our upper bounds are tight). This provides strong evidence that the number of measurements required for sparse recovery itself cannot be $O(k)$.

Our results show a significant difference between the model-based RIP-1 and RIP-2 matrices. For the ℓ_2 norm, the original paper [3] shows that the number of measurements is fully determined by the cardinality of the model. Specifically, their proof proceeds by applying the union bound over all elements of $\mathcal{M}_k$ on top of the Johnson–Lindenstrauss-type concentration inequality. This leads to a measurement bound of $m = O(k+\log|\mathcal{M}_k|)$, which is $O(k)$ for the tree-sparse or block-sparse models. In contrast, in case of the ℓ_1 norm our lower bounds show that such a "cardinality-based" argument does not apply, and the number of rows needed to achieve the RIP-1 property is substantially higher than $O(k)$. For instance, the tree-sparse case with $k = \omega(\log n)$ gives an almost optimal separation between the number of rows: $O(k)$ for $p = 2$ and $\Omega(k \log(n/k)/\log\log(n/k))$ for $p = 1$.

Our techniques Our lower bounds are obtained by relating RIP-1 matrices to novel combinatorial/geometric structures we call *generalized expanders*. Specifically, it is known [4] that any *binary* 0-1 matrix A that satisfies (k, ε)-RIP-1 is an adjacency matrix of an unbalanced (k, ε)-expander (see Section 2 for the formal definition). The notion of a generalized expander can be viewed as extending the notion of expansion to matrices that are not binary. Formally, we define it as follows.

Definition 1 (Generalized expander). *Let A be an $m \times n$ real matrix. We say that A is* a generalized (k, ε)-expander*, if all A's columns have ℓ_1-norm at most $1 + \varepsilon$, and for every $S \subseteq [n]$ with $|S| \leq k$ we have*

$$\sum_{i \in [m]} \max_{j \in S} |a_{ij}| \geq |S| \cdot (1 - \varepsilon).$$

Observe that the notion coincides with the standard notion of expansion for binary 0-1 matrices (after a proper scaling).

In this paper we show that any (not necessarily binary) RIP-1 matrix is also a generalized expander. We then use this fact to show that any RIP-1 matrix can be sparsified by replacing most of its entries by 0. This in turn lets us use counting arguments to lower bound the number of rows of such matrix.

Our upper bounds are obtained by constructing low-degree expander-like graphs. However, we only require that the expansion holds for the sets from the given model $\mathcal{M}_k$. This allows us to reduce the number of the right nodes of the graph, which corresponds to reducing the number of rows in its adjacency matrix.

2 Definitions

In this section we provide the definitions we will use throughout the text.

Definition 2 (Expander). *Let $G = (U, V, E)$ with $|U| = n$, $|V| = m$, $E \subseteq U \times V$ be a bipartite graph such that all vertices from U have the same degree d. Then we say that G is* a (k, ε)-expander*, if for every $S \subseteq U$ with $|S| \leq k$ we have*

$$|\{v \in V \mid \exists u \in S\ (u, v) \in E\}| \geq (1 - \varepsilon)d|S|.$$

Definition 3 (Model). *Let us call any non-empty subset*

$$\mathcal{M}_k \subseteq \Sigma_k = \{A \subseteq [n] \mid |A| = k\}$$

a model.

In particular, Σ_k is a model as well.

Definition 4 (Block-sparse model). *Suppose that $b, k \in [n]$. Moreover, b divides both k and n. Let us partition our universe $[n]$ into n/b disjoint blocks $B_1, B_2, \ldots, B_{n/b}$ of size b. We consider the following* block-sparse model*: $\mathcal{B}_{k,b}$ consists of all unions of k/b blocks.*

Definition 5 (Tree-sparse model). *Suppose that $k \in [n]$ and $n = 2^{h+1} - 1$, where h is a non-negative integer. Let us identify the elements of $[n]$ with the vertices of a full binary tree of depth h. Then,* tree-sparse model *$\mathcal{T}_k$ consists of all subtrees of size k that contain the root of the full binary tree.*

Definition 6 (Model-sparse vector/set). *Let $\mathcal{M}_k \subseteq \Sigma_k$ be any model. We say that a set $S \subseteq [n]$ is* $\mathcal{M}_k$-sparse*, if S lies within a set from $\mathcal{M}_k$. Moreover, let us call a vector $x \in \mathbb{R}^n$* $\mathcal{M}_k$-sparse*, if its support is a $\mathcal{M}_k$-sparse set.*

It is straightforward to generalize the notions of RIP-p matrix, expanders and generalized expanders to the case of $\mathcal{M}_k$-sparse vectors and sets. Let us call the corresponding objects *ε-$\mathcal{M}_k$-RIP-p matrix*, *ε-$\mathcal{M}_k$-expander* and *generalized ε-$\mathcal{M}_k$-expander*, respectively. Clearly, the initial definitions correspond to the case of Σ_k-sparse vectors and sets.

Our two main objects of interest are $\mathcal{B}_{k,b}$- and $\mathcal{T}_k$-RIP-1 matrices.

3 Sparsification of RIP-1 Matrices

In this section we show that any $n \times m$ matrix, which is (k, ε)-RIP-1, can be sparsified after removing $(1 - \Omega(1))n$ columns (Theorem 1). Then we state an obvious generalization of this fact (Theorem 2), which will be useful for proving lower bounds on the number of rows for $\mathcal{B}_{k,b}$- and $\mathcal{T}_k$-RIP-1 matrices.

Theorem 1. *Let A be any $m \times n$ matrix, which is (k, ε)-RIP-1. Then there exists an $m \times \Omega(n)$ matrix B which is $(k, O(\varepsilon))$-RIP-1, has at most $O(m/k)$ non-zero entries per column and can be obtained from A by removing some columns and then setting some entries to zero.*

We prove this theorem via the sequence of lemmas. First we prove that for every matrix A there exists a ± 1-vector x such that $\|Ax\|_1$ is small.

Lemma 1. *Let A be any $m \times k$ matrix. Then there exists a vector $x \in \{-1,1\}^k$ such that*

$$\|Ax\|_1 \leq \sum_{i\in[m]} \left(\sum_{j\in[k]} a_{ij}^2 \right)^{1/2}. \tag{5}$$

Proof. Let us use a probabilistic argument. Namely, let us sample all coordinates x_i independently and uniformly at random from $\{-1,1\}$. Then

$$\mathrm{E}\left[\|Ax\|_1\right] = \sum_{i\in[m]} \mathrm{E}\left[\left|\sum_{j\in[k]} a_{ij}x_j\right|\right] \leq \sum_{i\in[m]} \left(\mathrm{E}\left[\left(\sum_{j\in[k]} a_{ij}x_j\right)^2\right]\right)^{1/2} =$$

$$= \sum_{i\in[m]} \left(\mathrm{E}\left[\sum_{j\in[k]} a_{ij}^2 x_j^2\right]\right)^{1/2} = \sum_{i\in[m]} \left(\sum_{j\in[k]} a_{ij}^2\right)^{1/2}.$$

Thus, there exists a vector $x \in \{-1,1\}^k$ that satisfies (5). □

As a trivial corollary we have the following statement.

Corollary 1. *Let A be any $m \times k$ matrix that preserves (up to $1 \pm \varepsilon$) ℓ_1-norms of* all *vectors. Then*

$$\sum_{i\in[m]} \left(\sum_{j\in[k]} a_{ij}^2 \right)^{1/2} \geq (1-\varepsilon)k.$$

The next lemma shows that every (k,ε)-RIP-1 matrix is a generalized $(k, O(\varepsilon))$-expander. This is a generalization of a theorem from [4].

Lemma 2. *Let A be any $m \times n$ matrix, which is (k,ε)-RIP-1. Then, A is a generalized $(k, 3\varepsilon)$-expander.*

Proof. For the proof we need the following lemma.

Lemma 3. *For any $y \in \mathbb{R}^k$*

$$\|y\|_1 - \|y\|_\infty \leq \left(1 + \frac{1}{\sqrt{2}}\right)(\|y\|_1 - \|y\|_2). \tag{6}$$

Proof. Clearly, if $y = 0$, then the desired inequality is trivial. Otherwise, by homogenity we can assume that $\|y\|_1 = 1$. If $\|y\|_\infty = 1$, then $\|y\|_2 = 1$, and both sides of (6) are equal to zero. So, we can assume that $\|y\|_\infty < 1$. Suppose that $\|y\|_\infty = t$ for some $t \in (0;1)$. If $1/n > t \geq 1/(n+1)$ (thus, $n = \lceil 1/t - 1 \rceil$) for some positive integer n, then, clearly, $\|y\|_2 \leq \sqrt{nt^2 + (1-nt)^2}$. One can check using elementary analysis that for every $t \in (0;1)$

$$\frac{1-\|y\|_\infty}{1-\|y\|_2} \leq \frac{1-t}{1-\sqrt{\left\lceil \frac{1}{t} - 1 \right\rceil t^2 + \left(1 - \left\lceil \frac{1}{t} - 1 \right\rceil t\right)^2}} \leq 1 + \frac{1}{\sqrt{2}}$$

(equality is attained on $t = 1/2$). This concludes the proof. □

Let $S \subseteq [n]$ be any subset of size at most k. For any $i \in [m]$ let us denote $y_i = (a_{ij})_{j \in S} \in \mathbb{R}^S$.

We have

$$\begin{aligned}\sum_{i\in[m]} \|y_i\|_\infty &\geq \left(1+\frac{1}{\sqrt{2}}\right) \sum_{i\in[m]} \|y_i\|_2 - \frac{1}{\sqrt{2}} \cdot \sum_{i\in[m]} \|y_i\|_1 \quad \text{(by Lemma 3)} \\ &\geq \left(1+\frac{1}{\sqrt{2}}\right)(1-\varepsilon)|S| - \frac{1}{\sqrt{2}} \cdot (1+\varepsilon)|S| \quad \text{(by Corollary 1 and RIP-1)} \\ &= (1-(1+\sqrt{2})\varepsilon)|S|.\end{aligned}$$

So, A is a generalized $(k, (1+\sqrt{2})\varepsilon)$-expander. Since $1+\sqrt{2} < 3$, this concludes the proof.

□

Finally, we prove Theorem 1.

Proof (Proof of Theorem 1). By Lemma 2 A is a generalized $(k, 3\varepsilon)$-expander. Let us partition $[n]$ into n/k disjoint sets of size k arbitrarily: $[n] = S_1 \cup S_2 \cup \ldots \cup S_{n/k}$. Now for every $i \in [m]$ and every S_t let us zero out all the entries a_{ij} for $j \in S_t$ except one with the largest absolute value. Let A' be the resulting matrix. Since A is a generalized $(k, 3\varepsilon)$-expander, we know that the (vector) ℓ_1 norm of the difference $A - A'$ is at most $3\varepsilon n$. Thus, each column of $A - A'$ has the ℓ_1 norm of at most 3ε *on the average*. The number of non-zero entries in $A'$is at most mn/k, so a column has at most m/k non-zero entries *on the average*. Thus, by Markov inequality there is a set of $n/3$ columns such that we have moved each of them by at most 9ε and each of them contains at most $3m/k$ non-zero entries. We define a matrix B that consists of these columns. Since we have modified each of these columns by at most 9ε and A is (k, ε)-RIP-1 we have that B is $(k, 10\varepsilon)$-RIP-1. □

The following theorem is a straightforward generalization of Theorem 1. It can be proved via literally the same argument.

Theorem 2. *Suppose that a model $\mathcal{M}_k \subseteq \Sigma_k$ has the following properties:*

- *for some $l \leq k$ all sets from Σ_l are $\mathcal{M}_k$-sparse;*
- *there exists a partition of an $\Omega(1)$-fraction of $[n]$ into disjoint subsets of size $\Omega(k)$ such that each of these subsets is $\mathcal{M}_k$-sparse.*

Then if A is an $m \times n$ matrix which is ε-$\mathcal{M}_k$-RIP-1 for some sufficiently small $\varepsilon > 0$, there exists an $m \times \Omega(n)$ matrix B which is $(l, O(\varepsilon))$-RIP-1, has at most $O(m/k)$ non-zero entries per column and can be obtained from A by removing some columns and then setting some entries to zero.

4 Lower Bounds for Model-Based RIP-1 Matrices

In this section we prove lower bounds on the number of rows for $\mathcal{B}_{k,b}$- and $\mathcal{T}_k$-RIP-1 matrices.

This is done using the following general theorem.

Theorem 3. *If a model $\mathcal{M}_k \subseteq \Sigma_k$ satisfies the statement of Theorem 2 and A is an $m \times n$ matrix which is ε-$\mathcal{M}_k$-RIP-1 for some sufficiently small $\varepsilon > 0$, then*

$$m = \Omega\left(k \cdot \frac{\log(n/k)}{\log(k/l)}\right).$$

The proof is a combination of Theorem 2 and a counting argument similar to one used in [15].

First, we need the following standard geometric fact. See the full version for the proof.

Theorem 4. *Let $v_1, v_2, \ldots, v_n \in \mathbb{R}^d$ be a set of d-dimensional vectors such that*

- *for every $i \in [n]$ we have $\|v_i\|_1 \leq 1.1$;*
- *for every $i \neq j \in [n]$ we have $\|v_i - v_j\|_1 \geq 0.9$.*

Then, $n \leq 4^d$.

The next theorem shows a tradeoff between m and column sparsity for any RIP-1 matrix. Its variant was proved in [15], but we present here the proof for the sake of completeness.

Theorem 5 ([15]). *Let A be an $m \times n$ matrix, which is (k, ε)-RIP-1 for some sufficiently small $\varepsilon > 0$. Moreover, suppose that every column of A has at most s non-zero entries. Then*

$$s \log\left(\frac{m}{sk}\right) = \Omega\left(\log\left(\frac{n}{k}\right)\right).$$

Proof. We need a lemma from [15], which is proved by a standard probabilistic argument.

Lemma 4. *There exists a set $X \subseteq \mathbb{R}^n$ of $k/2$-sparse vectors such that*

- *$\log |X| = \Omega(k \log(n/k))$;*
- *every vector from X has a unit ℓ_1-norm;*
- *all pairwise ℓ_1-distances between the elements of X are at least 1.*

Now let us see how A acts on the elements of X. Clearly, for every $x \in X$ the vector Ax is sk-sparse. By pigeonhole principle we have that for some $S \subseteq [m]$ with $|S| \leq sk$ there exists a subset $X' \subseteq X$ with

$$|X'| \geq \frac{|X|}{\binom{m}{sk}} \tag{7}$$

such that for every $x \in X'$ the support of Ax lies within S.

On the other hand, since A is (k, ε)-RIP-1 one can easily see that the set $\{Ax\}_{x \in X'}$ (which lies in the sk-dimensional subspace) has the following properties:

- every vector from the set has ℓ_1-norm at most $1 + \varepsilon$;

– all pairwise distances are at least $1 - \varepsilon$.

Since this set lies in the sk-dimensional subspace by Theorem 4 its cardinality is bounded by 4^{sk} (provided that ε is sufficiently small). Thus, we have by plugging this bound into (7)

$$\frac{2^{\Omega(k \log(n/k))}}{\binom{m}{sk}} \leq 4^{sk}.$$

Now by using a standard estimate $\binom{m}{sk} \leq 2^{O(sk \log(m/sk))}$ we have the desired statement. □

Now we can finish the proof of Theorem 3.

Proof (Proof of Theorem 3). By Theorem 2 we can get an $m \times \Omega(n)$ matrix A with column sparsity $s = O(m/k)$ and which is $(l, O(\varepsilon))$-RIP-1. Then applying Theorem 5 we have $s \log(m/sl) = \Omega(\log(n/l))$. Since, $s = O(m/k)$ we get the desired bound

$$m = \Omega\left(k \cdot \frac{\log(n/k)}{\log(k/l)}\right).$$

□

Next we apply Theorem 3 to $\mathcal{B}_{k,b}$- and $\mathcal{T}_k$-RIP-1 matrices.

Theorem 6. *For any $k \geq 2b$ and sufficiently small $\varepsilon > 0$ if A is an $m \times n$ matrix which is ε-$\mathcal{B}_{k,b}$-RIP-1, then $m = \Omega(k \log_k n)$.*

Proof. Clearly, if $k \geq 2b$, then $\mathcal{B}_{k,b}$ satisfies the conditions of Theorem 2 for $l = 2$. Thus, by Theorem 3 we have

$$m = \Omega\left(k \cdot \frac{\log(n/k)}{\log k}\right) = \Omega(k \log_k n).$$

□

Theorem 7. *Let A be an $m \times n$ matrix which is ε-$\mathcal{T}_k$-RIP-1. Then, if ε is sufficiently small and $k = \omega(\log n)$,*

$$m = \Omega\left(k \cdot \frac{\log(n/k)}{\log\log(n/k)}\right).$$

Proof. The next Lemma shows that for any $k = \omega(\log n)$ the model $\mathcal{T}_k$ satisfies the first condition of Theorem 2 with $l = \Omega(k/\log(n/k))$.

Lemma 5. *Let $S \subseteq [n]$ be a subset of the full binary tree. Then there exists a subtree that contains both S and the root with at most $O(|S| \log(n/|S|))$ vertices.*

Proof. Let T be a subtree that consists of $\log |S|$ levels of the full binary tree that are closest to the root. Let T' be a subtree that is a union of T and paths from the root to all the elements of $|S|$. It is not hard to see that $|T' \setminus T| \leq |S| \log(n/|S|)$. As a result we get

$$|T'| \leq |T| + |S| \log(n/|S|) \leq O(|S| \log(n/|S|)).$$

□

The second condition of Theorem 2 is satisfied as well (here we use that $k = \omega(\log n)$). Thus, applying Theorem 3 we have

$$m = \Omega\left(k \cdot \frac{\log(n/k)}{\log\log(n/k)}\right).$$

□

5 Upper Bounds for Model-Based RIP-1 Matrices

In this section we complement the lower bounds by upper bounds. Since the proofs are quite standard probabilistic arguments, we omit them and refer the reader to the full version.

We use the following obvious modification of a theorem from [4].

Theorem 8 ([4]). *If a graph $G = (U, V, E)$ is an ε-$\mathcal{M}_k$-expander for some model $\mathcal{M}_k \subseteq \Sigma_k$, then the normalized (by a factor of d, where d is the degree of all vertices from U) adjacency matrix of G (which size is $|V| \times |U|$) is an $O(\varepsilon)$-$\mathcal{M}_k$-RIP-1 matrix.*

Thus, it is sufficient to build $\mathcal{B}_{k,b}$- and $\mathcal{T}_k$-expanders with as small m as possible. We use the standard probabilistic argument to show the existence of such graphs. Namely, for every vertex $u \in U$ we sample a subset of $[m]$ of size d (d has to be carefully chosen). Then, we connect u and all the vertices from this subset. All sets we sample are uniform (among all d-subsets of $[m]$) and independent.

Theorem 9. *For every $\varepsilon > 0$ and $b = \omega(\log n)$ there exists an ε-$\mathcal{B}_{k,b}$-RIP-1 matrix with*

$$m = O\left(\frac{k}{\varepsilon^2} \cdot \log_k n\right).$$

Theorem 10. *For every $\varepsilon > 0$ and $k = \omega(\log n)$ there exists an ε-$\mathcal{T}_k$-RIP-1 matrix with*

$$m = O\left(\frac{k}{\varepsilon^2} \cdot \frac{\log(n/k)}{\log\log(n/k)}\right).$$

We note that the requirement that b is not too small is necessary. E.g., if we had $b = 1$, then the property is equivalent to the standard (k, ε)-RIP-1 matrix, for which the upper bound of $O(k \log_k n)$ can not be achieved.

Acknowledgments. This work was supported by Akamai Presidential Graduate Fellowship, NSF, MADALGO project and Packard Foundation. The second author would like to thank Jelani Nelson for useful discussions.

References

1. Open problems in data streams, property testing, and related topics (2011), http://people.cs.umass.edu/~mcgregor/papers/11-openproblems.pdf
2. Ba, K.D., Indyk, P., Price, E., Woodruff, D.P.: Lower Bounds for Sparse Recovery. In: SODA, pp. 1190–1197 (2010)
3. Baraniuk, R.G., Cevher, V., Duarte, M.F., Hegde, C.: Model-based compressive sensing. IEEE Transactions on Information Theory 56(4), 1982–2001 (2010)
4. Berinde, R., Gilbert, A.C., Indyk, P., Karloff, H.J., Strauss, M.J.: Combining geometry and combinatorics: A unified approach to sparse signal recovery. Allerton (2008)
5. Candès, E.J., Romberg, J.K., Tao, T.: Stable signal recovery from incomplete and inaccurate measurements. Comm. Pure Appl. Math. 59(8), 1207–1223 (2006)
6. Donoho, D.L.: Compressed sensing. IEEE Transactions on Information Theory 52(4), 1289–1306 (2006)
7. Eldar, Y., Mishali, M.: Robust recovery of signals from a structured union of subspaces. IEEE Trans. Inform. Theory 55(11), 5302–5316 (2009)
8. Foucart, S., Pajor, A., Rauhut, H., Ullrich, T.: The Gelfand widths of lp-balls for $0p \leq 1$. J. Complex. 26(6), 629–640 (2010)
9. Garnaev, A.Y., Gluskin, E.D.: On widths of the Euclidean ball. Sov. Math. Dokl. 30, 200–204 (1984)
10. Gilbert, A., Indyk, P.: Sparse recovery using sparse matrices. Proceedings of IEEE (2010)
11. Gluskin, E.D.: Norms of random matrices and widths of finite-dimensional sets. Math. USSR, Sb. 48, 173–182 (1984)
12. Indyk, P., Price, E.: K-median clustering, model-based compressive sensing, and sparse recovery for earth mover distance. In: STOC, pp. 627–636 (2011)
13. Kasin, B.S.: Diameters of some finite-dimensional sets and classes of smooth functions. Math. USSR, Izv. 11, 317–333 (1977)
14. Muthukrishnan, S.: Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci. 1(2), 117–236 (2005)
15. Nachin, M.: Lower Bounds on the Column Sparsity of Sparse Recovery Matrices. MIT Undergraduate Thesis (2010)
16. Romberg, J., Choi, H., Baraniuk, R.: Bayesian tree-structured image modeling using wavelet-domain Hidden Markov Models. 10(7), 1056–1068 (2001)

A RIP-1 Yields Sparse Recovery

In this section we show improved upper bounds on the number of measurements needed to recover good block- or tree-sparse approximations with ℓ_1/ℓ_1 guarantee and constant approximation factor. This result is folklore, but we include it for completeness.

Suppose that $\mathcal{M}_k \subseteq \Sigma_k$ is some model. We say that an $m \times n$ matrix A is ε-$\mathcal{M}_k^{(2)}$-RIP-1, if for every $x \in \mathbb{R}^n$ such that $\operatorname{supp} x \subseteq S_1 \cup S_2$ for some $\mathcal{M}_k$-sparse sets S_1 and S_2 one has

$$(1-\varepsilon)\|x\|_1 \leq \|Ax\|_1 \leq (1+\varepsilon)\|x\|_1.$$

Let A be *any* ε-$\mathcal{M}_k^{(2)}$-RIP-1 matrix for a sufficiently small ε such that $\|A\|_1 \leq 1+\varepsilon$. Algorithm 1 (whose running time is exponential in n) given $y = Ax$ for some $x \in \mathbb{R}^n$ recovers a vector $x^* \in \mathbb{R}^n$ such that

$$\|x - x^*\|_1 \leq (3 + O(\varepsilon)) \cdot \min_{x' \text{ is } \mathcal{M}_k\text{-sparse}} \|x - x'\|_1. \tag{8}$$

Note that the optimization problem within the for-loop can be easily reduced

Algorithm 1. Model-based sparse recovery

Input: $y = Ax$ for some $x \in \mathbb{R}^n$
Output: a good $\mathcal{M}_k$-approximation x^* of x

```
x* ← 0
for S ⊆ [n] is an M_k-sparse set do
    x̃ ← argmin_{supp x' ⊆ S} ‖y − Ax'‖_1
    if ‖y − Ax̃‖_1 ≤ ‖y − Ax*‖_1 then
        x* ← x̃
    end if
end for
```

to a linear program. For the proof that (8) holds for x^* see the full version.

It is immediate to see that any ε-$\mathcal{B}_{2k,b}$-RIP-1 matrix is ε-$\mathcal{B}_{k,b}^{(2)}$-RIP-1. Similarly, any ε-$\mathcal{T}_{2k}$-RIP-1 matrix is ε-$\mathcal{T}_k^{(2)}$-RIP-1. Moreover, since all the singletons are both block- and tree-sparse, we have that these matrices have ℓ_1-norm at most $1+\varepsilon$. Thus, plugging Theorems 9 and 10 we get the following result.

Theorem 11. *The problem of model-based stable sparse recovery with ℓ_1/ℓ_1 guarantee and a constant approximation factor can be solved*

- *with*
$$m = O(k \log_k n)$$
measurements for $\mathcal{B}_{k,b}$, provided that $b = \omega(\log n)$;
- *with*
$$m = O\left(k \cdot \frac{\log(n/k)}{\log\log(n/k)}\right)$$
measurements for $\mathcal{T}_k$, provided that $k = \omega(\log n)$.

Robust Pseudorandom Generators*

Yuval Ishai[1,**], Eyal Kushilevitz[1,***], Xin Li[2,†], Rafail Ostrovsky[3,‡], Manoj Prabhakaran[4,§], Amit Sahai[3,¶], and David Zuckerman[5,‖]

[1] Technion
[2] University of Washington
[3] University of California, Los Angeles
[4] University of Illinois at Urbana-Champaign
[5] University of Texas at Austin

Abstract. Let $G : \{0,1\}^n \to \{0,1\}^m$ be a pseudorandom generator. We say that a circuit implementation of G is (k,q)-*robust* if for every set S of at most k wires anywhere in the circuit, there is a set T of at most $q|S|$ outputs, such that conditioned on the values of S and T the remaining outputs are pseudorandom. We initiate the study of robust PRGs, presenting explicit and non-explicit constructions in which k is close to n, q is constant, and $m >> n$. These include unconditional constructions of robust r-wise independent PRGs and small-bias PRGs, as well as conditional constructions of robust cryptographic PRGs.

In addition to their general usefulness as a more resilient form of PRGs, our study of robust PRGs is motivated by cryptographic applications in which an adversary has a local view of a large source of

* The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

** Supported by the European Research Council as part of project CaC (grant 259426).

*** Supported by ISF grant 1361/10 and BSF grant 2008411.

† Supported by a Simons postdoctoral fellowship.

‡ 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.

§ Research supported in part by NSF grants 1228856 and 0747027.

¶ Research supported in part from a DARPA/ONR PROCEED award, NSF grants 1228984, 1136174, 1118096, 1065276, 0916574 and 0830803, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389.

‖ Research supported by NSF Grants CCF-0916160, CCF-1218723, and DMS-0835373.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 576–588, 2013.

secret randomness. We apply robust r-wise independent PRGs towards reducing the randomness complexity of private circuits and protocols for secure multiparty computation, as well as improving the "black-box complexity" of constant-round secure two-party computation.

1 Introduction

Pseudorandomness is a central tool in complexity theory and cryptography. A *pseudorandom generator* (PRG) is a deterministic function $G : \{0,1\}^n \to \{0,1\}^m$ which stretches a short random seed into a longer output which looks random to any computationally bounded distinguisher. The question we ask in this work can be pictorially described as follows. Consider an implementation of G by a boolean circuit, and suppose that an attacker can observe a set S of k wires anywhere in the circuit. Since S may contain output wires, the output conditioned on S may no longer look random. But how big is the "shadow" S can cast on the output? Can we design PRG implementations in which the effect of observing any such S is localized to roughly k bits of the output?

We formalize the above question via the notion of *robust pseudorandom generators*. We say that a circuit implementation of G is (k, q)-robust if for every set S of at most k wires anywhere in the circuit there is a set T ("shadow") of at most $q|S|$ outputs such that conditioned on the values of S, the outputs outside T are pseudorandom. We will be mainly interested in a stronger notion of robustness in which the conditioning is on both S and T; if such a stronger requirement is met we say that G is *strongly* (k, q)-robust. We consider the robustness of three distinct types of PRG: *r-wise independent* PRGs, where the distinguisher can observe any r bits of the output, *small-bias* PRGs [1], where the distinguisher can compute the parity of any subset of the outputs, and *cryptographic* PRGs [2, 3], where the distinguisher can perform arbitrary polynomial-time computations.

To motivate the notion of robust PRGs, consider a simple application of cryptographic PRGs for one-time symmetric encryption. To encrypt a long message $M \in \{0,1\}^m$ with a short secret key $K \in \{0,1\}^n$, it suffices to compute $C = M \oplus G(K)$. Since $G(K)$ is indistinguishable from random, so is C. Now, suppose that $k << n$ intermediate values in the computation of C are leaked. What do these values together with C reveal about M? The dense model theorem [4–6] assures us that if G is sufficiently strong, then M is indistinguishable from some source whose min-entropy is roughly $m - k$. However, even a single lost bit of entropy can correspond to *global* information about m. For instance, if an intermediate value reveals the parity of $G(K)$, this information together with C reveals the parity of M. Our goal is to provide the guarantee that if arbitrary k physical bits are leaked during the process of computing C, this is no worse than leaking (roughly) k physical bits of M.

We turn to the question of constructing robust PRGs, starting with some simple observations. First, if $k = n$, the set S can include the entire PRG seed, conditioned on which the entire output is fixed. We thus restrict the attention to the case where $k < n$. Second, allowing n to be much bigger than k, we can

use the following naïve construction: if $G' : \{0,1\}^{n'} \to \{0,1\}^m$ is a PRG then $G : \{0,1\}^{n'(k+1)} \to \{0,1\}^m$ defined by $G(x_1, \ldots, x_{k+1}) = G'(x_1) \oplus G'(x_2) \oplus \cdots \oplus G'(x_{k+1})$ (computed in the natural way) is a strong $(k,1)$ robust PRG. The main weakness of this construction is that its seed length is far from optimal. A secondary weakness, which turns out to be crucial for one of our motivating applications, is that the *circuit size* of G is much bigger than its output length. Our main goal in this work is to construct robust PRGs in which n is very close to k, while keeping q constant and maximizing the stretch function $m(n)$. As a secondary goal, we would like to minimize the circuit size of robust PRGs. These goals are nontrivial to meet also when considering non-explicit constructions.

1.1 Our Results

We present several constructions of robust PRGs with near-optimal parameters.

- **Robust r-wise independent PRGs:** Using explicit constructions of unbalanced lossless expanders [7, 8], we get constructions of strong (k,q)-robust r-wise PRGs with $q = O(1)$ and either $r, k = \Omega(n)$ and linear stretch, or $r, k = n^{1-\eta}$ and arbitrary polynomial (or even $\exp(n^{\delta})$) stretch, for an arbitrary constant $\eta > 0$. For randomized (non-explicit) constructions, we can get an arbitrary polynomial stretch with $r, k = \Omega(n)$.
- **Robust ϵ-biased PRGs:** We get an explicit construction of a strong (k,q)-robust ϵ-biased PRG with $q = O(1)$, $k = \Omega(n)$, linear stretch, and exponentially small bias. We also get a randomized construction with a small polynomial stretch which satisfies the weaker notion of robustness.
- **Robust cryptographic PRGs:** PRG constructions with constant output and input locality [9–12] yield (n,q)-robust PRGs with linear stretch and $q = O(1)$. We show that a cryptographic PRG from [9], which has linear stretch, is robust with a better value of q (under a similar assumption).

The output locality of the above PRGs (i.e., the number of inputs on which each output depends) is at most polylogarithmic in the seed length, and their circuit size is at most quasilinear in the output length.

As discussed above, robust PRGs can be directly motivated by their usefulness as a more resilient alternative to traditional PRGs. We present several other applications of (strong) robust r-wise independent PRGs in cryptography. The high level idea behind these applications is as follows. Suppose that a cryptographic computation, which has secret inputs w and secret randomness ρ, is attacked by an adversary who can observe intermediate values in the computation. Whenever it is guaranteed that the adversary's view depends only on a small number of bits from ρ (but can arbitrarily depend on w), we can replace the true randomness ρ by pseudorandomness generated using a robust r-wise independent PRG without degrading the security of the implementation. Note that robustness is necessary here because the PRG computation becomes a part of the new implementation and hence it is also subject to attacks. We apply this idea in the following domains.

Private Circuits. A *t-private circuit* [13] is a randomized circuit which transforms a randomly encoded input into an encoded output while providing the guarantee that the joint values of any t wires reveal nothing about the input. We show that any t-private circuit in which each wire depends on at most ℓ bits of randomness can be converted via the use of robust r-wise PRGs into a t private circuit which uses roughly $t\ell$ bits of randomness. Applying this to a variant of the construction from [13], we get t-private circuits which can use $O(t^3)$ bits of randomness to protect an arbitrary poly(t)-time computation.

Secure Multiparty Computation. We show a similar application of r-wise PRGs in the related context of unconditionally secure multiparty computation. Here we improve the randomness complexity of a previous randomness-efficient protocol from [14], which implicitly relies on the naïve robust PRG construction described above.

Secure Two-Party Computation. We obtain a constant-round two-party computation protocol secure against malicious parties in which evaluating a circuit of size s with security $2^{-\kappa}$ requires only a polylogarithmic (in κ, s) number of calls to a cryptographic PRG for each gate of the circuit, where κ is a security parameter, and a small number of oblivious transfers. In fact, our protocol is *non-interactive* in a model that allows parallel oblivious transfers. This improves over previous constant-round protocols which combine Yao's garbled circuit construction with a "cut-and-choose" technique (e.g., [15]), where the number of PRG calls per gate is $O(\kappa)$. This also improves over a previous protocol from [16] in which the number of PRG calls is similar to our protocol but the number of oblivious transfers is very large (comparable to the number of PRG calls). The improvement over [16] results from implementing randomized circuits of a near-optimal size which use a small amount of randomness to make any disjunction of circuit wires or their negation essentially independent of the input. For this application, the crucial feature of our robust PRG constructions is their near-optimal *circuit size* rather than seed length.

1.2 Related Work

It is instructive to view the question we study in the broader context of leakage-resilient cryptography. The general goal in this area is to get an implementation of a cryptographic function (say, a PRG or an encryption scheme) which remains "as secure as possible" in the presence of information leakage. One way of classifying works in this area is according to the following criteria:

- What is the class of leakage functions? One may consider either **(A)** *local leakage,* where the adversary can probe k physical bits in the implementation, or **(B)** *global leakage,* where the adversary can learn arbitrary k bits of information.
- Which parts of the system leak? Here one can consider either **(1)** *confined leakage,* which applies only to part of the implementation (e.g., a secret key, a seed, or an online phase), or **(2)** *unconfined leakage,* where the leakage applies to the entire implementation.

In case (1) one can hope to offer full protection against leakage, whereas in case (2) one needs to settle for allowing a similar type of leakage in the "ideal model." That is, if arbitrary k bits of information can be leaked, the best we can hope for is that the adversary will learn k bits of information *of the same type* about the secrets.

Most work on leakage resilient cryptography falls either into category (B1) (e.g., [6, 17–20]), (A1) (e.g., [13, 21, 22]), or (B2) (e.g., [23–25]). Our work may be the first to study nontrivial questions of type (A2).

We conclude by comparing our notion of robust PRGs with other notions of robustness for PRGs considered in the literature. An *exposure resilient function* (ERF) [22] is a PRG whose output remains pseudorandom even if k physical bits of the *seed* (and the seed alone) are leaked. Thus, ERFs can be classified into category (A1). An ERF can be obtained by applying a standard PRG on top of an extractor for bit-fixing sources [26]. A natural approach for constructing a robust PRG from an ERF is to apply a private circuit compiler (such as [13]) to the ERF. This approach fails because of the high randomness complexity of private circuits. Even if one uses the randomness-efficient private circuits mentioned above, the parameters of the resulting robust PRG will end up being worse than those obtained by taking the exclusive-or of $k+1$ standard PRGs. Finally, the dense model theorem (already mentioned above) implies that *any* sufficiently strong cryptographic PRG offers leakage resilience of type (B2). That is, leaking arbitrary k bits of information about the seed is no worse than leaking roughly k bits of information about the output.

Organization. In Section 2 we give formal definitions for different variants of robust PRGs. Section 3 describes our constructions of robust r-wise independent PRGs and Section 4 describes their application to reducing the randomness complexity of private circuits. For lack of space, additional constructions and applications as well as some of the proofs are deferred to the full version.

2 Definitions

In this section we define the different notions of robust PRGs we will be interested in. We will need the following notion of "fooling" with respect to functions of varying input lengths.

Definition 1. *Let $\mathcal{F} = \bigcup_n \mathcal{F}_n$ be a class of functions, where the functions in $\mathcal{F}_n$ are from $\{0,1\}^n$ to $\{0,1\}$. A probability distribution D on $\{0,1\}^n$ is said to* ϵ-fool *$\mathcal{F}$ if for any $f \in \mathcal{F}_n$, $|\Pr[f(D)=1] - \Pr[f(U)=1]| \leq \epsilon$.*

Definition 2. *A circuit implementation C of a function $G : \{0,1\}^n \to \{0,1\}^m$ is a (k,q)-robust pesudorandom generator (PRG) for a class $\mathcal{F}$ of functions with error ϵ if the following holds. Let X be the uniform distribution over $\{0,1\}^n$ and $Y = G(X)$. For any set S of at most k wires in G, there is a set T of at most $q|S|$ output bits such that conditioned on any fixing of the values C_S of the wires in S, the distribution $Y_{\bar{T}}$ of the output bits not in T ϵ-fools $\mathcal{F}$. We say that G is*

a strong (k, q)*-robust PRG for* $\mathcal{F}$ *if conditioned on any fixing of the values* C_S *and* Y_T*, we have that* $Y_{\bar{T}}$ ϵ*-fools* $\mathcal{F}$*.*

If the implementation is understood or unimportant, we may simply say that a function is a robust PRG. This may happen if each output bit depends on only few input bits, and any implementation that does not involve using extraneous bits is robust.

When each $\mathcal{F}_n$ consists of all tests on r bits, we call a robust PRG for $\mathcal{F}$ with 0 error a robust r-wise independent PRG. When each $\mathcal{F}_n$ consists of all parities on subsets of the n bits, we call a robust PRG for $\mathcal{F}$ a robust ϵ-biased PRG. A robust *cryptographic* PRG is one which is robust for each $\mathcal{F}$ that can be computed by circuits of poly(n) size with negligible error $\epsilon(n)$.

We handle leakage of arbitrary wire values by constructing a *local* PRG which can handle leakage of inputs. In particular, we have the following definitions and simple lemma.

Definition 3. *A function* $f : \{0,1\}^n \to \{0,1\}^m$ *is* d*-local if each output bit depends on at most* d *input bits.*

Definition 4. *A function* $G : \{0,1\}^n \to \{0,1\}^m$ *is a* (k, q)-input *robust PRG for a class* $\mathcal{F}$ *of functions with error* ϵ *if the following holds. Let* X *be the uniform distribution over* $\{0,1\}^n$ *and* $Y = G(X)$*. For any set* S *of at most* k *input bits, there is a set* T *of at most* $q|S|$ *output bits such that conditioned on any fixing of the values* X_S *of the inputs* S*, the values* $Y_{\bar{T}}$ *of the output bits not in* T ϵ*-fools* $\mathcal{F}$*. We say that* G *is a strong* (k, q)*-input robust PRG for* $\mathcal{F}$ *if conditioned on any fixing of the values* X_S *and* Y_T*, we have that* $Y_{\bar{T}}$ ϵ*-fools* $\mathcal{F}$*.*

Lemma 1. *A* d*-local (strong)* (dk, q)*-input robust PRG is a (strong)* (k, dq)*-robust PRG with the same error.*

3 Robust r-Wise Independence Generators

Our construction of a robust r-wise independent PRG is simple. A bipartite graph $H = ([m], [n], E)$ induces a function $G_H : \{0,1\}^n \to \{0,1\}^m$ where the ith output bit is the parity of the input bits corresponding to the neighbors of i. That is, $G_H(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ where $y_i = \oplus_{j \in \Gamma(i)} x_j$.

We will take H to be a bipartite expander with expansion bigger than half the degree. This is defined as follows.

Definition 5. *A bipartite graph* $([m], [n], E)$ *with left vertices* $[m]$ *and right vertices* $[n]$ *is an* (ℓ, b)-expander *if for any subset* $V \subseteq [m]$ *on the left with* $|V| \leq \ell$*, we have that* $|\Gamma(V)| \geq b|V|$*.*

We can now state our theorem.

Theorem 1. *Suppose* H *is a* d*-left-regular* $(\ell, (1/2 + \gamma)d)$*-expander. Then for any constant* $0 < \alpha < 1$*, we have that* G_H *is a strong* $(\alpha\gamma\ell, 1/\gamma)$*-robust* r*-wise independent PRG, with* $r = (1 - \alpha)\ell$*.*

One surprising feature of this theorem is that the degree d of the expander does not appear anywhere explicitly. Besides expansion bigger than $d/2$, the important parameter of the expander is ℓ, the maximum size of subsets which expand. The parameter ℓ determines the robustness of the PRG. We have $\ell \leq 2n/d$, so the degree appears implicitly there.

First we state a result for random d-left-regular graphs H with $d \leq \sqrt{n}$ and $m = n^{\Omega(d)}$. In this case, for any $\gamma < 1/2$, with probability $1 - n^{-\Omega(d)}$, the random graph H is an $(\Omega(n/d), (1/2 + \gamma)d)$-expander, which is essentially best possible. This gives:

Theorem 2. *There exists $\beta > 0$ such that for any $d \leq \sqrt{n}$, for a random d-left-regular H with $m = n^{\beta d}$, with probability $1 - n^{-\beta d}$, the function $G_H : \{0,1\}^n \to \{0,1\}^m$ is a strong $(\beta n/d, 21)$-robust r-wise independent PRG for $r = \beta n$.*

We now instantiate theorem 1 with known expander constructions. Capalbo et al. [7] achieved expansion bigger than $d/2$ for constant degree graphs, with $\ell = \Omega(n/d)$. This yields:

Theorem 3. *For any constant $C > 0$ there is a constant $\beta > 0$ such that there is an explicit $O(1)$-local strong $(\beta n, 21)$-robust r-wise independent PRG $G : \{0,1\}^n \to \{0,1\}^{Cn}$ for $r = \beta n$.*

For larger stretch but lower robustness we use the expanders of Guruswami, Umans, and Vadhan [8]. They achieve $\ell = n^{1-\eta}$ for any $\eta > 0$, which gives:

Theorem 4. *For any $\eta > 0$ there exists $\delta, C > 0$ such that for any $m \leq \exp(n^\delta)$, there is an explicit d-local strong $(n^{1-\eta}, 21)$-robust r-wise independent PRG $G : \{0,1\}^n \to \{0,1\}^m$ for $r = n^{1-\eta}$ and $d \leq \log^C m$.*

We now prove theorem 1. We do this by showing that G_H is strongly $(d\alpha\gamma\ell, 1/(\gamma d))$-input-robust. Let S be the set of input bits that are fixed with $|S| = s \leq kd \leq \alpha\gamma d\ell = \alpha c\ell$.

We now consider the input bits in $\bar{S}$, which are the bits not in S. Let $x \in \{0,1\}^n$ be the input string, and $y = G(x) \in \{0,1\}^m$ be the output string. For any output bit y_i we associate with it a vector $V_i \in \{0,1\}^n$. The vector V_i has exactly d 1s at the d positions of y_i's neighbors (more precisely, y_i's corresponding vertex's neighbors), and has 0s everywhere else, i.e., V_i is the indicator vector of whether an x_j influences y_i. We then let $\bar{V}_i \in \{0,1\}^{n-s}$ be the vector that is obtained by projecting V_i into the bits that are in $\bar{S}$. Now let X be the uniform distribution over $\{0,1\}^n$, and $Y = G(X)$ be the output distribution of the PRG. The following fact is immediate.

Fact 5. *For any subset W of the output bits, we have*

$$\bigoplus_{Y_i \in W} Y_i \text{ is a constant} \iff \sum_{Y_i \in W} \bar{V}_i = 0.$$

We now have the following lemma, whose proof appears in the full version.

Lemma 2. *For any subset W of the output bits, let $\bar{W}$ be the output bits not in W. Assume that conditioned on some fixing of the input bits $\{X_i \in S\}$ and the output bits $\{Y_h \in W\}$, there exist some bits $Y_{j1}, \cdots, Y_{jl} \in \bar{W}$ such that $Y' = \bigoplus_{i=1}^{l} Y_{ji}$ is not uniform. Then Y' is a constant. Moreover, $\bar{V}' = \sum_{i=1}^{l} \bar{V}_{ji}$ is in $\mathsf{Span}(\{\bar{V}_h : Y_h \in W\})$, and vice versa.*

We now slightly abuse notation and use S to also denote the set of right vertices in H that correspond to the fixed bits. Below we borrow some techniques from [27]. We have the following definition.

Definition 6. *Suppose $H = ([m], [n], E)$ is a d-left-regular $(\ell, d/2 + c)$-expander where $c = \gamma d$. For any subset $T \subset [m]$ we let $\Delta(T) \subset [n]$ be the set of unique neighbors of T, i.e. the set of right vertices that are adjacent to only one vertex in T. For any subset $S \subset [n]$ with $|S| < c\ell$, we define an inference relation $\vdash_S$ on subsets of the left vertices as follows.*

$$T_1 \vdash_S T_2 \iff |T_2| \leq \ell - |S|/c \wedge |\Delta(T_2) \setminus [\Gamma(T_1) \cup S]| < c|T_2|.$$

We now set $T = \emptyset$ and repeat the following step as long as it is possible: if there exists a non-empty subset $T_1 \subset [m] \setminus T$ such that $T \vdash_S T_1$ then let $T = T \cup T_1$. Since the graph is finite the above procedure terminates in finite steps. We denote the final T by $Cl(S)$. We now have the following lemma.

Lemma 3. *If $|S| < c\ell$ then $|Cl(S)| \leq |S|/c$.*

Proof. Assume for the sake of contradiction that $|Cl(S)| > |S|/c$. Consider the sequence of subsets of left vertices $T_1, T_2, \cdots, T_v$ that we add to the set T. Note that all these T_i's are disjoint. Let $C_v = \cup_{i=1}^{v} T_i$ be the set of left vertices derived in v steps. Thus $|C_v| = \sum_{i=1}^{v} |T_i|$.

Let v_0 be the first v such that $|C_v| > |S|/c$. Thus $C_{v_0-1} \leq |S|/c$ and $|C_{v_0}| \leq |C_{v_0-1}| + |T_{v_0}| \leq |S|/c + \ell - |S|/c \leq \ell$. By the expansion property, C_{v_0} has at least $(d/2 + c)|C_{v_0}|$ neighbors and thus $\Delta(C_{v_0}) \geq 2(d/2 + c)|C_{v_0}| - d|C_{v_0}| = 2c|C_{v_0}|$. Therefore $|\Delta(C_{v_0}) \setminus S| \geq 2c|C_{v_0}| - |S| > c|C_{v_0}|$.

On the other hand, since each time when we add T_i to T, the number of unique neighbors in $\Delta(T) \setminus S$ increases by at most $|\Delta(T_i) \setminus [\Gamma(T) \cup S]|$, we have $|\Delta(C_{v_0}) \setminus S| < \sum_{i=1}^{v_0} c|T_i| = c|C_{v_0}|$, which is a contradiction. □

Now we have the following lemma.

Lemma 4. *Let $T = Cl(S)$. Then conditioned on any fixing of the input bits in S and any fixing of the output bits in T, the output bits that are not in T are r-wise independent, where $r = \ell - |S|/c \geq (1 - \alpha)\ell$.*

Proof. Let $\bar{T}$ be the set of output bits that are not in T. Assume that the lemma is not true. Then, for some fixing of the input bits in S and some fixing of the output bits in T, there exist $1 \leq l \leq r = \ell - |S|/c$ output bits $\{Y_{j1}, \cdots, Y_{jl} \in \bar{T}\}$ such that $\bigoplus_{i=1}^{l} Y_{ji}$ is not uniform.

By lemma 2, $\bar{V}' = \sum_{i=1}^{l} \bar{V}_{ji}$ is in $\mathsf{Span}(\{\bar{V}_h : Y_h \in T\})$. Let T_1 be the set of left vertices corresponding to $\{Y_{j1}, \cdots, Y_{jl}\}$. Then we have that $\Delta(T_1) \setminus S \subset \Gamma(T)$. Thus $|\Delta(T_1) \setminus \Gamma(T) \cup S| = 0 < c|T_1|$. This means that we can add T_1 to T in the procedure where we obtain $Cl(S)$, which contradicts the fact that $Cl(S)$ is obtained when the procedure stops. □

Note that $|T| \leq |S|/c \leq kd/c = k/\gamma$, thus the theorem is proved.

Due to lack of space, our robust ϵ-biased generators and cryptographic generators are deferred to the full version.

4 Applications

In this section and in the full version we present several applications of robust *r-wise* PRGs in cryptography. The following technical lemma captures a typical application scenario in which a strong robust r-wise PRG is used to replace a true source of randomness in cryptographic implementations in which the adversary has a local view of the randomness. The security of this approach will follow by showing that the view of any wire-probing adversary who attacks the "real world" implementation, in which a robust PRG is used to generate randomness, can be simulated given the view of a wire-probing adversary who attacks an ideal implementation which uses a true source of randomness. To simplify notation, we will use G to denote both the function computed by a robust PRG and its circuit implementation.

Lemma 5. *Let λ be a positive integer and $G : \{0,1\}^n \to \{0,1\}^m$ be a strong (k, q) robust r-wise PRG with $r \geq \max(\lambda, kq)$. Then for any set S of at most k wires in G there is a set $T \subseteq [m]$, $|T| \leq q|S|$, and a randomized algorithm* Sim *(a simulator) such that the following holds. For every $Q : \{0,1\}^m \to V$ which depends on at most λ bits of its input, the distributions* Real *and* Sim(Ideal) *are identical, where* $\mathsf{Real} = (Q(G(X)), G_S(X))$, $\mathsf{Ideal} = (Q(R), R_T)$, *$X$ is uniformly distributed over $\{0,1\}^n$, and R is uniformly distributed over $\{0,1\}^m$. Moreover, if G is linear over the binary field then* Sim *can be implemented in probabilistic polynomial time.*

Lemma 5 provides a general recipe for reducing the randomness complexity of cryptographic implementations in which the adversary's view depends on at most λ bits of randomness (but potentially also on many bits of secret data). In the next section and in the full version we give examples for such applications.

4.1 Private Circuits

A *t-private circuit* is a randomized circuit which transforms a randomly encoded input into an encoded output while providing the guarantee that the joint values of any t wires reveal nothing about the input. (For simplicity we address here the stateless variant of private circuits with encoded inputs and outputs, see [13, Section 3] and [28, Section 2.1]; our result applies to other variants as well.) We will

show that robust r-wise PRGs can be used to reduce the randomness complexity of private circuits in which each wire depends on few bits of the randomness. The latter feature can be enforced by adding a simple "rerandomization gadget" to existing constructions.

Definition 7. (Private circuit) *A private circuit for $f : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$ is defined by a triple (I, C, O), where*

- *$I : \{0,1\}^{n_i} \to \{0,1\}^{\hat{n}_i}$ is a randomized input encoder;*
- *C is a randomized boolean circuit with input $\hat{w} \in \{0,1\}^{\hat{n}_i}$, output $\hat{y} \in \{0,1\}^{\hat{n}_o}$, and randomness $\rho \in \{0,1\}^m$;*
- *$O : \{0,1\}^{\hat{n}_o} \to \{0,1\}^n_o$ is an output decoder.*

We say that C a t-private implementation of f with encoder I and decoder O if the following requirements hold:

- Correctness: *For any input $w \in \{0,1\}^{n_i}$ we have $\Pr[O(C(I(w), \rho)) = f(w)] = 1]$, where the probability is over the randomness of I and ρ.*
- Privacy: *For any $w, w' \in \{0,1\}^{n_i}$ and any set P of t wires in C, the distributions $C_P(I(w), \rho)$ and $C_P(I(w'), \rho)$ are identical.*

We say that C makes an ℓ-local use of its randomness if the value of each of its wires is determined by its input $\hat{w}$ and at most ℓ bits of the randomness ρ (where the identity of these bits may depend on the wire). Unless noted otherwise, we assume I and O to be the following canonical encoder and decoder: I encodes each input bit w_i by a block of $t+1$ random bits with parity w_i, and O takes the parity of each block of $t+1$ bits.

Note that without any requirement on I and O the above definition is trivially satisfied by having I compute a secret sharing of $f(w)$ which is passed by C to the decoder. However, applications of private circuit require the use of encoder and decoder which are independent of f.

The following theorem applies robust r-wise PRGs towards reducing the randomness complexity of private circuits.

Theorem 6. *Suppose C is a qt-private implementation of f with encoder I and decoder O, where C uses m random bits and makes an ℓ-local use of its randomness. Let $G : \{0,1\}^n \to \{0,1\}^m$ be a strong (t,q) robust r-wise PRG with $r = t \cdot \max(\ell, q)$. Then, the circuit C' defined by $C'(\hat{w}, \rho') = C(\hat{w}, G(\rho'))$ is a t-private implementation of f with encoder I and decoder O which uses n random bits.*

Proof. We show that the view of an adversary A' who attacks $C'(\hat{w}, \rho')$ by probing a set S of $t' \leq t$ wires in G and a set P of $t - t'$ additional wires in C is independent of the input w. Since C is qt-private, it suffices to show that the view of A' can be simulated given the view of an adversary A who probes at most qt wires in $C(\hat{w}, \rho)$.

Let T and Sim be as promised by Lemma 5 for $\lambda = t\ell$. For any $\hat{w}$, let $Q_{\hat{w}}(\rho) = C_P(\hat{w}, \rho)$. Since C makes an ℓ-local use of its randomness, $Q_{\hat{w}}$ depends on at most λ bits of ρ. Thus, for any fixed $\hat{w}$, we have $\mathsf{Sim}(Q_{\hat{w}}(R), R_T) \equiv$

$(Q_{\hat{w}}(G(X)), G_S(X))$, where R is uniform on $\{0,1\}^m$ and X is uniform on $\{0,1\}^n$. It follows that $\mathsf{Sim}(Q_{I(w)}(R), R_T) \equiv (Q_{I(w)}(G(X)), G_S(X))$. Since the distribution to which Sim is applied captures the view of an adversary A who corrupts a set $P \cup T$ of at most $qt' + (t - t') \leq qt$ wires in C and the distribution on the right hand side captures the view of A', it follows that the view of A' is independent of w as required. □

Theorem 6 implies that robust PRGs can be used to reduce the question of improving the randomness complexity of private circuits to that of improving their randomness locality. Luckily, known constructions such as the one from [13], while technically not satisfying the randomness locality condition, can be easily modified to have good randomness locality. Indeed, a variant of the construction of [13] (see full version) shows the following.

Lemma 6. *Any function f with circuit size s admits a t-private implementation (I, C, O) with the canonical encoder I and decoder O, where C uses $O(t^2 s)$ random bits and makes an $O(t^2)$-local use of its randomness.*

Combining Lemma 6 with Theorems 2,4 we get the following corollary.

Corollary 1. *For any polynomial $s(\cdot)$, any function f of circuit size $s(t)$ admits a t-private implementation (I, C, O) with the canonic encoder I and decoder O, where C uses $O(t^3)$ random bits (alternatively, $O(t^{3+\epsilon})$ random bits for an explicit construction of C from a circuit for f).*

Using a naive implementation of robust PRGs obtained by taking the exclusive-or of $k+1$ independent r-wise PRGs gives an $O(t^4)$ bound on the randomness. Improving the randomness locality of private circuits would immediate yield a corresponding improvement to Corollary 1.

An Alternative Model. While we considered in this section a model of private circuit which receives encoded inputs and produces encoded outputs, the results apply also to an alternative variant in which the inputs and outputs are not protected by an encoder and decoder. In this case one should settle for the following relaxed t-privacy requirement: the distribution of any set of at most t wires in C can be simulated given t bits from the input and output. To apply robust PRGs and get efficient simulation in this context, one needs to use the efficient simulation variant of Lemma 5.

See full version for applications to secure multiparty computation.

Acknowledgements. We thank Benny Applebaum for helpful comments and an anonymous reviewer for pointing out the relevance of [27].

References

1. Naor, J., Naor, M.: Small–bias probability spaces: Efficient constructions and applications. SIAM Journal on Computing 22(4), 838–856 (1993)
2. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudo-random bits. SIAM J. Comput. 13, 850–864 (1984)

3. Yao, A.C.: Theory and application of trapdoor functions. In: Proc. 23rd FOCS
4. Tao, T., Ziegler, T.: The primes contain arbitrarily long polynomial progressions (2006), http://arxiv.org/abs/math.NT/0610050
5. Reingold, O., Trevisan, L., Tulsiani, M., Vadhan, S.P.: Dense subsets of pseudorandom sets. In: FOCS, pp. 76–85 (2008)
6. Dziembowski, S., Pietrzak, K.: Leakage-resilient cryptography. In: FOCS (2008)
7. Capalbo, M., Reingold, O., Vadhan, S., Wigderson, A.: Randomness conductors and constant-degree expansion beyond the degree/2 barrier. In: STOC, pp. 659–668 (2002)
8. Guruswami, V., Umans, C., Vadhan, S.: Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. Journal of the ACM 56, 1–34 (2009)
9. Applebaum, B., Ishai, Y., Kushilevitz, E.: On pseudorandom generators with linear stretch in NC0. Journal of Computational Complexity 17, 38–69 (2008)
10. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography with constant input locality. J. Cryptology 22(4), 429–469 (2009)
11. 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)
12. Applebaum, B.: Pseudorandom generators with long stretch and low locality from random local one-way functions. In: STOC, pp. 805–816 (2012)
13. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
14. Canetti, R., Kushilevitz, E., Ostrovsky, R., Rosén, A.: Randomness versus fault-tolerance. J. Cryptology 13(1), 107–142 (2000)
15. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
16. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (2011)
17. Bennett, C.H., Brassard, G., Robert, J.M.: Privacy amplification by public discussion. SIAM J. Comput. 17(2), 210–229 (1988)
18. Akavia, A., Goldwasser, S., Vaikuntanathan, V.: Simultaneous hardcore bits and cryptography against memory attacks. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 474–495. Springer, Heidelberg (2009)
19. Naor, M., Segev, G.: Public-key cryptosystems resilient to key leakage. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 18–35. Springer, Heidelberg (2009)
20. Goldwasser, S., Rothblum, G.N.: How to compute in the presence of leakage. In: FOCS, pp. 31–40 (2012)
21. Rivest, R.L.: All-or-nothing encryption and the package transform. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 210–218. Springer, Heidelberg (1997)
22. Canetti, R., Dodis, Y., Halevi, S., Kushilevitz, E., Sahai, A.: Exposure-resilient functions and all-or-nothing transforms. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 453–469. Springer, Heidelberg (2000)
23. Halevi, S., Lin, H.: After-the-fact leakage in public-key encryption. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 107–124. Springer, Heidelberg (2011)
24. Garg, S., Jain, A., Sahai, A.: Leakage-resilient zero knowledge. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 297–315. Springer, Heidelberg (2011)

25. Bitansky, N., Canetti, R., Halevi, S.: Leakage-tolerant interactive protocols. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 266–284. Springer, Heidelberg (2012)
26. Chor, B., Goldreich, O., Håstad, J., Friedman, J., Rudich, S., Smolensky, R.: The bit extraction problem of t-resilient functions. In: FOCS, pp. 396–407 (1985)
27. Alekhnovich, M., Hirsch, E.A., Itsyksonz, D.: Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. Journal of Automated Reasoning 35, 51–72 (2005)
28. Ajtai, M.: Secure computation with information leaking to an adversary. In: STOC, pp. 715–724 (2011); Full version on ECCC 18, 82 (2011)

A Robust AFPTAS for Online Bin Packing with Polynomial Migration*,**

Klaus Jansen and Kim-Manuel Klein

Christian-Albrechts-University to Kiel
{kj,kmk}@informatik.uni-kiel.de

Abstract. In this paper we develop general LP and ILP techniques to find an approximate solution with improved objective value close to an existing solution. The task of improving an approximate solution is closely related to a classical theorem of Cook et al. [1] in the sensitivity analysis for LPs and ILPs. This result is often applied in designing robust algorithms for online problems. We apply our new techniques to the online bin packing problem, where it is allowed to reassign a certain number of items, measured by the migration factor. The migration factor is defined by the total size of reassigned items divided by the size of the arriving item. We obtain a robust asymptotic fully polynomial time approximation scheme (AFPTAS) for the online bin packing problem with migration factor bounded by a polynomial in $\frac{1}{\epsilon}$. This answers an open question stated by Epstein and Levin [2] in the affirmative. As a byproduct we prove an approximate variant of the sensitivity theorem by Cook at el. [1] for linear programs.

1 Introduction

The idea behind robust algorithms is to find solutions of an optimization problem that are not only good for a single instance, but also if the instance may change in certain ways. Instances change for example due to uncertainty or when new data arrive. With changing parameters and data, we have the effort to keep as much parts of the existing solution as possible, since modifying a solution is often connected with costs or may even be impossible in practice. Achieving robustness especially for linear programming (LP) and integer linear programming (ILP) is thus a big concern and a very interesting research area. Looking at worst case scenarios, how much do we have to modify a solution if the LP/ILP is changing? There is a result of Cook et al. [1] giving an upper bound for ILPs when changing the right hand side of the ILP. Many algorithms in the theory of robustness are based on this theorem.

As a concrete application we consider the classical online bin packing problem, where items arrive over time and our objective is to assign these items into as

* Supported by DFG Project, Entwicklung und Analyse von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme, Ja 612/14-1.

** Full paper: http://arxiv.org/abs/1302.4213

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 589–600, 2013.

few bins as possible. The notion of robustness allows to repack a certain number of already packed items when a new item arrives. On the one hand we want to guarantee that we use only a certain number of additional bins away from the minimum solution and on the other hand, when a new item arrives, we want to repack as few items as possible. In the case of offline bin packing it is known that unless $\mathcal{P} = \mathcal{NP}$ there is no polynomial time approximation algorithm for offline bin packing that produces a solution better than $\frac{3}{2}OPT$, where OPT is the minimum number of bins needed. For this reason, the most common way to deal with the inapproximability problem is the introduction of the asymptotic approximation ratio. The *asymptotic approximation ratio* for an algorithm A is defined to be $\lim_{x\to\infty} \sup\{\frac{A(I)}{OPT(I)} \mid OPT(I) = x\}$. This leads to the notion of *asymptotic polynomial time approximation schemes (APTAS)*. Given an instance of size n and a parameter $\epsilon \in (0, 1]$, an APTAS has a running time of $poly(n)^{f(\frac{1}{\epsilon})}$ and asymptotic approximation ratio $1 + \epsilon$, where f is an arbitrary function. An APTAS is called an *asymptotic fully polynomial time approximation scheme (AFPTAS)* if its running time is polynomial in n and $\frac{1}{\epsilon}$. The first APTAS for offline bin packing was developed by Fernandez de la Vega & Lueker [3], and Karmakar & Karp improved this result by giving an AFPTAS [4] (see survey on bin packing [5]). Since the introduction by Ullman of the classical online bin packing problem [6], there has been plenty of research (see survey [7]). The best known algorithm has an asymptotic competitive ratio of 1.58889 [8] compared to the optimum in the offline case, while the best known lower bound is 1.54037 [9]. Due to the relatively high lower bound of the classical online bin packing problem, there has been effort to extend the model with the purpose to obtain an improved competitive ratio. The model we follow is the notion of robustness. Introduced by Sanders et al. [10] it allows repacking of arbitrary items while the number of items that are being repacked is limited. To give a measure on how many items are allowed to be repacked Sanders et al. [10] defined the *migration factor*. It is defined by the complete size of all moved items divided by the size of the arriving one. An (A)PTAS is called *robust* if its migration factor is of the size $f(\frac{1}{\epsilon})$, where f is an arbitrary function that only depends on $\frac{1}{\epsilon}$. Since the promising introduction of robustness, several robust algorithms have been developed. Sanders et al. [10] found a robust PTAS for the online scheduling problem on identical machines, where the goal is to minimize the makespan. The robust PTAS has constant but exponential migration factor $2^{\mathcal{O}(\frac{1}{\epsilon}\log^2\frac{1}{\epsilon})}$. In case of bin packing Epstein and Levin [2] developed a robust APTAS for the classical bin packing problem with migration factor $2^{\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})}$ and running time double exponential in $\frac{1}{\epsilon}$. In addition they proved that there is no optimal online algorithm with a constant migration factor. Furthermore, Epstein and Levin [11] showed that the robust APTAS for bin packing can be generalized to packing d-dimensional cubes into a minimum number of unit cubes. Recently Epstein and Levin [12] also designed a robust algorithm for preemptive online scheduling of jobs on identical machines, where the corresponding offline problem is polynomial solvable. They presented an algorithm with migration factor $1-\frac{1}{m}$ that computes an optimal solution whenever a new item arrives. Skutella and

Verschae [13] studied the problem of maximizing the minimum load given n jobs and m machines. They proved that there is no robust PTAS for this machine covering problem. On the positive side, they gave a robust PTAS for the machine covering problem in the case that migrations can be reserved for a later timestep. The algorithm has an amortized migration factor of $2^{\mathcal{O}(\frac{1}{\epsilon}\log^2\frac{1}{\epsilon})}$.

Our Results

An online algorithm is called *fully robust* if its migration factor is bounded by $p(\frac{1}{\epsilon})$, where p is a polynomial in $\frac{1}{\epsilon}$. The purpose of this paper is to give methods to develop fully robust algorithms. In Section 2 we develop a theorem for a given linear program (LP) $\min\{\|x\|_1 \,|Ax \geq b, x \geq 0\}$. Given an approximate solution x' with value $(1+\delta)LIN$ (where LIN is the minimum objective value of the LP) and a parameter $\alpha \in (0, \delta LIN]$, we prove the existence of an improved solution x'' with value $(1+\delta)LIN - \alpha$ and distance $\|x'' - x'\|_1 \leq \alpha(2/\delta + 2)$. In addition, for a given fractional solution x' and corresponding integral solution y', the existence of an improved integral solution y'' with $\|y'' - y'\|_1 = \mathcal{O}(\frac{\alpha+m}{\delta})$ is shown (m is the number of rows of A). Since both results are constructive, we propose also algorithms to compute such improved solutions. Previous robust online algorithms require an optimum solutions of the corresponding ILP and use a sensitivity theorem by Cook et al.[1]. This results in an exponential migration factor in $\frac{1}{\epsilon}$ ([2, 11, 13, 10]). In contrast to this we consider approximate solutions of the corresponding LP relaxations and are able to use the techniques above to improve the fractional and integral solutions. Furthermore we also prove an approximate version of a sensitivity theorem for LPs with modified right hand side b and b'. During the online algorithm the number of non-zero variables increases from step to step and would result in a large additive term. To avoid this we present algorithms in Section 2.2 to control the number of non-zero variables of the LP and ILP solutions. We can bound the number of non-zero variables and the additive term by $\mathcal{O}(\epsilon LIN) + \mathcal{O}(\frac{1}{\epsilon^2})$. In Section 3 we present the fully robust AFPTAS for the robust bin packing problem. We use a modified version of the clever rounding techniques of Epstein and Levin [2]. This rounding technique is used to round the incoming items dynamically and control the number of item sizes. One difficulty is that we use approximate solutions of the LP. During the online algorithm items are rounded to different values and are shifted across different rounding groups. We show how to embed the rounded instance into another rounded instance that fulfills several invariants. By combining the dynanic rounding and the algorithm to get improved solutions of the LP and ILP, we are able to obtain a fully robust AFPTAS for the online bin packing problem. The algorithm has a migration factor of $\mathcal{O}(1/\epsilon^4)$ and running time polynomial in $\frac{1}{\epsilon}$ and t, where t is the number of arrived items. This resolves an open question of Epstein and Levin [2]. We believe that our techniques can be used for other online problems like strip packing, scheduling moldable tasks, resource constrained scheduling and multi-commodity flow problems to obtain online algorithms with low migration factors. Proofs omitted due to space constraints can be found in the full version of the paper. For a detailed view on this paper we recommend the full paper version [14].

2 Robustness of Approximate LPs and ILPs

We consider a matrix $A \in \mathbb{R}_{\geq 0}^{m \times n}$, a vector $b \in \mathbb{R}_{\geq 0}^{m}$ and a cost vector $c \in \mathbb{R}_{\geq 0}^{n}$. The goal in a *linear program* (LP) is to find a $x \geq 0$ with $Ax \geq b$ such that the *objective value* $c^T x$ is minimal. We say x^{OPT} is an optimal solution if $c^T x^{OPT} = \min\left\{c^T x | Ax \geq b, x \geq 0\right\}$ and we define $LIN = c^T x^{OPT}$. In general we suppose that the objective function of a solution is positive and hence $LIN > 0$. We say x' is an approximate solution with approximation ratio $(1+\delta)$ for some $\delta \in (0,1]$ if $\|x'\|_1 \leq (1+\delta)LIN$. We will present Theorem 1 with general objective value c, but for the remaining part of the paper we will assume that $c^T = (1,1,\ldots,1)$ and therefore $c^T x^{OPT} = \left\|x^{OPT}\right\|_1 = LIN$. The following theorem is central. Given an approximate solution x', we want to improve its approximation by some constant. But to achieve robustness we have to maintain most parts of x'. We show that by changing x' by size of $\mathcal{O}(\frac{\alpha}{\delta})$, we can improve the approximation by a constant α.

Theorem 1. *Consider the LP* $\min\left\{c^T x | Ax \geq b, x \geq 0\right\}$ *and an approximate solution* x' *with* $c^T x' = (1+\delta)LIN$ *for some* $\delta > 0$. *For every positive* $\alpha \leq \delta LIN$ *there exists a solution* x'' *with objective value of at most* $c^T x'' \leq (1+\delta)LIN - \alpha$ *and distance* $\|x' - x''\|_1 \leq \alpha(1/\delta+1)\frac{\|x'\|_1 + \|x^{OPT}\|_1}{c^T x'}$. *If* $c^T = (1,1,\ldots,1)$ *then* $\|x' - x''\|_1 \leq 2\alpha(1/\delta+1)$.

The proof basically relies on a convex combination between x' and an optimal solution (see full paper). From here on we suppose that $c^T = (1,1,\ldots,1)$.

Of course, one major application of Theorem 1 is to improve the approximation. But we can also apply Theorem 1 to obtain an approximate variant of the theorem of Cook et al. [1] for the sensitivity analysis of an LP (see full paper).

2.1 Algorithmic Use

Let x' be an approximate solution of the LP with $\|x'\|_1 \leq (1+\delta)LIN$. In Theorem 1, we have proven the existence of a solution x'' near x' with $\|x''\|_1 \leq (1+\delta)LIN - \alpha$. We are looking now for algorithmic ways to calculate this improved solution x''. The following algorithm computes an improved solution x'' that is near to x'.

Algorithm 1

1. *Set* $x^{var} := \frac{\alpha(1/\delta+1)}{\|x'\|} x'$, $x^{fix} := x' - x^{var}$ *and* $b^{var} := b - A(x^{fix})$
2. *Solve the LP* $\hat{x} = \min\{\|x\|_1 | Ax \geq b^{var}, x \geq 0\}$
3. *Generate a new solution* $x'' = x^{fix} + \hat{x}$

If $\hat{x}$ is a basic feasible solution, compared to x', our new solution x'' has up to m additional non-zero components. See full paper for the proof of the following theorem.

Theorem 2. *Given solution* x' *with* $\|x'\|_1 \leq (1+\delta)LIN$ *and* $\|x'\|_1 \geq \alpha(1/\delta+1)$. *Algorithm 1 returns a feasible solution* x'' *with* $\|x''\|_1 \leq (1+\delta)LIN - \alpha$ *and the distance between* x' *and* x'' *is* $\|x'' - x'\|_1 \leq 2\alpha(1/\delta+1)$.

2.2 Integer Programming

In this section we discuss how we can apply results from the previous sections to integer programming. Consider a fractional solution x' of the LP and a corresponding integral solution y'. By rounding each component x'_i up to the next integer value, it is easy to get a feasible integer solution y' with an additional additive term $\|y'\|_1 \leq \|x'\|_1 + C$, where C is the number of non-zero components. We can apply any of the previous algorithms to x' to get an improved solution x''. But our actual goal is to find a corresponding integer solution y'' with improved objective value $\|y''\|_1 \leq (1+\delta)LIN + C - \alpha$ such that the distance between y'' and y' is small. In the following we present an algorithm that computes a suitable y'' with improved objective value and small distance between y'' and y'. We show that the distance between y'' and y' is bounded by $\mathcal{O}(\frac{m+\alpha}{\delta})$. Note that the straight forward approach to simply round up each component x''_i leads to a distance between y'' and y' that depends on C and hence (depending on the LP) is too high. The running time of the presented algorithm depends on the number of non-zero components and the time to compute an optimal solution of an LP.

Let x' be an approximate solution of the LP $\min\{\|x\|_1 \mid Ax \geq b, x \geq 0\}$ with $\|x'\|_1 \leq (1+\delta)LIN$ and $\|x'\|_1 \geq \alpha(1/\delta+1)$. Furthermore let y' be an approximate integer solution of the LP with $\|y'\|_1 \leq (1+2\delta)LIN$ and $\|y'\|_1 \geq (m+1)(1/\delta+2)$ and $y'_i \geq x'_i$ for $i = 1, \ldots, n$. In addition we suppose that both x' and y' have exactly $K \leq \delta LIN$ non-zero components. Our goal is now to compute a fractional solution x'' and and integer solution y'' having improved approximation properties and still $\leq \delta LIN$ non-zero components. For a vector $z \in \mathbb{R}^n_{\geq 0}$, let $V(z)$ be the set of all integral vectors $v = (v_1, \ldots, v_n)^T$ such that $0 \leq v_i \leq z_i$. Furthermore we denote with $a_1, \ldots, a_K$ the indices of the non-zero components y'_{a_j} such that $y'_{a_1} \leq \ldots \leq y'_{a_K}$ are sorted in non-decreasing order.

Algorithm 2

1. *Choose ℓ maximally such that the sum of smallest ℓ components $1, \ldots, \ell$ are $\sum_{1 \leq i \leq \ell} y'_{a_i} \leq (m+1)(1/\delta+2)$*
2. *Set* $x_i^{var} = \begin{cases} x'_i & \text{if } i = a_j \text{ for } j \leq \ell \\ \frac{\alpha(1/\delta+1)}{\|x'\|} x'_i & \text{else} \end{cases}$
   and $\bar{y}_i = \begin{cases} 0 & \text{if } i = a_j \text{ for } j \leq \ell \\ y'_i & \text{else} \end{cases}$
3. *Set $x^{fix} = x' - x^{var}$, $b^{var} = b - A(x^{fix})$ and compute an optimal solution $\hat{x}$ of* $\min\{\|x\|_1 \,|Ax \geq b^{var}, x \geq 0\}$
4. *Set $x'' = x^{fix} + \hat{x}$*
5. *For each $1 \leq i \leq n$ set $\hat{y}_i = \max\{\lceil x''_i \rceil, \bar{y}_i\}$*
6. *If possible choose $d \in V(\hat{y} - x'')$ such that $\|d\|_1 = \alpha(1/\delta+1)$ otherwise choose $d \in V(\hat{y} - x'')$ such that $\|d\|_1 < \alpha(1/\delta+1)$ is maximal.*
7. *Return $y'' = \hat{y} - d$*

Theorem 3. *Let x' be a solution of the LP with $\|x'\|_1 \leq (1+\delta)LIN$ and $\|x'\|_1 \geq \alpha(1/\delta+1)$. Let y' be an integral solution of the LP with $\|y'\|_1 \leq (1+2\delta)LIN$ and $\|y'\|_1 \geq (m+1)(1/\delta+2)$. Solutions x' and y' have both exactly K non-zero components and for each component we have $x'_i \leq y'_i$. Then Algorithm 2 returns a fractional solution x'' with $\|x''\|_1 \leq (1+\delta)LIN - \alpha$ and an integral solution y'' with $\|y''\|_1 \leq (1+2\delta)LIN - \alpha$. Both x'' and y'' have the same number of non-zero components with $x''_i \leq y''_i$ and the number of non-zero components is bounded by δLIN. The distance between y'' and y' is bounded by $\|y''-y'\|_1 = \mathcal{O}(\frac{m+\alpha}{\delta})$.*

Proof. **Feasibility**: Feasibility and approximation for the fractional solution x'' follow easily from correctness of Algorithm 1 and the fact that removing additional components $x'_{a_1}, \ldots, x'_{a_\ell}$ and reassigning them optimally does not worsen the approximation. Each integral component $\hat{y}_i$ is by definition (step 5) greater or equal than x''_i. By choice of d step 6 and 7 retain this property for y'' and imply thus feasibility for y''.

Distance between y'' and y': The only steps where components of y' are changed are step 2, 5 and 7. In step 2 we change y' to obtain $\bar{y}$, in step 5 we change $\bar{y}$ to obtain $\hat{y}$ and in step 7 we change $\hat{y}$ to obtain y''. Summing up the change in each step leads therefore to the maximum possible distance between y'' and y'. In step 2 of the algorithm ℓ components of y' are set to zero to obtain $\bar{y}$, which by the definition of ℓ results in a change of at most $(m+1)(1/\delta+2)$. We define L by $L = \sum_{1 \leq i \leq \ell} y'_{a_i}$ with $0 \leq L \leq (m+1)(1/\delta+2)$. In step 5, the only components $\bar{y}_i$ being changed are the ones where x''_i is larger than $\bar{y}_i$. So the change in step 5 is bounded by $\sum_{x''_i > \bar{y}_i} (\lceil x''_i \rceil - \bar{y}_i) = \sum_{x''_i > \bar{y}_i} (\lceil x_i^{fix} + \hat{x}_i \rceil - \bar{y}_i) \leq \sum_{x''_i > \bar{y}_i} (\lceil x_i^{fix} \rceil - \bar{y}_i + \lceil \hat{x}_i \rceil) \leq \sum_{x''_i > \bar{y}_i} \lceil \hat{x}_i \rceil$ by knowing that $\lceil x_i^{fix} \rceil - \bar{y}_i \leq 0$ since $x_i^{fix} = \bar{y}_i = 0$ if $i = a_j$ for a $j \leq \ell$ or $\lceil x_i^{fix} \rceil < \lceil x'_i \rceil \leq y'_i$. Furthermore we can bound $\sum_{x''_i > \bar{y}_i} \lceil \hat{x}_i \rceil \leq \|\hat{x}\|_1 + m \leq \|x^{var}\|_1 + m$ since $\hat{x}$ is a basic feasible solution and $\|x^{var}\|_1$ can be bounded by $L + \alpha(1/\delta+1)$ (i.e. we get L for the size of components $x'_{a_1}, \ldots, x'_{a_K}$ plus $\sum_{i>\ell} \frac{\alpha(1/\delta+1)}{\|x'\|_1} x'_{a_i} \leq \alpha(1/\delta+1)$ for the remaining ones). Therefore we have $\|\hat{y} - \bar{y}\|_1 \leq L + \alpha(1/\delta+1) + m$. In step 7, $\|y'' - \hat{y}\|_1 = \|d\|_1 \leq \alpha(1/\delta+2) + m$. In sum this makes a total change of at most $(m+1)(1/\delta+2) + L + \alpha(1/\delta+1) + m + \alpha(1/\delta+2) + m \leq 2(m+1)(1/\delta+2) + 2m + \alpha(2/\delta+3) = \mathcal{O}(\frac{m+\alpha}{\delta})$.

Number of components: The property that x' and y' have the same number of non-zero components together with the property that $y'_i \geq x'_i$ implies that $x'_i > 0$ whenever $y'_i > 0$. This property holds also for x^{fix} and $\bar{y}$ since a component $\bar{y}_i$ is set to zero if and only if $x_i^{fix} = 0$. Notice that $y'' = \hat{y} - d \geq x''$. Suppose by contradiction that there is a component i with $x''_i = 0$ and $y''_i > 0$, then $\hat{y}_i = y''_i + d_i > 0$ and by definition of $\hat{y}$ we obtain $\bar{y}_i > 0$. In this case we have $x_i^{fix} > 0$, which gives a contradiction to $x''_i = 0 = x_i^{fix} + \hat{x}_i > 0$. Using the property that x'' and y'' have the same number of non-zero components, it is sufficient to prove that the number of non-zero components of x'' is limited by δLIN. Our new solution x'' is composed of x^{fix} and $\hat{x}$. Solution x^{fix} has $K - \ell$ non-zero components, since in step 2 we set ℓ components of x^{fix} to zero. Being a basic feasible solution, $\hat{x}$ has at most m non-zero components and

hence x'' has at most $K + m - \ell$ non-zero components. If $\ell \geq m$, then x'' has $\leq K \leq \delta LIN$ non-zero components. So let $\ell < m$: The total number of non-zero components after step 4 is $(K + m - \ell)$. We now prove that this number is bounded by δLIN. Parameter ℓ is chosen to be maximal, therefore $\sum_{i \leq l+1} y'_{a_i} \geq (m+1)(1/\delta + 2)$. Hence, the average size of components $y'_{a_1}, \ldots, y'_{a_{\ell+1}}$ is greater than $\frac{(m+1)(1/\delta+2)}{\ell+1} \overset{\ell+1 \leq m}{\geq} \frac{(m+1)(1/\delta+2)}{m} > 1/\delta + 2$. Since the components are sorted in non-decreasing order, every component y'_i with $i \geq \ell + 1$ has size $> 1/\delta + 2$. Summing over all non-zero components of y' yields the following inequality: $\|y'\|_1 = \sum_{i=\ell+2}^{K} y'_{a_i} + y'_{a_{\ell+1}} + L \geq (K - \ell - 1)(1/\delta + 2) + y'_{a_{\ell+1}} + L \geq (K - \ell - 1)(1/\delta + 2) + (m + 1)(1/\delta + 2) = (K - \ell + m)(1/\delta + 2)$. Using that $\|y'\|_1 \leq (1 + 2\delta)LIN$ yields $(1 + 2\delta)LIN \geq (K - \ell + m)(1/\delta + 2)$. Dividing both sides by $(1/\delta + 2)$ gives $(K - \ell + m) \leq \delta LIN$. This shows that the number of non-zero components of x'' and y'' is at most δLIN.

Approximation: Case1: $\|d\|_1 = \alpha(1/\delta + 2) + m$
The following inequalities $\|\hat{y}\|_1 \leq \|\bar{y}\|_1 + L + \alpha(1/\delta + 2) + m = \|y'\|_1 + \alpha(1/\delta + 1) + m$ and $\|y'\|_1 \leq (1+2\delta)LIN$ together yield the aimed approximation $\|y''\|_1 = \|\hat{y}\|_1 - \|d\|_1 = \|\hat{y}\|_1 - \alpha(1/\delta + 2) - m \leq (1 + 2\delta)LIN - \alpha$.
Case2: $\|d\|_1 < \alpha(1/\delta + 2) + m$
Since d is chosen maximally, $y''_i - x''_i < 1$ for every components $i = 1, \ldots, n$. Since $\|x''\|_1 \leq (1 + \delta)LIN - \alpha$ and y'' has at most δLIN non-zero components $\|y''\|_1$ is bounded by $(1 + \delta)LIN - \alpha + \delta LIN = (1 + 2\delta)LIN - \alpha$. □

In case of bin packing the LP solution $\hat{x}$ can not be computed efficiently. Therefore we adapt the algorithm further. The final algorithm called Algorithm B (see full paper), which we use later in the bin packing algorithm allows the use of an approximate LP solution. An approximate fractional solution for bin packing can then be solved efficiently using max-min resource sharing [15].

3 AFPTAS for Robust Bin Packing

The goal of this section is to give a fully robust AFPTAS for the bin packing problem using the methods developed in the previous section. For that purpose we show at first the common way how one can formulate a rounded instance of bin packing as an ILP. In Section 3.2 we present abstract properties of a rounding that need to be fulfilled to obtain a suitable rounding and in Section 3.3 we present the used dynamic rounding algorithm. The crucial part however is the analysis of the dynamic rounding in combination with ILP techniques. Since the ILP and its optimal value are in constant change due to the dynamic rounding, it is difficult to to give a bound for the approximation. Based on the abstract properties we therefore develop techniques how to view and analyze the problem as a whole.

The *online bin packing problem* is defined as follows: Let $I_t = \{i_1, \ldots i_t\}$ be an instance with t items at time step $t \in \mathbb{N}$ and let $s : I_t \to (0, 1]$ be a mapping that defines the sizes of the items. Our objective is to find a function $B_t : \{i_1, \ldots, i_t\} \to \mathbb{N}^+$, such that $\sum_{i:B_t(i)=j} s(i) \leq 1$ for all j and minimal

$\max_i \{B_t(i)\}$ (i.e. B_t describes a packing of the items into a minimum number of bins). We allow to move few items when creating a new solution B_{t+1} for instance $I_{t+1} = I_t \cup \{i_{t+1}\}$. Sanders et al. [10] and also Epstein and Levin [2] defined the *migration factor* to give a measure for the amount of repacking. The migration factor is defined as the total size of all items that are moved between the solutions divided by the size of the arriving item. Formally the migration factor of two packings B_t and B_{t+1} is defined by $\sum_{j \leq t: B_t(i_j) \neq B_{t+1}(i_j)} s(i_j)/s(i_{t+1})$.

3.1 LP-Formulation

Let I be an instance of bin packing with m different item sizes $s_1, \ldots, s_m$. Suppose that for each item $i_k \in I$ there is a size s_j with $s(i_k) = s_j$. A configuration C_i is a multiset of sizes $\{a(C_i, 1) : s_1, a(C_i, 2) : s_2, \ldots a(C_i, m) : s_m\}$ with $\sum_{1 \leq j \leq m} a(C_i, j)s_j \leq 1$, where $a(C_i, j)$ denotes how often size s_j appears in configuration C_i. We denote by C the set of all configurations. Let $|C| = n$. We consider the following LP relaxation of the bin packing problem:

$$\begin{aligned} &\min \|x\|_1 \\ &\sum_{C_i \in C} x_i a(C_i, j) \geq b_j \qquad \forall 1 \leq j \leq m \\ &x_i \geq 0 \qquad \forall 1 \leq j \leq n \end{aligned}$$

Component b_j states the number of items i in I with $s(i) = s_j$ for $j = 1, \ldots, m$. This LP-formulation was first described by Eisemann [16]. Suppose that each size s_j is larger or equal to $\epsilon/2$ for some $\epsilon \in (0, 1/2]$. Since the number of different item sizes is m, the number of feasible packings for a bin is bounded by $|C| = n \leq (\frac{2}{\epsilon} + 1)^m$. Obviously an optimal integral solution of the LP gives a solution to our bin packing problem. We denote by $OPT(I)$ the value of an optimal solution. An optimal fractional solution is a lower bound for the optimal value. We denote the optimal fractional solution by $LIN(I)$.

3.2 Rounding

We use a rounding technique based on the offline APTAS by Fernandez de La Vega & Lueker [3]. As we plan to modify the rounding through the dynamic rounding algorithm we give a more abstract approach on how we can round the items to obtain an approximate packing. At first we divide the set of items into *small* ones and *large* ones. An item i is called *small* if $s(i) < \epsilon/2$, otherwise it is called *large*. Instance I is partitioned accordingly into the large items I_L and the small items I_S. We treat small items and large items differently. Small items can be packed using a greedy algorithm and large items need to be rounded using a rounding function. We define a *rounding function* as a function $R : I_L \mapsto \mathbb{N}$ which maps each large item i to a *group* j. By R^j we denote the set of items being mapped to the same group j, i.e. $R^j = \{i \in I_L \mid R(i) = j\}$. By λ_j^R we denote an item i with $s(i) = \max\{s(i_k) \mid i_k \in R^j\}$. Given an instance I and a

rounding function R, we define the rounded instance I^R by rounding the size of every large item $i \in R^j$ for $j \geq 1$ up to the size $s(\lambda_j^R)$ of the largest item in its group. Items in R^0 are excluded from instance I^R. We write $s_R(i)$ for the rounded size of item i in I^R. Depending on constants c and d, we define the following properties for a rounding function R.

(A) $\max\{R(i) \mid i \in I_L\} = c/\epsilon^2$ for a constant $c \in \mathbb{R}^+$
(B) $|R^i| = |R^j|$ for all $i, j \geq 2$
(C) $|R^0| = d|R^1|$ for a constant $d \in \mathbb{R}^+$ with $d \geq 1$
(D) $s(i) \leq s(j) \Leftrightarrow R(i) \geq R(j)$

Any rounding function fulfilling property (A) has at most $\Theta(1/\epsilon^2)$ different item sizes and hence instance I^R can now be solved approximately using the LP relaxation. The resulting LP relaxation has $\Theta(1/\epsilon^2)$ rows and can be solved approximatley with accuracy $(1+\delta)$ using the max-min resource sharing [15] in polynomial time. Based on the fractional solution we obtain an integral solution y of the LP with $\|y\|_1 \leq (1+\delta)LIN(I^R) + C$ for some additive term $C \geq 0$. We say a packing B *corresponds* to a rounding R and solution y if items in $R^1, \ldots, R^m$ are packed by B according to the integral solution y of the LP. The LP is defined by instance I^R. Items in R^0 are each packed in separate bins.

Lemma 4. *Given instance I with items greater than $\epsilon/2$ and a rounding function R fulfilling properties (A) to (D), then $OPT(I^R) \leq OPT(I)$ and $|R^0| \leq \frac{2d}{c}\epsilon OPT(I)$. Let y be an integral solution of the LP for instance I^R with $\|y\|_1 \leq (1+\delta)LIN(I^R) + C$ for some value $C \geq 0$, let B be a packing of I which corresponds to R and y and let $\epsilon' = \frac{2d}{c}\epsilon$. Then*

$$\max_i \{B_t(i)\} = \|y\|_1 + |R^0| \leq (1 + \epsilon' + \delta)OPT(I) + C.$$

Given instance $I = \{i_1, \ldots, i_t\}$, we define m by $m = \lceil 1/\epsilon^2 \rceil$ if $\lceil 1/\epsilon^2 \rceil$ is even and otherwise $m = \lceil 1/\epsilon^2 \rceil + 1$. By definition m is always even. For every instance I we find a rounding function R with rounding groups $R^0, R^1, \ldots R^m$ which fulfills properties (A)-(D) such that $|R^0| < 2|R^1|$ and $|R^0| \geq |R^1|$. Using this rounding technique in combination with an approximate LP solver leads to an AFPTAS for the offline bin packing problem (see [3] or full paper).

3.3 Online Bin Packing

Let us consider the case where items arrive online. As new items arrive we are allowed to repack several items but we intend to keep the migration factor as small as possible. We present operations that modify the current rounding R_t and packing B_t to give a solution for the new instance. The given operations worsen the approximation but by applying the results from the previous section we can maintain an approximation ratio that depends on ϵ. The presented rounding technique is similar to the one used in [2]. In our algorithm we use approximate solutions of ILPs in contrast to the APTAS of Epstein & Levin which solve the ILPs optimally.

Handling with approximate ILPs results in a different analysis of the algorithm because many helpful properties of optimal solution are getting lost.

Note that in an online scenario of bin packing where large and small items arrive online, small items do not need to be considered. We use the same techniques as in [2] to pack small items. As a small item arrives we place it via FirstFit [17]. In this case FirstFit increases the number of bins being used by at most 1 ([3]) and the migration factor is zero as we repack no item. Whenever a new large item arrives several small items might also need to be replaced. Every small item in a bin that is repacked by the algorithm, is replaced via FirstFit. Packing small items with this strategy does not increase the number of bins that need to be repacked as a large item arrives. Later on the migration factor will solely be determined by the number of bins that are being repacked. More precisely, we will prove that the number of bins, that need to be repacked is bounded by $\mathcal{O}(1/\epsilon^3)$. Therefore we assume without loss of generality that every arriving item is large, i.e. has a size $\geq \epsilon/2$ (see also [2]). Our rounding R_t will be constructed by three different *operations*, called the *insertion, creation* and *union* operation. The insertion operation is performed whenever a large item arrives. This operation is followed by a creation or an union operation depending on the phase the algorithm is in. For the exact definition of the operations and how to use them see full paper. The following algorithm is our central algorithm for the robust online bin packing problem. The exact algorithm as well as its analysis are stated in the full paper.

Algorithm 4

- *While the size of all items* $= \mathcal{O}(m/\bar{\delta})$ *use the offline AFPTAS.*
- *Afterwards use operations insert, create or union repetitively to obtain a suitable rounding* R_t *and a packing for each instance* I_t
- *Use Algorithm B before each operation to improve the packing by a constant.*

Operations insert, create and union worsen the approximation ratio. Therefore we use Algorithm B in advance of each operation to maintain a fixed approximation guarantee. It remains to prove that applying Algorithm B to a solution for I^{R_t} impacts the overall approximation $\Delta = \bar{\epsilon} + \bar{\delta} + \bar{\epsilon}\bar{\delta}$ in the same way. We define $C = \Delta OPT(I_t) + m$. See full paper for the proof.

Theorem 5. *Given a rounding function* R_t *and an LP defined for* I^{R_t}. *Let* x *be a fractional solution of the LP with* $\|x\|_1 + |R_t^0| \leq (1+\Delta)OPT(I_t)$ *and* $\|x\|_1 \geq 2\alpha(1/\bar{\delta}+1)$ *and* $\|x\|_1 = (1+\delta')LIN(I^{R_t})$ *for some* $\delta' > 0$. *Let* y *be an integral solution of the LP with* $\|y\|_1 \geq (m+2)(1/\bar{\delta}+2)$ *and corresponding packing* B *such that* $\max_i B_t(i) = \|y\|_1 + |R_t^0| \leq (1+2\Delta)OPT(I_t)+m$. *Suppose* x *and* y *have the same number* $\leq C$ *of non-zero components and for all components* i *we have* $y_i \geq x_i$. *Then using Algorithm B on* x *and* y *returns new solutions* x' *with* $\|x'\|_1 + |R_t^0| \leq (1+\Delta)OPT(I_t) - \alpha$ *and integral solution* y' *with corresponding packing* B_t' *such that*

$$\max_i B_t'(i) \leq (1+2\Delta)OPT(I_t) + m - \alpha.$$

Further, both solutions x' and y' have the same number $\leq C$ of non-zero components and for each component we have $x'_i \leq y'_i$.

Set $\bar{\delta} = \bar{\epsilon}$. Then $\Delta = 2\bar{\epsilon} + \bar{\epsilon}^2 = \mathcal{O}(\epsilon)$. Using the above theorem inductively after each operation leads to the following main theorem.

Theorem 6. *Algorithm 4 is a fully robust AFPTAS for the bin packing problem.*

3.4 Running Time

Storing items that are in the same rounding group in a heap structure, we can perform each operation (insertion, creation and union) in time $\mathcal{O}(\frac{1}{\epsilon^2}\log(\epsilon^2 t))$. Furthermore Algorithm B needs to look through all non-zero components. The number of non-zero components is bounded by $\mathcal{O}(\epsilon OPT) = \mathcal{O}(\epsilon t)$. Main part of the complexity lies in finding an approximate LP solution. Let $M(n)$ be the time to solve a system of n linear equations. The running time of max-min resource sharing is then in our case $\mathcal{O}(M(\frac{1}{\epsilon^2})\frac{1}{\epsilon^4} + \frac{1}{\epsilon^7})$ (see [18]). Therefore the running time of the Algorithm is $\mathcal{O}(M(\frac{1}{\epsilon^2})\frac{1}{\epsilon^4} + \epsilon t + \frac{1}{\epsilon^2}\log(\epsilon^2 t))$.

4 Conclusion

Based on approximate solutions, we developed an analog to a theorem of Cook et al. [1]. Our improvement helps to develop online algorithms with a migration factor that is bounded by a polynomial in $1/\epsilon$, while algorithms based on Cook's theorem usually have exponential migration factors. We therefore applied our techniques to the famous online bin packing problem. This led to the creation of the first fully robust AFPTAS for an NP-hard online optimization problem. The migration factor of our algorithm is of size $\mathcal{O}(\frac{1}{\epsilon^4})$, which is a notable reduction compared to previous robust algorithms. When a new item arrives at time t the algorithm needs running time of $\mathcal{O}(M(\frac{1}{\epsilon^2})\frac{1}{\epsilon^4} + \epsilon t + \frac{1}{\epsilon^2}\log(\epsilon^2 t))$, where $M(n)$ is the time to solve a system of n linear equations. Any improvement to the max-min resource sharing algorithm based on the special structure of bin packing would immediately speed up our online algorithm. We believe that there is room to reduce the running time and the migration factor. Note for example that we give only a very rough bound for the migration factor as the algorithm repacks $\mathcal{O}(\frac{1}{\epsilon^3})$ bins. Repacking these bins in a more carefully way might lead to a smaller migration factor. An open question is the existence of an AFPTAS with a constant migration factor that is independent of ϵ. We mention in closing that the LP/ILP-techniques we present are very general and hence can possibly be used to obtain fully robust algorithms for several other online optimization problems as well (i.e. multi-commodity flow, strip packing, scheduling with malleable/moldable task or scheduling with resource constraints).

References

[1] Cook, W., Gerards, A., Schrijver, A., Tardos, E.: Sensitivity theorems in integer linear programming. Mathematical Programming 34(3), 251–264 (1986)

[2] Epstein, L., Levin, A.: A robust APTAS for the classical bin packing problem. Mathematical Programming 119(1), 33–49 (2009)
[3] Fernandez de la Vega, W., Lueker, G.: Bin packing can be solved within $1 + \epsilon$ in linear time. Combinatorica 1(4), 349–355 (1981)
[4] Karmarkar, N., Karp, R.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: 23rd Annual Symposium on Foundations of Computer Science(FOCS), pp. 312–320. IEEE Computer Society (1982)
[5] Coffman, E., Garey, M., Johnson, D.: Approximation algorithms for bin packing: a survey. In: Hochbaum, D. (ed.) Approximation algorithms for NP-hard problems, pp. 46–93. PWS Publishing Co. (1997)
[6] Ullman, J.: The Performance of a Memory Allocation Algorithm. Technical report. Princeton University (1971)
[7] Csirik, J., Woeginger, G.J.: On-line packing and covering problems. In: Fiat, A. (ed.) Online Algorithms 1996. LNCS, vol. 1442, pp. 147–177. Springer, Heidelberg (1998)
[8] Seiden, S.: On the online bin packing problem. Journal of the ACM 49(5), 640–671 (2002)
[9] Balogh, J., Békési, J., Galambos, G.: New lower bounds for certain classes of bin packing algorithms. In: Jansen, K., Solis-Oba, R. (eds.) WAOA 2010. LNCS, vol. 6534, pp. 25–36. Springer, Heidelberg (2011)
[10] Sanders, P., Sivadasan, N., Skutella, M.: Online scheduling with bounded migration. Mathematics of Operations Research 34(2), 481–498 (2009)
[11] Epstein, L., Levin, A.: Robust approximation schemes for cube packing (2010) (unpublished manuscript)
[12] Epstein, L., Levin, A.: Robust algorithms for preemptive scheduling. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 567–578. Springer, Heidelberg (2011)
[13] Skutella, M., Verschae, J.: A robust PTAS for machine covering and packing. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 36–47. Springer, Heidelberg (2010)
[14] Jansen, K., Klein, K.: A robust AFPTAS for online bin packing with polynomial migration (2012), `http://arxiv.org/abs/1302.4213`
[15] Grigoriadis, M., Khachiyan, L., Porkolab, L., Villavicencio, J.: Approximate max-min resource sharing for structured concave optimization. SIAM Journal on Optimization 11, 1081 (2001)
[16] Eisemann, K.: The trim problem. Management Science 3(3), 279–284 (1957)
[17] Coffman, E., Garey, M., Johnson, D.: Approximation algorithms for bin-packing: An updated survey. Algorithm Design for Computer System Design, 49–106 (1984)
[18] Jansen, K.: Approximation algorithms for min-max and max-min resource sharing problems, and applications. In: Bampis, E., Jansen, K., Kenyon, C. (eds.) Efficient Approximation and Online Algorithms. LNCS, vol. 3484, pp. 156–202. Springer, Heidelberg (2006)

Small Stretch Pairwise Spanners*

Telikepalli Kavitha and Nithin M. Varma

Tata Institute of Fundamental Research, India
{kavitha,nithin}@tcs.tifr.res.in

Abstract. Let $G = (V, E)$ be an undirected unweighted graph on n vertices. A subgraph H of G is called an (α, β) spanner of G if for each $(u, v) \in V \times V$, the u-v distance in H is at most $\alpha \cdot \delta_G(u, v) + \beta$. The following is a natural relaxation of the above problem: we care for only certain distances, these are captured by the set $\mathcal{P} \subseteq V \times V$ and the problem is to construct a sparse subgraph H, called an (α, β) $\mathcal{P}$-spanner, where for every $(u, v) \in \mathcal{P}$, the u-v distance in H is at most $\alpha \cdot \delta_G(u, v) + \beta$.

We show how to construct a (1,2) $\mathcal{P}$-spanner of size $\tilde{O}(n \cdot |\mathcal{P}|^{1/3})$ and a (1,2) $(S \times V)$-spanner of size $\tilde{O}(n \cdot (n|S|)^{1/4})$. A D-spanner is a $\mathcal{P}$-spanner when $\mathcal{P}$ is described implicitly via a distance threshold D as $\mathcal{P} = \{(u, v) : \delta(u, v) \geq D\}$. For a given $D \in \mathbb{Z}^+$, we show how to construct a $(1, 4)$ D-spanner of size $\tilde{O}(n^{3/2}/D^{1/4})$ and for $D \geq 2$, a $(1, 4 \log D)$ D-spanner of size $\tilde{O}(n^{3/2}/\sqrt{D})$.

1 Introduction

Let $G = (V, E)$ be an undirected unweighted graph on n vertices. A subgraph H of G is said to be a spanner with stretch function (α, β) if for every pair of vertices (u, v) we have: $\delta_H(u, v) \leq \alpha \cdot \delta_G(u, v) + \beta$, where $\delta_H(u, v)$ is the distance between u and v in the graph H and $\delta_G(u, v)$ is the distance between u and v in the graph G. The goal in spanner problems is to show the sparsest possible subgraph for a given stretch function. Constructing sparse spanners with small stretch is a fundamental problem in graph algorithms and this problem has been well-studied in weighted and unweighted graphs during the last 25 years [4, 5, 8, 14–16, 18, 20, 22–24, 26, 27].

The following is a natural variant of the spanner problem: we relax the condition that the stretch in the subgraph has to be bounded by (α, β) for *all* pairs of vertices. We care for only certain distances here – this set is given by a subset $\mathcal{P} \subseteq V \times V$ and the problem is to compute a sparse subgraph H such that $\delta_H(u, v) \leq \alpha \cdot \delta_G(u, v) + \beta$ for all $(u, v) \in \mathcal{P}$. For pairs outside $\mathcal{P}$, the stretch in H can be arbitrary. Such a subgraph H is called an (α, β) $\mathcal{P}$-spanner or *pairwise spanner* of G. When $\mathcal{P} = S \times V$, we will refer to the $\mathcal{P}$-spanner as a *sourcewise spanner*.

Pairwise preservers were studied by Coppersmith and Elkin [10] in 2005 where the input is $G = (V, E)$ along with $\mathcal{P} \subseteq V \times V$ and the problem is to construct

* This work was supported by IMPECS (Indo-German Max Planck Center for Computer Science).

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 601–612, 2013.

a sparse subgraph H such that the u-v distance for each $(u,v) \in \mathcal{P}$ is *exactly* preserved, i.e., $\delta_H(u,v) = \delta_G(u,v)$ for all $(u,v) \in \mathcal{P}$. They constructed such a subgraph H of size $O(\min\{n\sqrt{|\mathcal{P}|},\ n+\sqrt{n}|\mathcal{P}|\})$. They posed the problem of computing sparser subgraphs where distances between pairs in $\mathcal{P}$ are *approximately* preserved, namely the problem of computing sparse $\mathcal{P}$-spanners.

Pettie [22] showed a $(1, O(\log n))$ $(S \times V)$-spanner of size $O(n\sqrt{|S|})$. Cygan et al. [13] very recently showed size-stretch trade-off results for $(1, 2k)$ sourcewise spanners and $(1, 4k)$ pairwise spanners, for all integers $k \geq 1$. In particular, their $(1,2)$ $(S \times V)$-spanner has size $O(n^{4/3}|S|^{1/3})$. Also known is a $(1,2)$ $(S \times S)$-spanner of size $O(n\sqrt{|S|})$ (see [13, 22]).

Our Results. We consider the problem of constructing sparse (1,2) $\mathcal{P}$-spanners for $\mathcal{P} = S \times V$, where $S \subseteq V$, and for general $\mathcal{P} \subseteq V \times V$. Such pairwise spanners are relevant in settings where only certain distances are of interest to us and we seek a sparse subgraph that captures these distances *almost* exactly.

Another motivation for this problem is that all the currently known "purely additive" spanners can be constructed via appropriate (1,2) $\mathcal{P}$-spanners. When the stretch function is $(1, \beta)$ for $\beta = O(1)$, the corresponding spanner is said to be purely additive. Purely additive spanners are known for unweighted graphs when $\beta = 2, 4, 6$. These results are as follows: a $(1,2)$ spanner of size $O(n^{3/2})$ [16], a $(1,4)$ spanner of size $O(n^{1.4}\log^{0.2} n)$ [8], and a $(1,6)$ spanner of size $O(n^{4/3})$ [4].

The $(1,2)$ spanner is obviously a $(1,2)$ $\mathcal{P}$-spanner, where $\mathcal{P} = V \times V$. The $(1,6)$ spanner of size $O(n^{4/3})$ can be obtained using the sparse $(1,2)$ $(S \times S)$-spanner of size $O(n\sqrt{|S|})$ for an appropriate $S \subseteq V$. We will see that a $(1,4)$ spanner of size $O(n^{1.4}\log^{0.2} n)$ can be obtained via a $(1,2)$ $(S \times V)$-spanner of size $O(n \cdot (n|S|\log n)^{1/4})$. Note that this is a deterministic construction of a $(1,4)$ spanner of this size, thereby improving upon the construction in [8] which is randomized and which shows an $O(n^{1.4}\log^{0.2} n)$ bound on the expected size of the $(1,4)$ spanner. We show the following results here.

Theorem 1. *For any $S \subseteq V$, there is a polynomial time algorithm to compute a $(1,2)$ $(S \times V)$-spanner of size $O(n \cdot (n|S|\log n)^{1/4})$.*

Theorem 2. *For any $\mathcal{P} \subseteq V \times V$, there is a polynomial time algorithm to compute a $(1,2)$ $\mathcal{P}$-spanner of size $O(n \cdot (|\mathcal{P}|\log n)^{1/3})$.*

We next study a variant of the $\mathcal{P}$-spanner problem where the set $\mathcal{P}$ is not specified explicitly as a part of the input. Instead, the set $\mathcal{P}$ is described implicitly via a *distance threshold* D. Here along with the input graph G, we are given $D \in \mathbb{Z}^+$ and the set $\mathcal{P} = \{(u,v) : \delta_G(u,v) \geq D\}$. The problem of computing sparse *D-preservers*, where the problem is to compute a sparse $\mathcal{P}$-preserver for the set $\mathcal{P} = \{(u,v) : \delta_G(u,v) \geq D\}$, was studied by Bollobás et al. [7] in 2003.

The motivation for studying D-preservers was the following result of Elkin and Peleg [16]: for any $\epsilon > 0$ and $k \in \mathbb{Z}^+$, there exists a value $f(\epsilon, k)$ such that there is a subgraph H of G with $O(n^{1+1/k})$ edges where $\delta_H(u,v) \leq (1+\epsilon) \cdot \delta_G(u,v)$ for all (u,v) with $\delta_G(u,v) \geq f(\epsilon,k)$. This motivated the question studied in [7]

of whether there is a sparse subgraph, where all large distances can be exactly preserved and they showed such a D-preserver of size $O(n^2/D)$. Here we consider the question of whether there is a subgraph sparser than the D-preserver where distances $\geq D$ can be approximated with a small additive stretch. Let an (α, β) D-spanner refer to an (α, β) $\mathcal{P}$-spanner for $\mathcal{P} = \{(u, v) : \delta_G(u, v) \geq D\}$. We show the following results here.

Theorem 3. *For any $D \in \mathbb{Z}^+$, there is a polynomial time algorithm to compute a $(1, 4)$ D-spanner of size $O(n^{3/2} \cdot (\log n/D)^{1/4})$.*

Theorem 4. *For any integer $D \geq 2$, there is a polynomial time algorithm to compute a $(1, 4 \log D)$ D-spanner of size $O(n \cdot \sqrt{n \log n/D})$.*

1.1 Background and Related Work

Graph spanners were introduced by Peleg and Schaffer [20] in 1989. Thereafter, spanners have been very well-studied, and there are several applications involving spanners, including algorithms for approximate shortest paths [1, 9, 15], approximate distance oracles [3, 6, 25], labeling schemes [17, 19], network design [21], and routing [2, 11, 12].

There are several algorithms known for computing multiplicative and additive spanners in weighted and unweighted graphs. It is known that every graph on n vertices admits a $(2k - 1, 0)$-spanner of size $O(n^{1+1/k})$ [5, 18, 23, 24]. For unweighted graphs, Dor et al. [14] (with subsequent work by Elkin and Peleg in [16]) showed the first purely-additive spanner of stretch (1,2) and size $O(n^{3/2})$. Baswana et al. [4] showed the construction a $(1, 6)$ spanner of size $O(n^{4/3})$. Very recently, Chechik [8] showed the construction of a (1,4)-spanner with expected size $O(n^{1.4}\log^{0.2} n)$.

The problem of constructing sparse $\mathcal{P}$-spanners, for a given $\mathcal{P} \subseteq V \times V$, was studied by Cygan et al. [13] who showed the following constructions for any integer $k \geq 1$: a $(1, 4k)$ $\mathcal{P}$-spanner of size $O(n^{1+1/(2k+1)} \cdot (k|\mathcal{P}|)^{k/(4k+2)})$ and a $(1, 2k)$ $(S \times V)$-spanner of size $O(n^{1+1/(2k+1)} \cdot (k|S|)^{k/(2k+1)})$ for any $S \subseteq V$.

Techniques. The algorithms in [4, 13] are based on *clustering* and *path-buying*. The clustering step is common to several spanner algorithms. Section 2 describes this step. The path-buying technique was introduced in the (1,6) spanner algorithm [4]. Here a path p is assigned a *value* and a *cost* – if p's value dominates its cost, then p is "bought", i.e., p is added to the subgraph being constructed.

Clustering along with breadth-first-search trees rooted at all *cluster centers* yields a (1,2) spanner [14, 16]. The algorithm in [8] for (1,4) spanner uses randomization first to pick a small sample of vertices for rooting BFS trees and then to cover the neighborhood of high-degree vertices with high probability; some selected shortest paths are also added and the resulting graph is shown to be a (1,4) spanner with high probability.

All our algorithms combine these three steps: (1) clustering, (2) path-buying, and (3) selecting a few vertices to root BFS trees.

We mark shortest paths as *high-cost* or *low-cost*. High-cost paths turn out to be easy to approximate via BFS trees. The path-buying step assesses each low-cost path p – if adding p to our subgraph improves a sufficient number of relevant distances, then p is added to our subgraph, else we show that our subgraph already contains a good approximation of p. These algorithms are described in Sections 3 and 4.

2 Clustering and BFS Trees

A clustering of a graph $G = (V, E)$ is a partition of the vertex set V into subsets $C_1, \ldots, C_\lambda$ called *clusters* and the set $U = V \setminus \cup_i C_i$ of *unclustered* vertices. Associated with each cluster C_i is a vertex called its *cluster center*, denoted by center(C_i), with the following property:

– In the graph G, center(C_i) is a common neighbor of all $x \in C_i$.

Several clusters can share the same cluster center and center(C_i) $\notin C_i$ for any i. We follow the clustering procedure used in [13]. Given an integer h, where $1 \leq h \leq n$, this procedure efficiently constructs the clustering $\mathcal{C} = \{C_1, \ldots, C_\lambda, U\}$ as follows:

– initially all the vertices are unclustered and $\mathcal{C} = \emptyset$.
– while there exists a $v \in V$ with more than h unclustered neighbors
  • $C =$ an arbitrary subset of h unclustered neighbors of v; center(C) $= v$.
  • all vertices in C are marked clustered; $\mathcal{C} = \mathcal{C} \cup \{C\}$.
– let U denote the set of unclustered vertices. $\mathcal{C} = \mathcal{C} \cup \{U\}$.

Thus each cluster C is a collection of exactly h vertices and there can be at most n/h clusters in $\mathcal{C}$. Associated with $\mathcal{C}$ is a subgraph $G_\mathcal{C}$, whose edge set consists of the following:

(i) all edges (a, b) in G where either a or b (or both) is in U,
(ii) for each cluster C_i, all edges e in G where both endpoints of e are in C_i, and
(iii) for each cluster C_i, the edges $(x, \text{center}(C_i))$ $\forall$ $x \in C_i$.

It is easy to see that the subgraph $G_\mathcal{C}$ has size $O(nh)$. It follows from (i) and (ii) above that any edge not present in $G_\mathcal{C}$ has to be an *inter-cluster* edge, i.e., it is an edge whose endpoints belong to distinct clusters. The following definition of the cost of a path will be used in the rest of the paper.

Definition 1. *For any path p in G, let* $\mathsf{cost}(p)$ *be the number of edges in p that are not present in $G_\mathcal{C}$.*

Lemma 1 (from [13]). *Let ρ be a shortest path in G with* $\mathsf{cost}(\rho) \geq t$*, where $t \in \mathbb{R}^+$. Then there are at least $t/2$ clusters of $\mathcal{C}$ having at least one vertex on ρ.*

For each pair of vertices u and v, we select an arbitrary shortest u-v path in G as *the* shortest u-v path. Let $\mathcal{R} = \{\rho_1, \ldots, \rho_{\binom{n}{2}}\}$ be the set of all $\binom{n}{2}$ pairwise shortest paths in G.

Definition 2. *Call a path ρ high-cost if $\rho \in \mathcal{R}$ and $\mathsf{cost}(\rho) \geq (n \ln n)/h^2$.*

Lemma 2. *There is a polynomial time algorithm to compute cluster indices $i_1, \ldots, i_k$, where k is $O(h)$, so that the set $C_{i_1} \cup \cdots \cup C_{i_k}$ has at least one vertex on every high-cost path.*

Proof. We form a table T_0 whose rows are all the high-cost paths. Let these paths be $\rho'_1, \ldots, \rho'_\ell$. The columns of T_0 are all the clusters $C_1, \ldots, C_\lambda$ in $\mathcal{C}$. For $1 \leq i \leq \ell$ and $1 \leq j \leq \lambda$, we have $T_0[i,j] = 1$ if cluster C_j contains at least one vertex on path ρ'_i, else $T[i,j] = 0$.

It follows from Lemma 1 that there are at least $t/2$ 1's in each row, where $t = (n \ln n)/h^2$. So the total number of 1's in T_0 is at least $\ell \cdot t/2$. Thus there has to be a column in T_0 with at least $\ell \cdot t/(2\lambda)$ 1's. Let i_1 be the index of such a column. Delete all the rows that have a 1 in the i_1-th column from T_0 and call the resulting table T_1.

Let r_1 be the number of rows in T_1. Note that $r_1 \leq \ell(1 - t/(2\lambda))$. The invariant remains that every row in T_1 has at least $t/2$ 1's. Repeat the same step for T_1 to determine a column index i_2 and so on.

After k steps, where $k = \lceil (4\lambda \ln n)/t \rceil$, the number of rows in T_k is at most

$$\ell(1 - t/(2\lambda))^{(4\lambda/t)\ln n} \leq \ell/n^2 < 1,$$

since $\ell \leq \binom{n}{2}$. In other words, T_k has zero rows. Thus the set $C_{i_1} \cup \cdots \cup C_{i_k}$ has at least one vertex on every high-cost path, where $i_1, \ldots, i_k$ are the column indices selected in these k steps. The value $k = \lceil (4\lambda \ln n)/t \rceil$, which is $O(h)$ since $\lambda \leq n/h$ and $t = (n \ln n)/h^2$. □

It follows from Lemma 2 that that every high-cost path ρ contains a neighbor of some vertex in $\{\text{center}(C_{i_1}), \ldots, \text{center}(C_{i_k})\}$. Hence the breadth-first-search tree in G rooted at this cluster center contains a path of length at most $|\rho| + 2$ between the endpoints of ρ.

Define $\mathcal{T}$ as the subgraph of G whose edge set is $T_{i_1} \cup \cdots \cup T_{i_k}$, where for $j = 1, \ldots, k$, T_{i_j} is the BFS tree in G rooted at $\text{center}(C_{i_j})$. We can conclude Corollary 1 from Lemma 2.

Corollary 1. *If (u,v) is a pair of vertices whose shortest path in $\mathcal{R}$ has cost $\geq (n \ln n)/h^2$, then $\mathcal{T}$ has a path of length at most $\delta_G(u,v) + 2$ between u and v.*

3 Algorithms for Sparse (1,2) $\mathcal{P}$-Spanners

In this section we first present our algorithm for a (1,2) $(S \times V)$-spanner and then use this sourcewise spanner to construct a (1,4) all-pairs spanner. We then present our algorithm to compute a (1,2) $\mathcal{P}$-spanner, for general $\mathcal{P}$.

3.1 A (1,2) Sourcewise Spanner

The input here is an undirected unweighted graph $G = (V, E)$ on n vertices and a subset $S \subseteq V$. Our algorithm to find a sparse (1,2) $(S \times V)$-spanner H in G

is presented below. Note that in the path-buying step, when a path p is added to H, it means H becomes $H \cup \{p\}$. Recall that $\mathcal{R} = \{\rho_1, \ldots, \rho_{\binom{n}{2}}\}$ is the set of all pairwise shortest paths in G.

1. *Clustering.* Call the clustering procedure with $h = \lceil (n|S| \ln n)^{1/4} \rceil$. This computes the clustering $\mathcal{C} = \{C_1, \ldots, C_\lambda, U\}$ and the post-clustering subgraph $G_{\mathcal{C}}$. Initialize H to $G_{\mathcal{C}}$.
2. *Path-buying.* For each $p \in \mathcal{R}$ where one endpoint of p (call it s) belongs to S and the other endpoint (call it v) belongs to a cluster in $\mathcal{C}$ do:
   – If $\mathsf{cost}(p) < \sqrt{\frac{n \ln n}{|S|}}$ and $|p| < \delta_H(s, C_i)$, where C_i is v's cluster, then add p to H. [note that $\delta_H(s, C_i) = \min\{\delta_H(s, x) : x \in C_i\}$]
3. *Adding BFS trees.* Select $O(h)$ cluster indices (as given by Lemma 2) so that the union of these clusters contains at least one vertex on each path in $\mathcal{R}$ with cost $\geq (n \ln n)/h^2$.
   – Add the edges of the BFS trees rooted at these $O(h)$ cluster centers to H.
4. Return the subgraph H.

The value $h = \lceil (n|S| \ln n)^{1/4} \rceil$. So a path $p \in \mathcal{R}$ with cost $\geq \sqrt{(n \ln n)/|S|}$ has cost at least $(n \ln n)/h^2$, i.e., it is high-cost (Definition 2). If (u, v) are the endpoints of such a path, then due to the edges added to H in step 3 of our algorithm, we have $\delta_H(u, v) \leq \delta_G(u, v) + 2$ (by Corollary 1).

Call a path $p \in \mathcal{R}$ *low-cost* if $\mathsf{cost}(p) < \sqrt{(n \ln n)/|S|}$. Suppose p is low-cost and its endpoints are s and v, where $s \in S$. Step 2 buys p only if v is clustered and p improves the current distance between s and v's cluster. Lemma 4 shows a (1,2) stretch distance in H for all $(S \times V)$ distances in G. Lemma 3 bounds the size of H. Theorem 1 stated in Section 1 follows from these lemmas.

Lemma 3. *The size of the subgraph H is $O(n \cdot (n|S| \log n)^{1/4})$.*

Proof. The clustering step and the step of adding BFS trees add $O(nh)$ edges to our subgraph H. Since $h = \lceil (n|S| \log n)^{1/4}) \rceil$, this bounds the number of such edges by $O(n \cdot (n|S| \log n)^{1/4})$.

We now to need to bound the number of edges added to H in the path-buying step (step 2). Since every path added in step 2 contributes at most $\sqrt{(n \ln n)/|S|}$ edges that are not present in $G_{\mathcal{C}}$, we can bound the total number of new edges added to H in step 2 by $\sqrt{(n \ln n)/|S|} \cdot$(the number of paths added in step 2).

Each path added in step 2 is between a source vertex and a clustered vertex. We now bound the number of s-C_i paths added in step 2 for a fixed pair (s, C_i), where $s \in S$ and C_i is a cluster in $\mathcal{C}$. Once an s-C_i path $p \in \mathcal{R}$ is added to H in step 2, the s-C_i distance in H becomes $\leq \delta_G(s, C_i) + 2$, since p is a shortest path.[1] Thereafter in step 2, at most 2 s-C_i paths can be bought. Thus in total, at most 3 s-C_i paths can be added to H in step 2 and this is true for each pair

[1] If $\delta_G(s, C_i) = d$, then $\delta_G(s, v) \in \{d, d+1, d+2\}$ $\forall v \in C_i$ due to length ≤ 2 paths between every pair of vertices in C_i.

(s, C_i). So the number of paths added to H in step 2 is at most $3|S|\lambda$, where λ is the number of clusters. So the number of edges added in step 2 is at most

$$\sqrt{\frac{n \ln n}{|S|}} \cdot 3|S| \cdot \frac{n}{(n|S| \ln n)^{1/4}} \quad = \quad 3n^{5/4} \cdot |S|^{1/4} \cdot \ln^{1/4} n.$$

Thus the size of the final subgraph H is $O(n \cdot (n|S| \log n)^{1/4})$. □

Lemma 4. *Let $(s,t) \in S \times V$. Then $\delta_H(s,t) \leq \delta_G(s,t) + 2$, where H is our final subgraph.*

Proof. Let $\rho \in \mathcal{R}$ be the s-t shortest path in G. If ρ contains only unclustered vertices, then the entire path ρ is present in H (since $G_{\mathcal{C}}$ is a subgraph of H).

So let us assume there are some clustered vertices on ρ. Let v be the *last* vertex on ρ that is clustered, where we regard s as the first vertex on ρ. Let ρ' be the subpath of ρ between s and v and let ρ'' be the subpath of ρ between v and t. Since every vertex on ρ after v is unclustered, the entire path ρ'' is present in H.

We will now show that H has a path of length at most $|\rho'| + 2$ between s and v. This will imply that H has a path of length at most $|\rho| + 2$ between s and t. Let $p \in \mathcal{R}$ be the s-v shortest path, so $|p| = |\rho'|$. If p is high-cost, then Corollary 1 proves there is a path of length at most $|p| + 2$ between s and v in one of the BFS trees added to H in step 3.

So let us assume p is low-cost. Our algorithm would have considered p in step 2 since the vertex v is clustered. Let $v \in C_i$, where $C_i \in \mathcal{C}$ and let X be our subgraph when path p is considered in step 2. Since p is low-cost, if $\delta_X(s, C_i) > |p|$, then we would have bought p, in which case $\delta_X(s, v) = |p|$. So let us assume that $\delta_X(s, C_i) \leq |p|$ when we considered p in step 2. So X has a path of length $\leq |p|$ between s and C_i.

Since X has a path of length at most 2 between every pair of vertices in C_i, this means there is a path of length at most $|p| + 2$ between s and v in X. Thus $\delta_H(s, v) \leq |\rho'| + 2$. Hence $\delta_H(s, t) \leq |\rho| + 2$. □

A (1,4) All-Pairs Spanner. As an application of our sparse (1,2) $(S \times V)$-spanner, we now show a simple deterministic construction of a (1,4) all-pairs spanner of size $O(n^{1.4} \log^{0.2} n)$.

The input is $G = (V, E)$. The set S of source vertices gets determined in step 1 of our algorithm, which is described below.

1. Let S be the set of center clusters in the clustering $\mathcal{C}$ computed by the clustering procedure on G with parameter $h = \lceil n^{0.4} \log^{0.2} n \rceil$. Let G_0 be the subgraph of $G_{\mathcal{C}}$ whose edge set consists of the following:
   - all edges with at least one endpoint unclustered in $\mathcal{C}$ and for each cluster $C_i \in \mathcal{C}$, the edges $(x, \text{center}(C_i))$ for all $x \in C_i$.
2. Call our (1,2) $(S \times V)$-spanner algorithm in G. Let H be this graph. Return $H' = G_0 \cup H$.

We can show the following lemma (proof omitted here) which bounds the size of the subgraph H' and shows it to be a $(1, 4)$ all-pairs spanner.

Lemma 5. *The size of H' is $O(n^{1.4} \log^{0.2} n)$. For any pair $(u, v) \in V \times V$, we have $\delta_{H'}(u, v) \leq \delta_G(u, v) + 4$.*

3.2 A (1,2) Pairwise Spanner

In this section we consider the problem of computing a sparse (1,2) $\mathcal{P}$-spanner in $G = (V, E)$, for a given subset $\mathcal{P} \subseteq V \times V$. The path-buying step is very simple here: for each $(u, v) \in \mathcal{P}$, we buy the shortest u-v path in G if its cost is "low", else the BFS trees added to H will take care of the (u, v) distance in H. Our algorithm is presented below.

1. *Clustering.* Call the clustering procedure with parameter $h = \lceil (|\mathcal{P}| \ln n)^{1/3} \rceil$ to compute the clustering $\mathcal{C}$ and the subgraph $G_{\mathcal{C}}$. Initialize H to $G_{\mathcal{C}}$.
2. *Path-buying.* For each $(u, v) \in \mathcal{P}$ do: if $\mathsf{cost}(\rho) < n \cdot \frac{(\ln n)^{1/3}}{|\mathcal{P}|^{2/3}}$, then add ρ to H, where $\rho \in \mathcal{R}$ is the shortest u-v path in G.
3. *Adding BFS trees.* Select $O(h)$ cluster indices as given by Lemma 2 so that the union of these clusters contains at least one vertex on each high-cost path (Definition 2). Add the BFS trees rooted at these $O(h)$ cluster centers to H.
4. Return this subgraph H.

It is easy to prove the following lemma which shows the correctness of the above algorithm. We can now conclude Theorem 2 stated in Section 1.

Lemma 6. *The final H is a (1,2) $\mathcal{P}$-spanner and its size is $O(n|\mathcal{P}|^{1/3} \log^{1/3} n)$.*

4 Purely Additive D-Spanners

Given an undirected unweighted graph $G = (V, E)$ on n vertices and a distance threshold $D \in \mathbb{Z}^+$, in this section we consider the problem of constructing sparse $(1, \beta)$ $\mathcal{P}$-spanners of G for the set $\mathcal{P} = \{(u, v) \in V \times V : \delta_G(u, v) \geq D\}$. We present our algorithm for $\beta = 4$ in Section 4.1 and we present our algorithm for $\beta = 4 \log D$ (assuming $D \geq 2$) in Section 4.2.

4.1 A (1,4) D-Spanner

Our path-buying strategy here uses the function $\mathsf{cost}(p)$ (Definition 1) that was used in the algorithms in Section 3 and it also uses a new function $\mathsf{value}_X(p)$ defined below. For a vertex v on p and a cluster C with at least one vertex on p, let $\delta_p(v, C)$ be the distance in p between v and C, i.e., the least length of a subpath of p between v and a vertex in C.

Definition 3. *For any path p in G and subgraph X of G, let* $\mathsf{value}_X(p)$ *be the number of pairs (v, C) such that $\delta_p(v, C) < \delta_X(v, C)$, where vertex v and cluster C are incident on p.*

That is, for a path p, $\mathsf{value}_X(p)$ counts the (vertex, cluster) pairs incident on p whose distance in p is strictly smaller than their distance in the subgraph X. Our algorithm to compute a (1,4) $\mathcal{P}$-spanner of G for the set $\mathcal{P} = \{(u, v) : \delta_G(u, v) \geq D\}$ is presented below.

1. *Clustering.* Call the clustering procedure with $h = \lceil \sqrt{n} \cdot (\ln n/D)^{1/4} \rceil$ to compute the clustering $\mathcal{C}$ and the post-clustering subgraph $G_{\mathcal{C}}$. Initialize H to $G_{\mathcal{C}}$.
2. *Path-buying.* For each $\rho \in \mathcal{R}$ do: if ρ satisfies $\mathsf{cost}(\rho) \leq \mathsf{value}_H(\rho) \cdot \sqrt{\frac{\ln n}{D}}$ in the current subgraph H, then add ρ to H.
3. *Adding BFS trees.* Select $O(h)$ cluster indices as given by Lemma 2. Add the edges of the BFS trees rooted at the corresponding cluster centers to H. Return this subgraph H.

Since $h = \lceil \sqrt{n} \cdot (\ln n/D)^{1/4} \rceil$, a path $\rho \in \mathcal{R}$ with cost $\geq \sqrt{D \ln n}$ has cost at least $(n \ln n)/h^2$, thus such a path is necessarily high-cost. We know from Corollary 1 that edges added in step 3 ensure a (1,2) stretch in H for the endpoints of high-cost paths.

Lemma 7 shows a (1,4) stretch in H for shortest paths of length $\geq D$ and cost $< \sqrt{D \ln n}$. Lemma 8 bounds the size of the final subgraph H. Their proofs are omitted here. Theorem 3 from Section 1 follows from these lemmas.

Lemma 7. *If $p \in \mathcal{R}$ with $|p| \geq D$ and $\mathsf{cost}(p) < \sqrt{D \ln n}$, then there is a path of length $\leq |p| + 4$ between p's endpoints in our subgraph at the end of step 2.*

Lemma 8. *The final subgraph H has size $O(n^{3/2} \cdot (\frac{\log n}{D})^{1/4})$.*

4.2 A $(1, 4 \log D)$ D-Spanner

Our input is the graph G and a distance threshold $D \in \mathbb{Z}^+$. Our path-buying step here will use the function $\mathsf{cost}_X(p)$ which is the number of edges in p that are missing in the subgraph X. We will also use the function $\mathsf{value}_X(p)$ (Definition 3) that we used in our previous algorithm. A pair (v, C) *supports* p in X if this pair of vertex and cluster contributes 1 to $\mathsf{value}_X(p)$.

For a pair of vertices (u, v) with distance $\geq D$ in G and whose shortest path p_0 is "low-cost", the path-buying step in this algorithm starts with p_0 and builds a sequence of u-v paths $p_1, p_2, \ldots$ using the subroutine next-path till it finds an *affordable* path. Such an idea was seen before in [13]. Our overall algorithm is given below and the subroutine next-path is described later.

1. *Clustering.* Call the clustering procedure with parameter $h = \lceil \sqrt{(n \ln n)/D} \rceil$. This computes the clustering $\mathcal{C}$ and the post-clustering subgraph $G_{\mathcal{C}}$. Initialize H to $G_{\mathcal{C}}$.

2. *Path-buying.* For each $(u, v) \in (V \times V)$ such that $\delta_G(u, v) \geq D$ and $\mathsf{cost}_H(p_0)$ is less than D (where $p_0 \in \mathcal{R}$ is the shortest u-v path in G) do:
   - for $i = 0, 1, 2, \ldots$ do
     (i) If $\mathsf{cost}_H(p_i) \leq 108 \cdot \mathsf{value}_H(p_i)/D$, then add p_i to H and quit the inner for-loop.
     (ii) Else construct the u-v path p_{i+1} using the subroutine $\mathsf{next\text{-}path}(p_i)$.
3. *Adding BFS trees.* Select $O(h)$ cluster indices as given by Lemma 2. Add the edges of the BFS trees rooted at the corresponding cluster centers to H. Return this subgraph H.

It can be shown that the condition in step 2(i) will have to be satisfied for some $i \leq \lceil \log_{9/5} D \rceil$. We now describe the subroutine $\mathsf{next\text{-}path}(p_i)$, whose input p_i satisfies $\mathsf{cost}_X(p_i) > 108 \cdot \mathsf{value}_X(p_i)/D$ in the current subgraph X. Also $\delta_G(u, v) \geq D$, where u and v are the endpoints of p_i. The path p_i need not be a minimum length u-v path, however we will maintain the following invariant: for every cluster $C \in \mathcal{C}$, there are at most 3 vertices of C on p_i.

The above property is true for p_0 since this is a shortest path. We now assume that p_i satisfies this property and construct a u-v path p_{i+1} that also satisfies this property. Additionally, $|p_{i+1}|$ and $\mathsf{cost}_X(p_{i+1})$ will satisfy Claim 1.

Claim 1 $|p_{i+1}| \leq |p_i| + 2$ *and* $\mathsf{cost}_X(p_{i+1}) \leq 5/9 \cdot \mathsf{cost}_X(p_i)$.

The subroutine $\mathsf{next\text{-}path}(p_i)$ constructs p_{i+1} as follows.

– Since $\delta_G(u, v) \geq D$, the path p_i has $\geq D$ edges. Let x be a vertex on p_i such that both the u-x subpath (call this left) and the x-v subpath (call this right) have $\geq \lfloor D/2 \rfloor$ edges.

– Let $\mathsf{cost}_X(p_i) = t$. So p_i has t edges that are missing in X. Thus either left or right has at least $\lceil t/2 \rceil$ edges that are missing in X. Assume it is right. Let q denote the longest suffix of right containing $\lfloor t/18 \rfloor$ missing edges. (If it was left that had $\geq \lceil t/2 \rceil$ missing edges, then q would have been a *prefix* of left.)

– Every edge missing in X has to be an inter-cluster edge, so these $\lfloor t/18 \rfloor$ missing edges are incident on $\lfloor t/18 \rfloor + 1 \geq t/18$ clustered vertices. In case $\lfloor t/18 \rfloor = 0$, this one clustered vertex is the endpoint of q other than b: this vertex has to be clustered by q's maximality.

– A single cluster has at most 3 vertices on p_i (by our invariant), hence there are $\geq t/54$ distinct clusters incident on q. Consider the set T of all pairs (w, C), where C is one of these $\geq t/54$ clusters incident on q and w is one of the $\geq D/2$ vertices of left. So $|T| \geq t \cdot D/108$.

– We did not buy p_i, so it has to be the case that $t > 108 \cdot \mathsf{value}_X(p_i)/D$, i.e., $\mathsf{value}_X(p_i) < tD/108$. Hence there exists a pair in T that does not support p_i in X. In other words, for this pair $(r, C_0) \in T$, we have $\delta_X(r, C_0) \leq \delta_{p_i}(r, C_0)$.

– We now construct a u-v path p' by combining the prefix of left between u and r with the shortest r-C_0 path in X, along with a path of length ≤ 2 in $G_{\mathcal{C}}$ between two vertices in C_0, and the suffix of right between C_0 and v.

– The path p' need not obey our invariant that every cluster has ≤ 3 vertices on p'. But using the length ≤ 2 paths in $G_{\mathcal{C}}$ between every pair of vertices in

the same cluster, it is easy to modify (in fact, shorten) p' so that this property holds. Call the resulting path p_{i+1}.

Proof of Claim 1. Recall that the path p' is made up of the following subpaths: (i) the subpath of p_i between u and r, (ii) an r-C_0 path in X of length $\leq \delta_{p_i}(r, C_0)$, (iii) a path of length ≤ 2 in X between 2 vertices of C_0, (iv) the subpath of p_i between the last vertex of C_0 and v.

It is easy to see that the sum of subpaths (i), (ii), and (iv) is at most $|p_i|$. Thus $|p'| \leq |p_i| + 2$. Since $|p_{i+1}| \leq |p'|$, we have $|p_{i+1}| \leq |p_i| + 2$.

We now bound the cost of p' in terms of the cost of p_i. It follows from our construction that among the 4 subpaths that we added to form p', subpaths (ii) and (iii) use only edges in X. So these have cost 0 in X.

Subpath (iv) has at most $\lfloor t/18 \rfloor$ edges that are missing in X, where $t = \mathsf{cost}_X(p_i)$. Similarly, subpath (i) is a prefix of left, which has at most $\lfloor t/2 \rfloor$ edges that are missing in X. Thus $\mathsf{cost}_X(p') \leq \lfloor t/2 \rfloor + \lfloor t/18 \rfloor \leq 5/9 \cdot t = 5/9 \cdot \mathsf{cost}_X(p_i)$.

Every edge in p_{i+1} that is not in p' is an edge of $G_{\mathcal{C}}$ (thus an edge of X). So $\mathsf{cost}_X(p_{i+1}) \leq \mathsf{cost}_X(p')$. Hence $\mathsf{cost}_X(p_{i+1}) \leq 5/9 \cdot \mathsf{cost}_X(p_i)$. □

The size of the final subgraph H can be bounded by $O(n \cdot \sqrt{(n \log n)/D})$ and the correctness of our algorithm follows from Lemma 9 (proof omitted here). For any $D \in \mathbb{Z}^+$, the stretch bound shown here is $\max\{4 \log D, 2\}$ which is $4 \log D$, for $D \geq 2$. Theorem 4 from Section 1 follows.

Lemma 9. *Let $D \in \mathbb{Z}^+$. For all $(u, v) \in V \times V$ such that $\delta_G(u, v) \geq D$, we have $\delta_H(u, v) \leq \delta_G(u, v) + \max\{4 \log D, 2\}$, where H is the final subgraph.*

Acknowledgments. We thank the reviewers for their helpful comments.

References

1. Awerbuch, B., Berger, B., Cowen, L., Peleg, D.: Near-linear time construction of sparse neighborhood covers. SIAM Journal on Computing 28(1), 263–277 (1998)
2. Awerbuch, B., Peleg, D.: Routing with polynomial communication-space trade-off. SIAM Journal on Discrete Math 5(2), 151–162 (1992)
3. Baswana, S., Kavitha, T.: Faster algorithms for all-pairs approximate shortest paths in undirected graphs. SIAM Journal on Computing 39(7), 2865–2896 (2010)
4. Baswana, S., Kavitha, T., Mehlhorn, K., Pettie, S.: Additive Spanners and (α, β)-Spanners. ACM Transactions on Algorithms 7(1), 5 (2010)
5. Baswana, S., Sen, S.: A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 384–396. Springer, Heidelberg (2003)
6. Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in $O(n^2 \log n)$ time. In: Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 271–280 (2004)
7. Bollobás, B., Coppersmith, D., Elkin, M.: Sparse Distance Preservers and Additive Spanners. In: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 414–423 (2003)

8. Chechik, S.: New Additive Spanners. In: Proc. 24th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 498–512 (2013)
9. Cohen, E.: Fast algorithms for constructing t-spanners and paths of stretch t. In: Proc. 34th IEEE Symp. on Foundations of Computer Science (FOCS), pp. 648–658 (1993)
10. Coppersmith, D., Elkin, M.: Sparse source-wise and pair-wise distance preservers. In: Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 660–669 (2005)
11. Cowen, L.J.: Compact routing with minimum stretch. Journal of Algorithms 28, 170–183 (2001)
12. Cowen, L.J., Wagner, C.G.: Compact roundtrip routing in directed networks. Journal of Algorithms 50(1), 79–95 (2004)
13. Cygan, M., Grandoni, F., Kavitha, T.: On Pairwise Spanners. In: Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 209–220 (2013)
14. Dor, D., Halperin, S., Zwick, U.: All-pairs almost shortest paths. SIAM Journal on Computing 29(5), 1740–1759 (2004)
15. Elkin, M.: Computing almost shortest paths. ACM Transactions on Algorithms 1(2), 283–323 (2005)
16. Elkin, M., Peleg, D.: $(1 + \epsilon, \beta)$-spanner construction for general graphs. SIAM Journal on Computing 33(3), 608–631 (2004)
17. Gavoille, C., Peleg, D., Perennes, S., Raz, R.: Distance labeling in graphs. In: Proc. 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 210–219 (2001)
18. Halperin, S., Zwick, U.: Unpublished result (1996)
19. Peleg, D.: Proximity-preserving labeling schemes. Journal of Graph Theory 33(3), 167–176 (2000)
20. Peleg, D., Schaffer, A.A.: Graph Spanners. Journal of Graph Theory 13, 99–116 (1989)
21. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. SIAM Journal on Computing 18, 740–747 (1989)
22. Pettie, S.: Low Distortion Spanners. ACM Transactions on Algorithms 6(1) (2009)
23. Roditty, L., Zwick, U.: On dynamic shortest paths problems. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 580–591. Springer, Heidelberg (2004)
24. Roditty, L., Thorup, M., Zwick, U.: Deterministic constructions of approximate distance oracles and spanners. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 261–272. Springer, Heidelberg (2005)
25. Thorup, M., Zwick, U.: Approximate Distance Oracles. Journal of the ACM 52(1), 1–24 (2005)
26. Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: Proc. 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 802–809 (2006)
27. Woodruff, D.P.: Additive Spanners in Nearly Quadratic Time. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 463–474. Springer, Heidelberg (2010)

Linear Kernels and Single-Exponential Algorithms via Protrusion Decompositions*

Eun Jung Kim[1], Alexander Langer[2], Christophe Paul[3], Felix Reidl[2], Peter Rossmanith[2], Ignasi Sau[3], and Somnath Sikdar[2]

[1] CNRS, LAMSADE, Paris, France
eunjungkim78@gmail.com
[2] Theoretical Computer Science, Department of Computer Science, RWTH Aachen University, Germany
{langer,reidl,rossmani,sikdar}@cs.rwth-aachen.de
[3] CNRS, LIRMM, Montpellier, France
{paul,sau}@lirmm.fr

Abstract. We present a linear-time algorithm to compute a decomposition scheme for graphs G that have a set $X \subseteq V(G)$, called a *treewidth-modulator*, such that the treewidth of $G - X$ is bounded by a constant. Our decomposition, called a *protrusion decomposition*, is the cornerstone in obtaining the following two main results. Our first result is that any parameterized graph problem (with parameter k) that has *finite integer index* and such that positive instances have a treewidth-modulator of size $O(k)$ admits a linear kernel on the class of H-topological-minor-free graphs, for any fixed graph H. This result partially extends previous meta-theorems on the existence of linear kernels on graphs of bounded genus and H-minor-free graphs.

Let $\mathcal{F}$ be a fixed finite family of graphs containing at least one planar graph. Given an n-vertex graph G and a non-negative integer k, Planar-$\mathcal{F}$-Deletion asks whether G has a set $X \subseteq V(G)$ such that $|X| \leqslant k$ and $G - X$ is H-minor-free for every $H \in \mathcal{F}$. As our second application, we present the first *single-exponential* algorithm to solve Planar-$\mathcal{F}$-Deletion. Namely, our algorithm runs in time $2^{O(k)} \cdot n^2$, which is asymptotically optimal with respect to k. So far, single-exponential algorithms were only known for special cases of the family $\mathcal{F}$.

Keywords: parameterized complexity, linear kernels, algorithmic meta-theorems, sparse graphs, single-exponential algorithms, graph minors.

1 Introduction

This work contributes to the two main areas of parameterized complexity, namely, kernels and fixed-parameter tractable (FPT) algorithms (see, e.g., [11] for an introduction). In many cases, the key ingredient in order to solve a hard graph

* This article replaces and extends the results of [CoRR, abs/1201.2780, 2012]. Research funded by DFG-Project RO 927/12-1 "Theoretical and Practical Aspects of Kernelization", ANR project AGAPE (ANR-09-BLAN-0159), and the Languedoc-Roussillon Project "Chercheur d'avenir" KERNEL.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 613–624, 2013.

problem is to find an appropriate *decomposition* of the input graph, which allows to take advantage of the structure given by the graph class and/or the problem under study. In this article we follow this paradigm and present (in Section 3) a novel linear-time algorithm to compute a decomposition for graphs G that have a set $X \subseteq V(G)$, called *t-treewidth-modulator*, such that the treewidth of $G - X$ is at most some constant $t - 1$. We then exploit this decomposition in two different ways: to *analyze* the size of kernels and to obtain *efficient* FPT algorithms. We would like to note that similar decompositions have already been (explicitly or implicitly) used for obtaining polynomial kernels [1, 4, 13, 15, 18].

Linear Kernels. During the last decade, a plethora of results emerged on linear kernels for graph-theoretic problems restricted to *sparse* graph classes. A celebrated result by Alber *et al.* [1] prompted an explosion of research papers on linear kernels on planar graphs. Guo and Niedermeier [18] designed a general framework and showed that problems that satisfy a certain "distance property" have linear kernels on planar graphs. Bodlaender *et al.* [4] provided a meta-theorem for problems to have a linear kernel on graphs of bounded genus. Fomin *et al.* [15] extended these results for bidimensional problems on H-minor-free graphs. A common feature of these meta-theorems on sparse graphs is a *decomposition scheme* of the input graph that, loosely speaking, allows to deal with each part of the decomposition independently. For instance, the approach of [18], which is much inspired from [1], is to consider a so-called *region decomposition* of the input planar graph. The key point is that in an appropriately reduced Yes-instance, there are $O(k)$ regions and each one has constant size, yielding the desired linear kernel. This idea was generalized in [4] to graphs on surfaces, where the role of regions is played by *protrusions*, which are graphs with small treewidth and small boundary (see Section 2 for details). The resulting decomposition is called *protrusion decomposition*. A crucial point is that while the reduction rules of [1] are *problem-dependent*, those of [4] are *automated*, relying on a property called *finite integer index* (FII), which was introduced by Bodlaender and de Fluiter [5]. Having FII essentially guarantees that "large" protrusions of an instance can be replaced by "small" equivalent gadget graphs. This operation is usually called the *protrusion replacement rule*. FII is also of central importance to the approach of [15] on H-minor-free graphs.

In the spirit of the above results, our algorithm to compute protrusion decompositions allows us to prove that we can obtain (in Section 4) linear kernels on a larger class of sparse graphs. A parameterized problem is *treewidth-bounding* if Yes-instances have a t-treewidth-modulator of size $O(k)$ for some constant t. Our first main result is:

Theorem I. Fix a graph H. Let Π be a parameterized graph problem on the class of H-topological-minor-free graphs that is treewidth-bounding and has FII. Then Π admits a linear kernel.

It turns out that a host of problems including Treewidth-t Vertex Deletion, Chordal Vertex Deletion, Interval Vertex Deletion, Edge Dominating Set, to name a few, satisfy the conditions of our theorem. Since for any

fixed graph H, the class of H-topological-minor-free graphs strictly contains the class of H-minor-free graphs, our result is in fact an extension of the results of Fomin *et al.* [15].

Efficient FPT algorithms. In the second part of the paper (Section 5) we are interested in *single-exponential* algorithms, that is, algorithms that solve a parameterized problem with parameter k on an n-vertex graph in time $2^{O(k)} \cdot n^{O(1)}$. Let $\mathcal{F}$ be a finite family of graphs containing at least one planar graph. In the Planar-$\mathcal{F}$-Deletion problem, given a graph G and a non-negative integer parameter k as input, we are asked whether G has a set $X \subseteq V(G)$ such that $|X| \leqslant k$ and $G - X$ is H-minor-free for every $H \in \mathcal{F}$.

Note that Vertex Cover and Feedback Vertex Set correspond to the special cases of $\mathcal{F} = \{K_2\}$ and $\mathcal{F} = \{K_3\}$, respectively. Recent works have provided, using quite different techniques, single-exponential algorithms for the particular cases $\mathcal{F} = \{K_3, T_2\}$ [7, 22], $\mathcal{F} = \{\theta_c\}$ [19], or $\mathcal{F} = \{K_4\}$ [20]. The Planar-$\mathcal{F}$-Deletion problem was first stated by Fellows and Langston [12], who proposed a non-uniform $f(k) \cdot n^2$-time algorithm for some function $f(k)$, relying on the meta-theorem of Robertson and Seymour [24]. Explicit bounds on the function $f(k)$ can be obtained via dynamic programming. Indeed, as the Yes-instances of Planar-$\mathcal{F}$-Deletion have treewidth $O(k)$, using standard dynamic programming techniques on graphs of bounded treewidth (see for instance [2]), it can be seen that Planar-$\mathcal{F}$-Deletion can be solved in time $2^{2^{O(k \log k)}} \cdot n^2$. Recently, Fomin *et al.* [14] provided a $2^{O(k)} \cdot n \log^2 n$-time algorithm for the Planar-Connected-$\mathcal{F}$-Deletion problem, which is the special case of Planar-$\mathcal{F}$-Deletion when *every* graph in the family $\mathcal{F}$ is *connected*. In this paper we get rid of the connectivity assumption:

Theorem II. Planar-$\mathcal{F}$-Deletion can be solved in time $2^{O(k)} \cdot n^2$.

This result unifies, generalizes, and simplifies a number of results given in [6, 8, 14, 17, 19, 20]. Besides the fact that removing the connectivity constraint is an important theoretical step towards the general case where $\mathcal{F}$ may not contain any planar graph, it turns out that many natural such families $\mathcal{F}$ do contain disconnected planar graphs [10]. An important feature of our approach, in comparison with previous work [14, 19, 20], is that our algorithm *does not use any reduction rule*. This is because if $\mathcal{F}$ may contain disconnected graphs, Planar-$\mathcal{F}$-Deletion has not FII for some choices of $\mathcal{F}$, and then the protrusion replacement rule cannot be applied. A more in-depth discussion can be found in the full version. Finally, it should also be noted that the function $2^{O(k)}$ in Theorem II is best possible assuming the Exponential Time Hypothesis (ETH), as Vertex Cover cannot be solved in time $2^{o(k)} \cdot \mathit{poly}(n)$ unless the ETH fails.

Further research. Concerning our kernelization algorithms, a natural question is whether similar results can be obtained for an even larger class of sparse graphs. As discussed in the full version, obtaining a kernel for Treewidth-t Vertex Deletion on graphs of bounded expansion is as hard as on general graphs, and according to Fomin *et al.* [14], this problem has a kernel of size $k^{O(t)}$ on general graphs, and no uniform polynomial kernel (a polynomial kernel whose degree

does not depend on t) is known. This fact makes us suspect that our kernelization result may settle the limit of meta-theorems about the existence of linear, or even uniform polynomial, kernels on sparse graph classes. We would like to note that the degree of the polynomial of the running time of our kernelization algorithm depends linearly on the size of the excluded topological minor H. It seems that the recent *fast protrusion replacer* of Fomin *et al.* [14] could be applied to get rid of this dependency on H.

Concerning the Planar-$\mathcal{F}$-Deletion problem, no single-exponential algorithm is known when the family $\mathcal{F}$ does not contain any planar graph. Is it possible to find such a family, or can it be proved that, under some complexity assumption, a single-exponential algorithm is not possible? Very recently, a randomized (Monte Carlo) constant-factor approximation algorithm for Planar-$\mathcal{F}$-Deletion has been given by Fomin *et al.* [14]. Finding a deterministic constant-factor approximation remains open.

2 Preliminaries

We use standard graph-theoretic notation (see [9] and the full version for any undefined terminology). Given a graph G, we let $V(G)$ denote its vertex set and $E(G)$ its edge set. A *minor* of G is a graph obtained from a subgraph of G by contracting zero or more edges. A *topological minor* of G is a graph obtained from a subgraph of G by contracting zero or more edges, such that each contracted edge has at least one endpoint with degree at most two. A graph G is *H-(topological)-minor-free* if G does not contain H as a (topological) minor.

A *parameterized graph problem* Π is a set of tuples (G, k), where G is a graph and $k \in \mathbb{N}_0$. If $\mathcal{G}$ is a graph class, we define Π *restricted to* $\mathcal{G}$ as $\Pi_{\mathcal{G}} = \{(G, k) \mid (G, k) \in \Pi \text{ and } G \in \mathcal{G}\}$. A parameterized problem Π is *fixed-parameter tractable* (FPT for short) if there exists an algorithm that decides instances (x, k) in time $f(k) \cdot poly(|x|)$, where f is a function of k alone. A *kernelization algorithm*, or just *kernel*, for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}_0$ is an algorithm that given $(x, k) \in \Gamma^* \times \mathbb{N}_0$ outputs, in time polynomial in $|x| + k$, an instance $(x', k') \in \Gamma^* \times \mathbb{N}_0$ such that $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$, and $|x'|, k' \leqslant g(k)$, where g is some computable function. The function g is called the *size* of the kernel. If $g(k) = k^{O(1)}$ or $g(k) = O(k)$, we say that Π admits a *polynomial kernel* and a *linear kernel*, respectively.

Given a graph $G = (V, E)$, we denote a *tree-decomposition of* G by $(T, \{W_x \mid x \in V(T)\})$, where T is a tree and $\{W_x \mid x \in V(T)\}$ are the bags of the decomposition. We refer the reader to Diestel's book [9] for an introduction to the theory of treewidth.

We restate the main definitions of the protrusion machinery developed in [4, 15]. Given a graph $G = (V, E)$ and a set $W \subseteq V$, we define $\partial_G(W)$ as the set of vertices in W that have a neighbor in $V \setminus W$. For a set $W \subseteq V$ the neighborhood of W is $N^G(W) = \partial_G(V \setminus W)$. Superscripts and subscripts are omitted when it is clear which graph is being referred to.

Given a graph G, a set $W \subseteq V(G)$ is a *t-protrusion* of G if $|\partial_G(W)| \leqslant t$ and $\mathbf{tw}(G[W]) \leqslant t-1$.[1] If W is a t-protrusion, the vertex set $W' = W \setminus \partial_G(W)$ is the *restricted protrusion of* W. We call $\partial_G(W)$ the *boundary* and $|W|$ the *size* of the t-protrusion W of G. Given a restricted t-protrusion W', we denote its *extended protrusion* by $W'^{+} = W' \cup N(W')$.

A *t-boundaried graph* is a graph $G = (V, E)$ with a set $\mathbf{bd}(G)$ (called the *boundary*[2] or the *terminals* of G) of t distinguished vertices labeled 1 through t. Let $\mathcal{G}_t$ denote the class of t-boundaried graphs, with graphs from $\mathcal{G}$. If $W \subseteq V$ is an r-protrusion in G, then we let G_W be the r-boundaried graph $G[W]$ with boundary $\partial_G(W)$, where the vertices of $\partial_G(W)$ are assigned labels 1 through r according to their order in G. *Gluing* two t-boundaried graphs G_1 and G_2 creates the graph $G_1 \oplus G_2$ obtained by taking the disjoint union of G_1 and G_2 and identifying each vertex in $\mathbf{bd}(G_1)$ with its corresponding vertex in $\mathbf{bd}(G_2)$, i.e. those vertices sharing the same label.

If G_1 is a subgraph of G with a t-boundary $\mathbf{bd}(G_1)$, *ungluing G_1 from G* creates the t-boundaried graph $G \ominus G_1 = G - (V(G_1) \setminus \mathbf{bd}(G_1))$ with boundary $\mathbf{bd}(G \ominus G_1) = \mathbf{bd}(G_1)$, the vertices of which are assigned labels according to their order in the graph G. Let W be a t-protrusion in G, let G_W denote the graph $G[W]$ with boundary $\mathbf{bd}(G_W) = \partial_G(W)$, and let G_1 be a t-boundaried graph. Then *replacing G_W by G_1* corresponds to the operation $(G \ominus G_W) \oplus G_1$.

An *(α, t)-protrusion decomposition* of a graph G is a partition $\mathcal{P} = Y_0 \uplus Y_1 \uplus \cdots \uplus Y_\ell$ of $V(G)$ such that: (1) for every $1 \leqslant i \leqslant \ell$, $N(Y_i) \subseteq Y_0$; (2) $\max\{\ell, |Y_0|\} \leqslant \alpha$; (3) for every $1 \leqslant i \leqslant \ell$, $Y_i \cup N_{Y_0}(Y_i)$ is a t-protrusion of G. Y_0 is called the *separating part* of $\mathcal{P}$. Hereafter, the value of t will be fixed to some constant. When G is the input of a parameterized graph problem with parameter k, we say that an (α, t)-protrusion decomposition of G is *linear* whenever $\alpha = O(k)$.

Let $\Pi_{\mathcal{G}}$ be a parameterized graph problem restricted to a class $\mathcal{G}$ and let G_1, G_2 be two t-boundaried graphs in $\mathcal{G}_t$. We say that $G_1 \equiv_{\Pi,t} G_2$ if there exists a constant $\Delta_{\Pi,t}(G_1, G_2)$ (that depends on Π, t, and the ordered pair (G_1, G_2)) such that for all t-boundaried graphs G_3 and for all k: (1) $G_1 \oplus G_3 \in \mathcal{G}$ iff $G_2 \oplus G_3 \in \mathcal{G}$; (2) $(G_1 \oplus G_3, k) \in \Pi$ iff $(G_2 \oplus G_3, k + \Delta_{\Pi,t}(G_1, G_2)) \in \Pi$. We say that the problem $\Pi_{\mathcal{G}}$ has *finite integer index in the class* $\mathcal{G}$ iff for every integer t, the equivalence relation $\equiv_{\Pi,t}$ has finite index. In the case that $(G_1 \oplus G, k) \notin \Pi$ or $G_1 \oplus G \notin \mathcal{G}$ for *all* $G \in \mathcal{G}_t$, we set $\Delta_{\Pi,t}(G_1, G_2) = 0$. Note that $\Delta_{\Pi,t}(G_1, G_2) = -\Delta_{\Pi,t}(G_2, G_1)$.

If a parameterized problem has FII then it can be reduced by "replacing protrusions", hinging on the fact that each "large" protrusion can be replaced by a "small" gadget from the same equivalence class that behaves similar w.r.t. to the problem at hand. Exchanging G_1 by a gadget G_2 changes the parameter k by $\Delta_{\Pi,t}(G_1, G_2)$. Lemma 1 guarantees the existence of a set of representatives such that the replacement operation does *not* increase the parameter. In the full version we show how to find protrusions in polynomial time and how to identify by which representative to replace a protrusion, assuming that we are *given* the

[1] In [4], $\mathbf{tw}(G[W]) \leqslant t$, but we want the size of the bags to be at most t.
[2] Usually denoted by $\partial(G)$, but this collides with our usage of ∂.

set of representatives, an assumption we make from now on. This makes our algorithms in Section 4 *non-uniform*, as those in previous works [4, 13–15].

Lemma 1. [⋆] [3] *Let Π be a parameterized graph problem that has FII in a graph class $\mathcal{G}$. Then for every t, there exists a finite set $\mathcal{R}_t$ of t-boundaried graphs such that for each $G \in \mathcal{G}_t$ there exists $G' \in \mathcal{R}_t$ such that $G \equiv_{\Pi,t} G'$ and $\Delta_{\Pi,t}(G, G') \geqslant 0$.*

For a parameterized problem Π that has FII in the class $\mathcal{G}$, let $\mathcal{R}_t$ denote the set of representatives as in Lemma 1. The *protrusion limit* of $\Pi_{\mathcal{G}}$ is a function $\rho_{\Pi_{\mathcal{G}}} : \mathbb{N} \to \mathbb{N}$ defined as $\rho_{\Pi_{\mathcal{G}}}(t) = \max_{G \in \mathcal{R}_t} |V(G)|$. We drop the subscript when it is clear which graph problem is being referred to. We also define $\rho'(t) := \rho(2t)$.

Lemma 2 ([4]). [⋆] *Let Π be a parameterized graph problem with FII in $\mathcal{G}$ and let $t \in \mathbb{N}$ be a constant. For a graph $G \in \mathcal{G}$, if one is given a t-protrusion $X \subseteq V(G)$ such that $\rho'_{\Pi_{\mathcal{G}}}(t) < |X|$, then one can, in time $O(|X|)$, find a $2t$-protrusion W such that $\rho'_{\Pi_{\mathcal{G}}}(t) < |W| \leqslant 2 \cdot \rho'_{\Pi_{\mathcal{G}}}(t)$.*

3 Constructing Protrusion Decompositions

We present our algorithm to compute protrusion decompositions. Algorithm 1 marks the bags of a tree-decomposition of an input graph G that comes equipped with a t-treewidth-modulator $X \subseteq V(G)$. Our algorithm also takes an additional integer parameter r, which depends on the graph class to which G belongs and the precise problem one might want to solve (see Sections 4 and 5 for details).

Note that an optimal tree-decomposition of every connected component C of $G - X$ such that $|N_X(C)| \geqslant r$ can be computed in time linear in $n = |V(G)|$ using the algorithm of Bodlaender [3]. In the full version we sketch how the Large-subgraph marking step can be implemented using standard dynamic programming techniques. It is quite easy to see that Algorithm 1 runs in linear time.

Lemma 3. [⋆] *Let Y_0 be the set of vertices computed by Algorithm 1. Every connected component C of $G - Y_0$ satisfies $|N_X(C)| < r$ and $|N_{Y_0}(C)| < r + 2t$, and thus forms a restricted protrusion.*

Given a graph G and a subset $S \subseteq V(G)$, we define a *cluster of $G - S$* as a maximal collection of connected components of $G - S$ with the same neighborhood in S. Note that the set of all clusters of $G - S$ induces a partition of the set of connected components of $G - S$, which can be easily found in linear time if G and S are given. By Lemma 3 and using the fact that $\mathbf{tw}(G - X) \leqslant t - 1$, the following proposition follows.

Proposition 1. *Let r, t be two positive integers, let G be a graph and $X \subseteq V(G)$ such that $\mathbf{tw}(G - X) \leqslant t - 1$, let $Y_0 \subseteq V(G)$ be the output of Algorithm 1 with input (G, X, r), and let $Y_1, \ldots, Y_\ell$ be the set of all clusters of $G - Y_0$. Then $\mathcal{P} := Y_0 \uplus Y_1 \uplus \cdots \uplus Y_\ell$ is a $(\max\{\ell, |Y_0|\}, 2t + r)$-protrusion decomposition of G.*

[3] The proofs of the results marked with '[⋆]' can be found in [CoRR, abs/1207.0835].

Algorithm 1: BAG MARKING ALGORITHM

Input: A graph G, a subset $X \subseteq V(G)$ such that $\mathbf{tw}(G - X) \leqslant t - 1$, and an integer $r > 0$.

Set $\mathcal{M} \leftarrow \emptyset$ as the set of marked bags;
Compute an optimal rooted tree-decomposition $\mathcal{T}_C = (T_C, \mathcal{B}_C)$ of every connected component C of $G - X$ such that $|N_X(C)| \geqslant r$;

Repeat the following loop for every rooted tree-decomposition $\mathcal{T}_C$;
while $\mathcal{T}_C$ *contains an unprocessed bag* **do**

Let B be an unprocessed bag at the farthest distance from the root of $\mathcal{T}_C$;

[**LCA marking step**]
if B *is the LCA of two marked bags of* $\mathcal{M}$ **then**
$\mathcal{M} \leftarrow \mathcal{M} \cup \{B\}$ and remove the vertices of B from every bag of $\mathcal{T}_C$;

[**Large-subgraph marking step**]
else if G_B *contains a connected component* C_B *such that* $|N_X(C_B)| \geqslant r$ **then**
$\mathcal{M} \leftarrow \mathcal{M} \cup \{B\}$ and remove the vertices of B from every bag of $\mathcal{T}_C$;

Bag B is now processed;

return $Y_0 = X \cup V(\mathcal{M})$;

In other words, each cluster of $G - Y_0$ is a restricted $(2t+r)$-protrusion. Note that Proposition 1 neither bounds ℓ nor $|Y_0|$. In the sequel, we will use Algorithm 1 and Proposition 1 to give explicit bounds on ℓ and $|Y_0|$, in order to achieve our two main results.

4 Linear Kernels on Graphs Excluding a Topological Minor

In this section we prove our first main result (Theorem I). We then state a number of concrete problems that satisfy the structural constraints imposed by this theorem and discuss these constraints in the context of previous work in this area. With the protrusion machinery of Section 2 at hand, we can now describe the protrusion reduction rule. In the following, we will drop the subscript from the protrusion limit functions ρ_{Π} and ρ'_{Π}.

Reduction Rule 1 (Protrusion reduction rule). *Let $\Pi_{\mathcal{G}}$ denote a parameterized graph problem restricted to some graph class $\mathcal{G}$, let $(G, k) \in \Pi_{\mathcal{G}}$ be a* YES*-instance of $\Pi_{\mathcal{G}}$, and let $t \in \mathbb{N}$ be a constant. Suppose that $W' \subseteq V(G)$ is a t-protrusion of G such that $|W'| > \rho'(t)$. Let $W \subseteq V(G)$ be a $2t$-protrusion of G such that $\rho'(t) < |W| \leqslant 2 \cdot \rho'(t)$, obtained as described in Lemma 2. We let G_W denote the $2t$-boundaried graph $G[W]$ with boundary $\mathbf{bd}(G_W) = \partial_G(W)$. Let further $G_1 \in \mathcal{R}_{2t}$ be the representative of G_W for the equivalence relation $\equiv_{\Pi,|\partial(W)|}$ as defined in Lemma 1. The protrusion reduction rule (for boundary size t) is the following:* Reduce (G, k) to $(G', k') = (G \ominus G_W \oplus G_1, k - \Delta_{\Pi,2t}(G_1, G_W))$.

By Lemma 1, the parameter in the new instance does not increase. The safety of the above reduction rule is shown in the full version. Note that if (G, k) is reduced w.r.t. the protrusion reduction rule with boundary size β, then for all $t \leqslant \beta$, every t-protrusion W of G has size at most $\rho'(t)$.

Definition 1 (Treewidth-bounding). *A parameterized graph problem $\Pi_{\mathcal{G}}$ is called* (s,t)-treewidth-bounding *for a function $s\colon \mathbb{N} \to \mathbb{N}$ and a constant t if for all $(G,k) \in \Pi$ there exists $X \subseteq V(G)$ (the* treewidth-modulator*) such that $|X| \leqslant s(k)$ and $\mathbf{tw}(G - X) \leqslant t-1$. We call $\Pi_{\mathcal{G}}$* treewidth-bounding *on a graph class $\mathcal{G}$ if this condition holds under the restriction that $G \in \mathcal{G}$. We call s the* treewidth-modulator size *and t the* treewidth bound *of the problem $\Pi_{\mathcal{G}}$.*

We assume in the following that the problem $\Pi_{\mathcal{G}}$ at hand is (s,t)-treewidth-bounding. Note that in general s, t depend on $\Pi_{\mathcal{G}}$ and $\mathcal{G}$.

We first prove a slight generalization of Theorem I which highlights all the key ingredients required. To this end, we define the *constriction* operation, which essentially shrinks paths into edges.

Definition 2 (Constriction). *Let G be a graph and let $\mathcal{P}$ be a set of paths in G such that for each $P \in \mathcal{P}$ we have* (1) *the endpoints of P are not connected by an edge in G; and* (2) *for all $P' \in \mathcal{P}$, with $P' \neq P$, $V(P) \cap V(P')$ has at most one vertex, which must also be an endpoint of both paths. We define the* constriction *of G under $\mathcal{P}$, denoted by $G|_{\mathcal{P}}$, as the graph H obtained by connecting the endpoints of each $P \in \mathcal{P}$ by an edge and then removing all inner vertices of P.*

We say that H is a *d-constriction* of G if there exists $G' \subseteq G$ and a set of paths $\mathcal{P}$ in G' such that $d = \max_{P \in \mathcal{P}} |P|$ and $H = G'|_{\mathcal{P}}$. Given graph classes $\mathcal{G}, \mathcal{H}$ and some integer $d \geqslant 2$, we say that $\mathcal{G}$ *d-constricts into* $\mathcal{H}$ if for every $G \in \mathcal{G}$, every possible d-constriction H of G is contained in the class $\mathcal{H}$. For the case that $\mathcal{G} = \mathcal{H}$ we say that $\mathcal{G}$ is *closed under d-constrictions*. We will call $\mathcal{H}$ the *witness* class, as the proof of Theorem 1 works by taking an input graph G and constricting it into some witness graph H whose properties will yield the desired bound on $|G|$. We let $\omega(G)$ denote the size of a largest clique in G and $\#\omega(G)$ the total number of cliques in G (not necessarily maximal ones).

Theorem 1. [$\star$] *Let $\mathcal{G}, \mathcal{H}$ be graph classes closed under taking subgraphs such that $\mathcal{G}$ d-constricts into $\mathcal{H}$ for a fixed constant $d \in \mathbb{N}$. Assume that $\mathcal{H}$ has the property that there exist functions $f_E, f_{\#\omega}\colon \mathbb{N} \to \mathbb{N}$ and a constant $\omega_{\mathcal{H}}$ (depending only on $\mathcal{H}$) such that for each graph $H \in \mathcal{H}$ the following conditions hold:*

$$|E(H)| \leqslant f_E(|H|), \;\; \#\omega(H) \leqslant f_{\#\omega}(|H|), \;\; \text{and} \;\; \omega(H) < \omega_{\mathcal{H}}.$$

Let Π be a parameterized graph problem that has FII and is (s,t)-treewidth-bounding, both on the graph class $\mathcal{G}$. Define $x_k := s(k) + 2t \cdot f_E(s(k))$. Then any reduced instance $(G,k) \in \Pi$ has a protrusion decomposition $V(G) = Y_0 \uplus Y_1 \uplus \cdots \uplus Y_\ell$ such that: (1) $|Y_0| \leqslant x_k$; (2) $|Y_i| \leqslant \rho'(2t + \omega_{\mathcal{H}})$ for $1 \leqslant i \leqslant \ell$; and (3) $\ell \leqslant f_{\#\omega}(x_k) + x_k + 1$. Hence Π restricted to $\mathcal{G}$ admits kernels of size at most $x_k + (f_{\#\omega}(x_k) + x_k + 1)\rho'(2t + \omega_{\mathcal{H}})$.

Theorem 1 directly implies the following, using the fact that H-topological-minor-free graphs are ϵ-degenerate.

Theorem 2. [$\star$] *Fix a graph H and let $\mathcal{G}_H$ be the class of H-topological-minor-free graphs. Let Π be a parameterized graph-theoretic problem that has FII and is $(s_{\Pi,\mathcal{G}_H}, t_{\Pi,\mathcal{G}_H})$-treewidth-bounding on the class $\mathcal{G}_H$. Then Π admits a kernel of size $O(s_{\Pi,\mathcal{G}_H}(k))$.*

Theorem I is now just a consequence of the special case for which the treewidth-bound is linear. We present concrete problems that are affected by our result.

Corollary 1. *The following problems are linearly treewidth-bounding and have FII on $\mathcal{G}_H$ and hence admit linear kernels on $\mathcal{G}_H$:* Vertex Cover[4]*;* Cluster Vertex Deletion[4]*;* Feedback Vertex Set*;* Chordal Vertex Deletion*;* Interval *and* Proper Interval Vertex Deletion*;* Cograph Vertex Deletion*;* Edge Dominating Set.

Theorem I requires problems to be treewidth-bounding, at first glance, a quite strong restriction. However, the property of being treewidth-bounding appears implicitly or explicitly in previous work on linear kernels on sparse graphs [4,15].

5 Single-Exponential Algorithm for Planar-$\mathcal{F}$-Deletion

This section is devoted to the single-exponential algorithm for the Planar-$\mathcal{F}$-Deletion problem. Let henceforth H_p be some fixed (connected or disconnected) arbitrary planar graph in the family $\mathcal{F}$, and let $r := |V(H_p)|$. First of all, using iterative compression, we reduce the problem to obtaining a single-exponential algorithm for the Disjoint Planar-$\mathcal{F}$-Deletion problem, which is defined as follows: given a graph G and a subset of vertices $X \subseteq V(G)$ such that $G - X$ is H-minor-free for every $H \in \mathcal{F}$, compute a set $\tilde{X} \subseteq V(G)$ disjoint from X such that $|\tilde{X}| < |X|$ and $G - \tilde{X}$ is H-minor-free for every $H \in \mathcal{F}$, if such a set exists. The parameter is $k = |X|$.

The input set X is called the *initial solution* and the set $\tilde{X}$ the *alternative solution*. Let $t_{\mathcal{F}}$ be a constant (depending on the family $\mathcal{F}$) such that $\mathbf{tw}(G - X) \leqslant t_{\mathcal{F}} - 1$ (note that such a constant exists by Robertson and Seymour [23]). The following lemma relies on the fact that being $\mathcal{F}$-minor-free is a hereditary property with respect to induced subgraphs. For a proof, see for instance [6, 19–21].

Lemma 4. *If the parameterized* Disjoint Planar-$\mathcal{F}$-Deletion *problem can be solved in time $c^k \cdot p(n)$, where c is a constant and $p(n)$ is a polynomial in n, then the* Planar-$\mathcal{F}$-Deletion *problem can be solved in time $(c+1)^k \cdot p(n) \cdot n$.*

To solve Disjoint Planar-$\mathcal{F}$-Deletion, we first construct a protrusion decomposition using Algorithm 1 with input (G, X, r). But it turns out that the set Y_0

[4] Listed for completeness; these problems have a kernel with a linear number of vertices on general graphs.

output by Algorithm 1 does not define a *linear* protrusion decomposition of G, which is crucial for our purposes. To circumvent this problem, our strategy is to guess the intersection I of the alternative solution $\tilde{X}$ with the set Y_0. As a result, we obtain Proposition 2, which is fundamental in order to prove Theorem II.

Proposition 2 (Linear protrusion decomposition). *Let (G, X, k) be a* YES*-instance of the* DISJOINT PLANAR-$\mathcal{F}$-DELETION *problem. There exists a $2^{O(k)} \cdot n$-time algorithm that identifies a set $I \subseteq V(G)$ of size at most k and a $(O(k), 2t_{\mathcal{F}} + r)$-protrusion decomposition $\mathcal{P} = Y_0 \uplus Y_1 \uplus \cdots \uplus Y_\ell$ of $G - I$ such that: (1) $X \subseteq Y_0$; and (2) there exists a set $X' \subseteq V(G) \setminus Y_0$ of size at most $k - |I|$ such that $G - \tilde{X}$, with $\tilde{X} = X' \cup I$, is H-minor-free for every graph $H \in \mathcal{F}$.*

Towards the proof of Proposition 2, we need the following ingredient.

Proposition 3 (Thomason [25], Fomin, Oum, and Thilikos [16]). *There is a constant $\alpha < 0.320$ such that every n-vertex graph with no K_r-minor has at most $(\alpha r\sqrt{\log r}) \cdot n$ edges. There is a constant $\mu < 11.355$ such that, for $r > 2$, every n-vertex graph with no K_r-minor has at most $2^{\mu r \log\log r} \cdot n$ cliques.*

For the sake of simplicity, let henceforth $\alpha_r := \alpha r\sqrt{\log r}$ and $\mu_r := 2^{\mu r \log\log r}$. For each guessed set $I \subseteq Y_0$, we denote $G_I := G - I$.

Lemma 5. [$\star$] *If (G, X, k) is a* YES*-instance of the* DISJOINT PLANAR-$\mathcal{F}$-DELETION *problem, then the set $Y_0 = V(\mathcal{M}) \cup X$ of vertices returned by Algorithm 1 has size at most $k + 2t_{\mathcal{F}} \cdot (1 + \alpha_r) \cdot k$.*

Lemma 6. [$\star$] *If $(G_I, Y_0 \setminus I, k - |I|)$ is a* YES*-instance of the* DISJOINT PLANAR-$\mathcal{F}$-DELETION *problem, then the number of clusters of $G_I - Y_0$ is at most $(5t_{\mathcal{F}}\alpha_r\mu_r) \cdot k$, where Y_0 is the set of vertices returned by Algorithm 1.*

We are now ready to prove Proposition 2.

Proof (of Proposition 2). By Lemma 5, we can compute in linear time a set Y_0 of $O(k)$ vertices containing X such that every cluster of $G - Y_0$ is a restricted $(2t_{\mathcal{F}} + r)$-protrusion. If (G, X, k) is a YES-instance of the DISJOINT PLANAR-$\mathcal{F}$-DELETION problem, then there exists a set $\tilde{X}$ of size at most $|X|$ and disjoint from X such that $G - \tilde{X}$ does not contain any graph $H \in \mathcal{F}$ as a minor. Branching on every possible subset of $Y_0 \setminus X$, one can guess the intersection I of $\tilde{X}$ with $Y_0 \setminus X$. By Lemma 5, the branching degree is $2^{O(k)}$. As (G, X, k) is a YES-instance, for at least one of the guessed subsets I, the instance $(G_I, Y_0 \setminus I, k - |I|)$ is a YES-instance of the DISJOINT PLANAR-$\mathcal{F}$-DELETION problem. Now, by Lemma 6, the partition $\mathcal{P} = (Y_0 \setminus I) \uplus Y_1 \uplus \cdots \uplus Y_\ell$, where $\{Y_1, \ldots, Y_\ell\}$ is the set of clusters of $G_I - Y_0$, is an $(O(k), r + 2t_{\mathcal{F}})$-protrusion decomposition of G_I.

By Proposition 2, we can focus on solving DISJOINT PLANAR-$\mathcal{F}$-DELETION in single-exponential time when a linear protrusion decomposition is given. To that aim, we define an equivalence relation on subsets of vertices of each restricted protrusion Y_i. The key observation is that each of these equivalence relations

defines *finitely* many equivalence classes such that any partial solution lying on Y_i can be replaced with one of the representatives while preserving the feasibility. This basically follows from the *finite index* of MSO-definable properties (see, e.g., [5]). Then, we use a *decomposability* property of the solution, namely, that there always exists a solution which is formed by the union of one representative per restricted protrusion. Finally, in order to make the algorithm fully *constructive* and *uniform* on the family $\mathcal{F}$, we use classic arguments from tree automaton theory, such as the *method of test sets*. All details can be found in the full version.

Proposition 4. [$\star$] *Let (G, Y_0, k) be an instance of* Disjoint Planar-$\mathcal{F}$-Deletion *and let $\mathcal{P} = Y_0 \uplus Y_1 \uplus \cdots \uplus Y_\ell$ be an (α, β)-protrusion decomposition of G, for some constant β. There exists an $O(2^\ell \cdot n)$-time algorithm which computes a solution $\tilde{X} \subseteq V(G) \setminus Y_0$ of size at most k if it exists, or correctly decides that there is no such solution.*

We finally have all the ingredients to piece everything together.

Proof (of Theorem II). Lemma 4 states that Planar-$\mathcal{F}$-Deletion can be reduced to Disjoint Planar-$\mathcal{F}$-Deletion so that the former is single-exponential time solvable provided that the latter is, and the degree of the polynomial function in n increases by one. We now proceed to solve Disjoint Planar-$\mathcal{F}$-Deletion in time $2^{O(k)} \cdot n$. Given an instance (G, X, k) of Disjoint Planar-$\mathcal{F}$-Deletion, we apply Proposition 2 to either correctly decide that (G, X, k) is a No-instance, or identify in time $2^{O(k)} \cdot n$ a set $I \subseteq V(G)$ of size at most k and a $(O(k), 2t_{\mathcal{F}} + r)$-protrusion decomposition $\mathcal{P} = Y_0 \uplus Y_1 \uplus \cdots \uplus Y_\ell$ of $G - I$, with $X \subseteq Y_0$, such that there exists a set $X' \subseteq V(G) \setminus Y_0$ of size at most $k - |I|$ such that $G - \tilde{X}$, with $\tilde{X} = X' \cup I$, is H-minor-free for every graph $H \in \mathcal{F}$. Finally, using Proposition 4 we can solve the instance $(G_I, Y_0 \setminus I, k - |I|)$ in time $2^{O(k)} \cdot n$.

Acknowledgements. We thank D. M. Thilikos, B. Courcelle, D. Lokshtanov, and S. Saurabh for interesting discussions and helpful remarks on the manuscript.

References

1. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for Dominating Set. Journal of the ACM 51, 363–384 (2004)
2. Bodlaender, H.L.: Dynamic programming on graphs with bounded treewidth. In: Lepistö, T., Salomaa, A. (eds.) ICALP 1988. LNCS, vol. 317, pp. 105–118. Springer, Heidelberg (1988)
3. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)
4. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (Meta) Kernelization. In: Proc. of 50th FOCS, pp. 629–638. IEEE Computer Society (2009)
5. Bodlaender, H.L., van Antwerpen-de Fluiter, B.: Reduction algorithms for graphs of small treewidth. Information and Computation 167(2), 86–119 (2001)

6. Chen, J., Fomin, F.V., Liu, Y., Lu, S., Villanger, Y.: Improved algorithms for feedback vertex set problems. Journal of Computer and System Sciences 74(7), 1188–1198 (2008)
7. Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.O.: An improved FPT algorithm and quadratic kernel for pathwidth one vertex deletion. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 95–106. Springer, Heidelberg (2010)
8. Dehne, F., Fellows, M., Langston, M.A., Rosamond, F., Stevens, K.: An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 859–869. Springer, Heidelberg (2005)
9. Diestel, R.: Graph Theory, 4th edn. Springer, Heidelberg (2010)
10. Dinneen, M.: Too many minor order obstructions. Journal of Universal Computer Science 3(11), 1199–1206 (1997)
11. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
12. Fellows, M.R., Langston, M.A.: Nonconstructive tools for proving polynomial-time decidability. Journal of the ACM 35, 727–739 (1988)
13. Fomin, F.V., Lokshtanov, D., Misra, N., Philip, G., Saurabh, S.: Hitting forbidden minors: Approximation and kernelization. In: STACS 28th. LIPIcs, vol. 9, pp. 189–200. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2011)
14. Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Planar $\mathcal{F}$-Deletion: Approximation and Optimal FPT Algorithms. In: Proc. of 53rd FOCS, pp. 470–479. IEEE Computer Society (2012)
15. Fomin, F.V., Lokshtanov, D., Saurabh, S., Thilikos, D.M.: Bidimensionality and kernels. In: Proc. of 21st SODA, pp. 503–510. SIAM (2010)
16. Fomin, F.V., Oum, S., Thilikos, D.M.: Rank-width and tree-width of H-minor-free graphs. European Journal of Combinatorics 31(7), 1617–1628 (2010)
17. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. Journal of Computer and System Sciences 72(8), 1386–1396 (2006)
18. Guo, J., Niedermeier, R.: Linear problem kernels for NP-hard problems on planar graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 375–386. Springer, Heidelberg (2007)
19. Joret, G., Paul, C., Sau, I., Saurabh, S., Thomassé, S.: Hitting and harvesting pumpkins. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 394–407. Springer, Heidelberg (2011)
20. Kim, E.J., Paul, C., Philip, G.: A single-exponential FPT-algorithm for K_4-minor cover problem. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 119–130. Springer, Heidelberg (2012)
21. Lokshtanov, D., Saurabh, S., Sikdar, S.: Simpler parameterized algorithm for OCT. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 380–384. Springer, Heidelberg (2009)
22. Philip, G., Raman, V., Villanger, Y.: A quartic kernel for Pathwidth-One Vertex Deletion. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 196–207. Springer, Heidelberg (2010)
23. Robertson, N., Seymour, P.D.: Graph minors II. Algorithmic aspects of tree-width. Journal of Algorithms 7, 309–322 (1986)
24. Robertson, N., Seymour, P.D.: Graph minors XIII. The disjoint paths problem. Journal of Combinatorial Theory, Series B 63, 65–110 (1995)
25. Thomason, A.: The extremal function for complete minors. Journal of Combinatorial Theory, Series B 81(2), 318–338 (2001)

The Power of Linear Programming for Finite-Valued CSPs: A Constructive Characterization

Vladimir Kolmogorov

IST Austria (Institute of Science and Technology)
vnk@ist.ac.at

Abstract. A class of valued constraint satisfaction problems (VCSPs) is characterised by a valued constraint language, a fixed set of cost functions on a finite domain. An instance of the problem is specified by a sum of cost functions from the language with the goal to minimise the sum.

We study which classes of finite-valued languages can be solved exactly by the *basic linear programming relaxation* (BLP). Thapper and Živný showed [20] that if BLP solves the language then the language admits a *binary commutative fractional polymorphism*. We prove that the converse is also true. This leads to a necessary and a sufficient condition which can be checked in polynomial time for a given language. In contrast, the previous necessary and sufficient condition due to [20] involved infinitely many inequalities.

More recently, Thapper and Živný [21] showed (using, in particular, a technique introduced in this paper) that core languages that do not satisfy our condition are NP-hard. Taken together, these results imply that a finite-valued language can either be solved using Linear Programming or is NP-hard.

1 Introduction

We consider a particular linear programming relaxation of a class of optimization problems called in this paper *basic LP* (BLP). This relaxation has been studied extensively in various domains, especially for objective functions with unary and binary terms. Researchers analyzed its properties, developed efficient algorithms for (approximately) solving this LP, and applied to large-scale practical problems [18,14,22,10,23,5,19,1,11,17].

It has been long recognized that for some classes of optimization problems (e.g. for submodular functions on a chain) BLP relaxation is guaranteed to be tight (i.e. the integrality gap is zero), and allows to solve the problem exactly. A natural question is whether it is possible to characterize such classes.

One possible way to pose the problem formally is to use the framework of *Valued Constraint Satisfaction Problems (VCSPs)* [4]. In this framework a class of allowed objective functions is specified by a *language* Γ, which is a collection of cost functions over a fixed domain D. We say that BLP *solves* $VCSP(\Gamma)$ if the

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 625–636, 2013.

relaxation is tight for all functions that can be expressed as a sum of functions from Γ with overlapping sets of variables.

A major step in the characterization of languages that can be solved by BLP has been recently made by Thapper and Živný [20]. They gave a sufficient condition for BLP to solve $VCSP(\Gamma)$ that covers many known languages such as (1) submodular functions on arbitrary lattices; (2) k-submodular functions; (3) weakly (and hence strongly) tree-submodular functions on arbitrary trees.

Thapper and Živný also presented a necessary and sufficient condition for BLP to solve $VCSP(\Gamma)$. However, their characterization has one drawback: it involves infinitely many inequalities, which leaves an open question whether checking the condition is a decidable problem for a given finite language Γ.

We resolve this question affirmatively for finite-valued languages Γ. As our **main contribution**, we show that BLP solves such Γ iff Γ admits a *fractional symmetric polymorphism of arity k* for some $k \geq 2$. We prove this result using a mixture of algebraic tools and techniques from Linear Programming.

Very recently, Thapper and Živný [21] showed (using, in particular, a technique introduced in this paper) that a core language that does not satisfy our condition is NP-hard. It follows from [20,21] and from our results that a finite-valued language can either be solved by a Linear Programming or is NP-hard.

Related Work. Kun et al. [15] studied the BLP relaxation for CSP problems and for robust approximatility of Max-CSPs. They showed, in particular, that BLP robustly decides a CSP language iff it has *width 1*. Width-1 CSPs were introduced in [7]. A simple characterization of such CSPs was given in [6].

Our work heavily exploits the notion of *fractional polymorphisms* [2]. Fractional polymorphisms is a generalization of *multimorphisms* [4]. It is known that they can characterize all tractable VCSPs [2].

We also mention the work of Raghavendra on analyzing SDP relaxations [16]. Under the assumption of the *unique games conjecture* [9], it was shown that the basic SDP relaxation solves all tractable finite-valued VCSPs (without a characterization of the tractable cases). Furthermore, results in Chapters 6 and 7 of [16] imply that the basic SDP relaxation solves languages that admit a *cyclic fractional polymorphism* of some arity $m \geq 2$. If was not clear whether the SDP relaxation can solve exactly more languages compared to the BLP relaxation. Results in [20,21] and our results imply that this is not the case (assuming that $P \neq NP$).

2 Background and Statement of the Results

Let D be a finite domain. A *finite-valued language* Γ is a set of cost functions $f : D^n \to \mathbb{Q}$ where arity $n \geq 1$ may be different for different functions $f \in \Gamma$. The argument of f is called a *labeling*.

Definition 1. *An instance $\mathcal{I}$ of the valued constraint satisfaction problem (VCSP) is a function $D^V \to \mathbb{Q}$ given by*

$$Cost_{\mathcal{I}}(x) = \sum_{t \in T} f_t(x_{v(t,1)}, \ldots, x_{v(t,n_t)})$$

It is specified by a finite set of nodes V, finite set of terms T, cost functions $f_t : D^{n_t} \to \mathbb{Q}$ of arity n_t and indices $v(t,k) \in V$ for $t \in T, k = 1,\dots,n_t$. A solution to $\mathcal{I}$ is a labeling $x \in D^V$ with the minimum cost. Instance $\mathcal{I}$ is called a Γ-instance if all terms f_t belong to Γ.

The class of optimization problems consisting of all Γ-instances is referred to as $VCSP(\Gamma)$. Language Γ is called tractable if $VCSP(\Gamma')$ can be solved in polynomial time for **each** finite $\Gamma' \subseteq \Gamma$. It is called NP-hard if $VCSP(\Gamma')$ is NP-hard for **some** finite $\Gamma' \subseteq \Gamma$.

One way to tackle a VCSP instance is to formulate and solve a convex relaxation of the problem. Two examples are *basic LP relaxation* and *basic SDP relaxation*, as they are called in [20] and [16] respectively.

The basic LP relaxation will be of particular relevance to this paper. Following [20], we say that basic LP *solves* $VCSP(\Gamma)$ if for any instance $\mathcal{I}$ from $VCSP(\Gamma)$ the optimal value of the relaxation equals $\min_x Cost_{\mathcal{I}}(x)$.

We will study which languages Γ are solved by basic LP. We will do it indirectly by relying on the characterization of [20]. The formulation of basic LP will not be used, and so we omit it in order to avoid unnecessary notation. We refer interested readers to [20] for details.

Fractional Polymorphisms. We denote $\mathcal{O}^m$ to be the set of operations $g : D^m \to D$. A *fractional polymorphism* of arity m is a probability distribution ω over $\mathcal{O}^m$, i.e. a vector with components $\omega(g) \geq 0$ for $g : D^m \to D$ that sum to 1. Language Γ is said to *admit* ω if every cost function $f \in \Gamma$ of arity n satisfies

$$\sum_{g \in \mathcal{O}^m} \omega(g) f(g(x^1,\dots,x^m)) \leq f^m(x^1,\dots,x^m) \qquad \forall x^1,\dots,x^m \in D^n \quad (1)$$

where function f^m is defined via $f^m(x^1,\dots,x^m) = \frac{1}{m}(f(x^1) + \dots + f(x^m))$. We view labelings in D^n as column vectors; given m such columns, operation $g : D^m \to D$ produces a new column as shown on the right.

x^1	$\dots$	x^m	$g(x^1,\dots,x^m)$
x^1_1	$\dots$	x^m_1	$g(x^1_1,\dots,x^m_1)$
$\dots$	$\dots$	$\dots$	$\dots$
x^1_n	$\dots$	x^m_n	$g(x^1_n,\dots,x^m_n)$

Operation $g \in \mathcal{O}^m$ is called *symmetric* if it is invariant with respect to any permutation of its arguments: $g(a_1,\dots,a_m) = g(a_{\pi(1)},\dots,a_{\pi(m)})$ for any permutation $\pi : [1,m] \to [1,m]$ and any $(a_1,\dots,a_m) \in D^m$. It is called *cyclic* if $g(a_1,a_2,\dots,a_m) = (a_2,\dots,a_m,a_1)$ for any $(a_1,\dots,a_m) \in D^m$. Note, in the case $m = 2$ both definitions coincide. A fractional polymorphism ω is called symmetric (cyclic) if all operations in $supp(\omega)$ are symmetric (cyclic). As usual, $supp(\omega)$ denotes the support of distribution ω: $supp(\omega) = \{g \in \mathcal{O}^m \mid \omega(g) > 0\}$.

Generalized Fractional Polymorphisms. Let $\mathcal{O}^{m\to k}$ be the set of mappings $\mathbf{g} : D^m \to D^k$. A mapping $\mathbf{g} \in \mathcal{O}^{m\to k}$ can also be viewed as a sequence of k operations $\mathbf{g} = (g_1,\dots,g_k)$ with $g_i \in \mathcal{O}^m$. We define a *generalized fractional polymorphism of arity* $m \to k$ as a probability distribution ρ over $\mathcal{O}^{m\to k}$. We say that language Γ admits ρ if every cost function $f \in \Gamma$ of arity n satisfies

$$\sum_{\mathbf{g} \in \mathcal{O}^{m\to k}} \rho(\mathbf{g}) f^k(\mathbf{g}(x^1,\dots,x^m)) \leq f^m(x^1,\dots,x^m) \qquad \forall x^1,\dots,x^m \in D^n \quad (2)$$

Equivalently, ρ is a generalized fractional polymorphism of Γ of arity $m \to k$ iff vector

$$\omega = \sum_{\mathbf{g}=(g_1,\dots,g_k)\in\mathcal{O}^{m\to k}} \rho(\mathbf{g}) \cdot \frac{1}{k}(\chi_{g_1} + \ldots + \chi_{g_k}) \tag{3}$$

is a fractional polymorphism of Γ of arity m.

We can identify fractional polymorphisms of arity m with generalized fractional polymorphisms of arity $m \to 1$. For brevity, we will omit the word "generalized".

We always use the following convention: if $\mathbf{g} = (g_1, \dots, g_k)$ is a mapping in $\mathcal{O}^{m\to k}$ then we extend it to m labelings $x^1, \dots, x^m \in D^n$ component-wise, i.e. $[\mathbf{g}(x^1, \dots, x^m)]_v = \mathbf{g}(x^1_v, \dots, x^m_v)$ for all $v \in [1, n]$. Thus, $\mathbf{g}(x^1, \dots, x^m)$ is a sequence of k labelings $(g_1(x^1, \dots, x^m), \dots g_k(x^1, \dots, x^m))$ in D^n.

Fractional Polymorphisms and SDP/LP Relaxations. It has been shown that fractional polymorphisms can be used for characterizing languages that can be solved exactly by certain convex relaxations. For the SDP relaxation, the following is known; it is implied by results in Chapters 6 and 7 of [16].

Theorem 1 ([16]). *If Γ has a cyclic fractional polymorphism of some arity $k \geq 2$ then the basic SDP relaxation solves VCSP(Γ) in polynomial time.*

The more relevant for our paper case of the LP relaxation has been analyzed in [20]: [1]

Theorem 2 ([20]). *For a finite-valued language Γ, the basic LP relaxation solves $VCSP(\Gamma)$ (in polynomial time) iff Γ admits an m-ary symmetric fractional polymorphism for every $m \geq 2$.*

Theorem 3 ([20], Theorem 4.4). *If Γ admits a k-ary fractional polymorphism ω such that supp(ω) generates a symmetric m-ary operation[2] then Γ admits an m-ary symmetric fractional polymorphism.*

While the theorems give a necessary and sufficient condition for BLP to solve $VCSP(\Gamma)$, it was not clear whether checking this condition for a given finite language Γ is a decidable problem (we would need to consider infinitely many values of m). Our result given below resolves this question affirmatively.[3]

Theorem 4. *Suppose that a finite-valued language Γ admits a fractional polymorphism of arity $k \geq 2$ whose support contains at least one symmetric operation. Then Γ admits a symmetric fractional polymorphism of every arity $m \geq 2$ (and thus BLP solves $VCSP(\Gamma)$).*

[1] Thapper and Živný also present some results for infinite-valued languages; we refer to [20] for details.

[2] A set $\mathcal{O}$ of operations is said to generate f, if f can be obtained by composition from projections and operations in $\mathcal{O}$.

[3] To check whether Γ satisfies the condition of Theorem 4, we need to check whether there exists a probability distribution ω over $\mathcal{O}^2$ that satisfies inequality (1) for all $f \in \Gamma$ of arity n and all $x^1, x^2 \in D^n$. This problem can be casted as an LP whose size is polynomial in the size of language Γ (which is specified by $\sum_{f\in\Gamma} |D|^{\mathtt{ar}\, f}$ rational numbers, where $\mathtt{ar}\, f$ is the arity of function f).

A more recent result implies a dichotomy for finite-valued languages: every language is either tractable (via BLP) or is NP-hard.

Theorem 5 ([21]). *If a finite-valued core language Γ does not admit a symmetric fractional polymorphism of arity 2 then it is NP-hard.*[4]

STP Multimorphisms. As a minor contribution, we formally prove that if a finite-valued language admits an *STP multimorphism* then it also admits a submodularity multimorphism with respect to some total order on D. (This has implications for the complexity classification of *conservative* finite-valued languages.) This result is already known, but to our knowledge a formal proof has not been written down yet. We refer to section 4 for further discussion.

3 Proof of Theorem 4

We will prove the following result.

Theorem 6. *Suppose that a finite-valued language Γ over a domain D admits a symmetric fractional polymorphism of arity $m-1 \geq 2$. Then Γ admits a symmetric fractional polymorphism of arity m.*

This will imply Theorem 4. Indeed, suppose that Γ admits a fractional polymorphism of arity $k \geq 2$ which contains a symmetric operation. Induction on m yields that Γ admits an m-ary symmetric fractional polymorphism for every $m \geq k$ (we need to use Theorem 3 for the base case and Theorem 6 for the induction step). This also implies the claim for all $m \in [2, k-1]$: it is straightforward to show that if Γ admits a symmetric fractional polymorphism of arity pm where $p, m \in \mathbb{N}$ then it also admits a symmetric fractional polymorphism of arity m.

We thus concentrate on proving Theorem 6. From now on we fix a symmetric fractional polymorphism ω of Γ of arity $m-1$.

Consider a permutation π of the set $[1,m] \triangleq \{1,\dots,m\}$ and a symmetric operation $s \in supp(\omega)$ of arity $m-1$. For such π and s we introduce the following definitions.

- For a labeling $\alpha = (a_1,\dots,a_m) \in D^m$, let $\alpha^\pi \in D^m$ and $\alpha^s \in D^m$ be the following labelings:

$$\alpha^\pi = (a_{\pi(1)},\dots,a_{\pi(m)}) \tag{4}$$
$$\alpha^s = (s(\alpha_{-1}),\dots,s(\alpha_{-m})) \tag{5}$$

  where $\alpha_{-i} \in D^{m-1}$ is the labeling obtained from α by removing the i-th element.

[4] We refer to [21] for the definition of a core. (It differs from the definition in [8], but [21] shows that the two definitions are equivalent.) The coreness assumption is not a severe restriction: if Γ is not a core then there is a polynomial-time reduction between $VCSP(\Gamma)$ and $VCSP(\Gamma')$ for some core language Γ' on a smaller domain. It is also not difficult to show that Theorem 5 holds for non-core languages as well, using induction on the size of the domain D.

– For an operation $g : D^m \to D$, let $g^\pi : D^m \to D$ be the following operation:

$$g^\pi(\alpha) = g(\alpha^\pi) \tag{6}$$

– For a mapping $\mathbf{g} : D^m \to D^m$, let $\mathbf{g}^s : D^m \to D^m$ be the following mapping:

$$\mathbf{g}^s(\alpha) = [\mathbf{g}(\alpha)]^s \tag{7}$$

The last definition can also be expressed as

$$\mathbf{g}^s = (s \circ \mathbf{g}_{-1}, \dots, s \circ \mathbf{g}_{-m}) \tag{8}$$

where $\mathbf{g}_{-i} : D^m \to D^{m-1}$ is the sequence of $m-1$ operations obtained from $\mathbf{g} = (g_1, \dots, g_m)$ by removing the i-th operation.

Let $\mathbb{1}$ be the identity mapping $D^m \to D^m$, and let $V = \{\mathbb{1}^{s_1 \dots s_k} \mid s_1, \dots, s_k \in supp(\omega), k \geq 0\}$ be the set of all mappings that can be obtained from $\mathbb{1}$.

Proposition 7. *Every* $\mathbf{g} = (g_1, \dots, g_m) \in V$ *satisfies the following:*

$$(g_1^\pi, \dots, g_m^\pi) = (g_{\pi(1)}, \dots, g_{\pi(m)}) \qquad \forall \text{ permutation } \pi \tag{9}$$

Thus, permuting the arguments of $g_i(\cdot, \dots, \cdot)$ gives a mapping which is also present in the sequence $\mathbf{g}$, possibly at a different position.

Proof. Checking that $\mathbb{1}$ satisfies (9) is straightforward. Let us prove that for any $\mathbf{g} : D^m \to D^m$ satisfying (9) and for any symmetric operation $s \in \mathcal{O}^{m-1}$ mapping $\mathbf{g}^s$ also satisfies (9). Consider $i \in [1, m]$. We need to show that $(s \circ \mathbf{g}_{-i})^\pi = s \circ \mathbf{g}_{-\pi(i)}$. For each $\alpha \in D^m$ we have

$$\begin{aligned}
(s \circ \mathbf{g}_{-i})^\pi(\alpha) &= s \circ \mathbf{g}_{-i}(\alpha^\pi) \\
&= s(g_1(\alpha^\pi), \dots, g_{i-1}(\alpha^\pi), g_{i+1}(\alpha^\pi), \dots, g_m(\alpha^\pi)) \\
&= s(g_1^\pi(\alpha), \dots, g_{i-1}^\pi(\alpha), g_{i+1}^\pi(\alpha), \dots, g_m^\pi(\alpha)) \\
&= s(g_{\pi(1)}(\alpha), \dots, g_{\pi(i-1)}(\alpha), g_{\pi(i+1)}(\alpha), \dots, g_{\pi(m)}(\alpha)) = s \circ \mathbf{g}_{-\pi(i)}(\alpha)
\end{aligned}$$

□

Graph on Mappings. Let us define a directed weighted graph $G = (V, E, w)$ with the set of edges $E = \{(\mathbf{g}, \mathbf{g}^s) \mid \mathbf{g} \in V, s \in supp(\omega)\}$ and positive weights $w(\mathbf{g}, \mathbf{h}) = \sum_{s \in supp(\omega) : \mathbf{h} = \mathbf{g}^s} \omega(s)$ for $(\mathbf{g}, \mathbf{h}) \in E$. Clearly, we have

$$\sum_{\mathbf{h} : (\mathbf{g}, \mathbf{h}) \in E} w(\mathbf{g}, \mathbf{h}) = 1 \qquad \forall \mathbf{g} \in V \tag{10}$$

We define $\mathbb{H}[G]$ to be the set of strongly connected components $H \subseteq V$ of G which are sinks, i.e. all edges in G from H lead to vertices in H. We also denote $\widehat{H} = \bigcup_{H \in \mathbb{H}[G]} H \subseteq V$.

3.1 Proof Overview: Main Theorems

From now on we fix a function $f \in \Gamma$ of arity n. Recall that we defined $f^m(x^1, \dots, x^m) = \frac{1}{m} \sum_{i \in [1,m]} f(x^i)$. For a mapping $\mathbf{g} \in \mathcal{O}^{m \to m}$ we define

$$Range_n(\mathbf{g}) = \{\mathbf{g}(x^1, \dots, x^m) \mid x^1, \dots, x^m \in D^n\}$$

We will prove the following.

Theorem 8. *There exists a fractional polymorphism ρ of Γ of arity $m \to m$ with $supp(\rho) \subseteq \widehat{H}$.*

Theorem 9. *Let $\hat{\mathbf{g}}$ be a mapping in $\widehat{H}$ and $\mathbf{p} \in \mathcal{O}^{m \to m}$ be any mapping such that $\mathbf{p}(\alpha)$ is a permutation of α for all $\alpha \in D^m$. For any $(x^1, \dots, x^m) \in Range_n(\hat{\mathbf{g}})$ there holds $f^m(x^1, \dots, x^m) = f^m(\mathbf{p}(x^1, \dots, x^m))$.*

This will imply Theorem 6. Indeed, we can construct an m-ary symmetric fractional polymorphism of Γ as follows. Take vector ρ from Theorem 8, take a symmetric mapping $\mathbf{p} \in \mathcal{O}^{m \to m}$ satisfying the condition of Theorem 9, and define a fractional polymorphism of arity $m \to m$

$$\rho' = \sum_{\mathbf{g} \in supp(\rho)} \rho(\mathbf{g}) \chi_{\mathbf{p} \circ \mathbf{g}}$$

Function f admits ρ' since for any labelings $x^1, \dots, x^m \in D^n$ there holds

$$\begin{aligned} \sum_{\mathbf{h} \in supp(\rho')} \rho'(\mathbf{h}) f^m(\mathbf{h}(x^1, \dots, x^m)) &= \sum_{\mathbf{g} \in supp(\rho)} \rho(\mathbf{g}) f^m(\mathbf{p}(\mathbf{g}(x^1, \dots, x^m))) \\ &= \sum_{\mathbf{g} \in supp(\rho)} \rho(\mathbf{g}) f^m(\mathbf{g}(x^1, \dots, x^m)) \le f^m(x^1, \dots, x^m) \end{aligned}$$

Note, for any $\mathbf{h} = (h_1, \dots, h_m) \in supp(\rho')$ operations $h_1, \dots, h_m$ are symmetric. Indeed, we have $\mathbf{h} = \mathbf{p} \circ \mathbf{g}$ for some $\mathbf{g} \in V$. If $\alpha \in D^m$ and π is a permutation of the set $[1, m]$ then $\mathbf{h}(\alpha^\pi) = \mathbf{p}(\mathbf{g}(\alpha^\pi)) = \mathbf{p}([\mathbf{g}(\alpha)]^\pi) = \mathbf{p}(\mathbf{g}(\alpha)) = \mathbf{h}(\alpha)$ which implies the claim.

We can finally apply transformation (3) to vector ρ' to get a symmetric fractional polymorphism of arity m.

A proof of Theorem 8 is given in [12]. To prove Theorem 9, we will need an auxiliary result. Let us fix a connected component $H \in \mathbb{H}[G]$, and denote $I = H \times [1, m]$. Given labelings $x^1, \dots, x^m$, we define labelings $x^{\mathbf{g}i}$ for all $(\mathbf{g}, i) \in I$ via

$$(x^{\mathbf{g}1}, \dots, x^{\mathbf{g}m}) = \mathbf{g}(x^1, \dots, x^m) \tag{11}$$

Note that $x^{\mathbf{g}i}$ is a function of $(x^1, \dots, x^m)$; for brevity of notation, this dependence is not shown. For a vector $\lambda \in \mathbb{R}^H$ and an index $i \in [1, m]$ we define function F_i^λ via

$$F_i^\lambda(x^1, \dots, x^m) = \sum_{\mathbf{g} \in H} \lambda_{\mathbf{g}} f(x^{\mathbf{g}i}) \qquad \forall x^1, \dots, x^m \in D^n \tag{12}$$

Theorem 10. *Consider $H \in \mathbb{H}[G]$.*
(a) There holds $f^m(x^{\mathbf{g}'1}, \dots, x^{\mathbf{g}'m}) = f^m(x^{\mathbf{g}''1}, \dots, x^{\mathbf{g}''m})$ for all $\mathbf{g}', \mathbf{g}'' \in H$ and $x^1, \dots, x^m \in D^n$.
(b) There exists a probability distribution λ over H such that $F^\lambda_{i'}(x^1, \dots, x^m) = F^\lambda_{i''}(x^1, \dots, x^m)$ for all $i', i'' \in [1, m]$ and $x^1, \dots, x^m \in D^n$.

We prove this theorem in section 3.2; using this result, we then prove Theorem 9 in section 3.3.

3.2 Proof of Theorem 10

First, we make the following observation.

Proposition 11. *If $\mathbf{h} = \mathbf{g}^s$ where $\mathbf{g} \in H$, $s \in supp(\omega)$ then $x^{\mathbf{h}i} = s((x^{\mathbf{g}1}, \dots, x^{\mathbf{g}m})_{-i})$ for $i \in [1, m]$ where $(x^{\mathbf{g}1}, \dots, x^{\mathbf{g}m})_{-i}$ is the sequence of $m-1$ labelings obtained by removing the i-th labeling.*

Proof. Consider node $v \in [1, n]$, and denote $\alpha = (x^1_v, \dots, x^m_v)$, $\beta = (x^{\mathbf{g}1}_v, \dots, x^{\mathbf{g}m}_v)$, $\gamma = (x^{\mathbf{h}1}_v, \dots, x^{\mathbf{h}m}_v)$. By definition (11), $\beta = \mathbf{g}(\alpha)$ and $\gamma = \mathbf{h}(\alpha)$. Therefore, $\gamma = \mathbf{g}^s(\alpha) = [\mathbf{g}(\alpha)]^s = \beta^s$. In other words, the i-th component of γ equals $s(\beta_{-i})$, which is what we needed to show.

□

We will show that for fixed distinct mappings $\mathbf{g}', \mathbf{g}'' \in H$ there holds

$$\sum_{i \in [1,m]} f(x^{\mathbf{g}'i}) - \sum_{i \in [1,m]} f(x^{\mathbf{g}''i}) \le 0 \tag{13a}$$

and that there exists a probability distribution λ over H such that for fixed distinct indices $i', i'' \in [1, m]$ there holds

$$\sum_{\mathbf{g} \in H} \lambda_{\mathbf{g}} f(x^{\mathbf{g}i'}) - \sum_{\mathbf{g} \in H} \lambda_{\mathbf{g}} f(x^{\mathbf{g}i''}) \le 0 \tag{13b}$$

Clearly, this will imply Theorem 10.

To prove these facts, we will use the following strategy. For each $(\mathbf{g}, i) \in I$ let us write the polymorphism inequality for labelings $(x^{\mathbf{g}1}, \dots, x^{\mathbf{g}m})_{-i}$:

$$\sum_{s \in supp(\omega)} \omega(s) f(s((x^{\mathbf{g}1}, \dots, x^{\mathbf{g}m})_{-i})) \le \frac{1}{m-1} \sum_{j \in [1,m]-\{i\}} f(x^{\mathbf{g}j})$$

Let us multiply this inequality by weight $\lambda_{\mathbf{g}i} \ge 0$ (to be defined later), and apply Proposition 11 and the fact that $w(\mathbf{g}, \mathbf{h}) = \sum_{s \in supp(\omega): \mathbf{h} = \mathbf{g}^s} \omega(s)$:

$$\lambda_{\mathbf{g}i} \sum_{\mathbf{h}:(\mathbf{g},\mathbf{h}) \in E} w(\mathbf{g}, \mathbf{h}) f(x^{\mathbf{h}i}) - \frac{\lambda_{\mathbf{g}i}}{m-1} \sum_{j \in [1,m]-\{i\}} f(x^{\mathbf{g}j}) \le 0 \qquad \forall (\mathbf{g}, i) \in I$$

Summing these inequalities over $(\mathbf{g}, i) \in I$ gives

$$\sum_{(\mathbf{g},i) \in I} \left[\sum_{\mathbf{h}:(\mathbf{h},\mathbf{g}) \in E} w(\mathbf{h}, \mathbf{g}) \lambda_{\mathbf{h}i} - \sum_{j \in [1,m]-\{i\}} \frac{\lambda_{\mathbf{g}j}}{m-1} \right] f(x^{\mathbf{g}i}) \le 0 \tag{14}$$

Parts (a,b) of Lemma 12 together with Remark 1 below show that coefficients $\lambda_{\mathbf{g}i}$ can be chosen in such a way that the last inequality becomes equivalent to (13a) and (13b) respectively, thus proving Theorem 10.

Lemma 12. *(a) There exists vector $\lambda \in \mathbb{R}^I_{\geq 0}$ that satisfies*

$$\sum_{\mathbf{h}:(\mathbf{h},\mathbf{g})\in E} w(\mathbf{h},\mathbf{g})\lambda_{\mathbf{h}i} - \sum_{j\in[1,m]-\{i\}} \frac{\lambda_{\mathbf{g}j}}{m-1} = c_{\mathbf{g}} \qquad \forall(\mathbf{g},i)\in I \tag{15}$$

where $c_{\mathbf{g}} = [\mathbf{g}=\mathbf{g}'] - [\mathbf{g}=\mathbf{g}'']$ and $[\cdot]$ is the Iverson bracket: it equals 1 if the argument is true, and 0 otherwise.
(b) There exists vector $\lambda \in \mathbb{R}^{I\cup H}_{\geq 0}$ that satisfies

$$\sum_{\mathbf{h}:(\mathbf{h},\mathbf{g})\in E} w(\mathbf{h},\mathbf{g})\lambda_{\mathbf{h}i} - \sum_{j\in[1,m]-\{i\}} \frac{\lambda_{\mathbf{g}j}}{m-1} = c_i\lambda_{\mathbf{g}} \qquad \forall(\mathbf{g},i)\in I \tag{16a}$$

$$\sum_{\mathbf{g}\in H} \lambda_{\mathbf{g}} = 1 \tag{16b}$$

where $c_i = [i=i'] - [i=i'']$.

Remark 1. Note that vector λ in part (b) depends on the pair (i',i''); let us denote it as $\lambda^{i'i''}$. To establish Theorem 10(b), it suffices to show that vectors $\lambda^{i'i''}$ in Lemma 12(b) can be chosen in such a way that for a given $\mathbf{g}\in H$ components $\lambda^{i'i''}_{\mathbf{g}}$ are the same for all pairs (i',i''). This can be done as follows. Take vector λ^{12} constructed in Lemma 12(b). For a pair of distinct indices $(i',i'') \neq (1,2)$ select permutation π of $[1,m]$ with $\pi(i')=1$, $\pi(i'')=2$, and define vector $\lambda^{i'i''}$ via

$$\lambda^{i'i''}_{\mathbf{g}} = \lambda^{12}_{\mathbf{g}} \qquad \forall \mathbf{g}\in H \qquad\qquad \lambda^{i'i''}_{\mathbf{g}i} = \lambda^{12}_{\mathbf{g}\pi(i)} \qquad \forall(\mathbf{g},i)\in I$$

Clearly, vector $\lambda^{i'i''}$ satisfies conditions of Lemma 12(b) for the pair (i',i''). Thus, Lemma 12 indeed implies Theorem 10. The proof of part (a) of this lemma is given below; the proof of part (b) is based on the same idea, and is given in [12].

Proof. **Part (a)** Suppose the claim does not hold. By Farkas's lemma there exists vector $y\in\mathbb{R}^I$ such that

$$\sum_{(\mathbf{g},i)\in I} c_{\mathbf{g}} y_{\mathbf{g}i} < 0 \tag{17a}$$

$$\sum_{\mathbf{h}:(\mathbf{g},\mathbf{h})\in E} w(\mathbf{g},\mathbf{h})y_{\mathbf{h}i} - \sum_{j\in[1,m]-\{i\}} \frac{y_{\mathbf{g}j}}{m-1} \geq 0 \qquad \forall(\mathbf{g},i)\in I \tag{17b}$$

Denote $u_{\mathbf{g}} = \sum_{i\in[1,m]} y_{\mathbf{g}i}$. Summing inequalities (17b) over $i\in[1,m]$ gives

$$\sum_{\mathbf{h}:(\mathbf{g},\mathbf{h})\in E} w(\mathbf{g},\mathbf{h})u_{\mathbf{h}} - u_{\mathbf{g}} \geq 0 \qquad \forall \mathbf{g}\in H \tag{18}$$

Denote $H^* = \arg\max\{u_{\mathbf{g}} \,|\, \mathbf{g} \in H\}$. From (10) and (18) we conclude that $\mathbf{g} \in H^*$ implies $\mathbf{h} \in H^*$ for all $(\mathbf{g}, \mathbf{h}) \in E$. Therefore, $H^* = H$ (since H is a strongly connected component of G).

We showed that $u_{\mathbf{g}} = C$ for all $\mathbf{g} \in H$ where $C \in \mathbb{R}$ is some constant. But then the expression on the LHS of (17a) equals $C - C = 0$ - a contradiction.

□

3.3 Proof of Theorem 9

Let $H \in \mathbb{H}[G]$ be the strongly connected component that contains $\hat{\mathbf{g}}$, and let $\lambda \in \mathbb{R}^H_{\geq 0}$ be a vector constructed in Theorem 10(b). We denote $F^{\lambda}(x^1, \dots, x^m) = F^{\lambda}_i(x^1, \dots, x^m)$ for $i \in [1, m]$.

Lemma 13. *The following transformation does not change $F^{\lambda}(x^1, \dots, x^m)$: pick node $v \in [1, n]$ and permute values $(x^1_v, \dots, x^m_v)$.*

Proof. It suffices to prove the claim for a permutation which swaps values x^i_v and x^j_v for $i, j \in [1, m]$ (since any other permutation can be obtained by repeatedly applying such swaps). Since $m \geq 3$ there exists index $k \in [1, m] - \{i, j\}$. Using Proposition 7, it can be checked that the swap above does not affect labelings $x^{\mathbf{g}k}$ in (11) for $\mathbf{g} \in H$, and therefore $F^{\lambda}_k(x^1, \dots, x^m)$ does not change.

□

Lemma 14. *If $(x^1, \dots, x^m) \in Range_n(\hat{\mathbf{g}})$ then $(x^1, \dots, x^m) = (x^{\mathbf{g}1}, \dots, x^{\mathbf{g}m})$ for some $\mathbf{g} \in H$.*

Proof. It suffices to show that there exists $\mathbf{g} \in H$ with $\mathbf{g} \circ \hat{\mathbf{g}} = \hat{\mathbf{g}}$. We refer to [12] for the proof of this fact.

□

Lemma 15. *If $(x^1, \dots, x^m) \in Range_n(\hat{\mathbf{g}})$ then $f^m(x^1, \dots, x^m) = F^{\lambda}(x^1, \dots, x^m)$.*

Proof. From Theorem 10(a) and Lemma 14 we get that $f^m(x^{\mathbf{g}1}, \dots, x^{\mathbf{g}m}) = f^m(x^1, \dots, x^m)$ for all $\mathbf{g} \in H$. Using this fact and the definition of $F^{\lambda}_i(\cdot)$, we can write

$$F^{\lambda}(x^1, \dots, x^m) = \frac{1}{m} \sum_{i \in [1,m]} F^{\lambda}_i(x^1, \dots, x^m) = \frac{1}{m} \sum_{\mathbf{g} \in H} \lambda_{\mathbf{g}} \sum_{i \in [1,m]} f(x^{\mathbf{g}i})$$
$$= \sum_{\mathbf{g} \in H} \lambda_{\mathbf{g}} f^m(x^{\mathbf{g}1}, \dots, x^{\mathbf{g}m}) = \sum_{\mathbf{g} \in H} \lambda_{\mathbf{g}} f^m(x^1, \dots, x^m) = f^m(x^1, \dots, x^m)$$

□

To establish Theorem 9, it remains to prove that condition $(x^1, \dots, x^m) \in Range_n(\hat{\mathbf{g}})$ implies $\mathbf{p}(x^1, \dots, x^m) \in Range_n(\hat{\mathbf{g}})$. This proof follows mechanically from Proposition 7, and is omitted.

4 STP Multimorphisms

In this section we consider Symmetric Tournament Pairs (STP) multimorphisms [3].

Definition 2. *(a) A pair of operations* $\langle \sqcap, \sqcup \rangle$ *with* $\sqcap, \sqcup : D \times D \to D$ *is called an STP if*

$$a \sqcap b = b \sqcap a, \quad a \sqcup b = b \sqcup a \qquad \forall a, b \in D \qquad \textit{(commutativity)} \tag{19a}$$

$$\{a \sqcap b, a \sqcup b\} = \{a, b\} \qquad \forall a, b \in D \qquad \textit{(conservativity)} \tag{19b}$$

(b) Pair $\langle \sqcap, \sqcup \rangle$ *is called a* submodularity operation *if there exists a total order on* D *for which* $a \sqcap b = \min\{a, b\}$, $a \sqcup b = \max\{a, b\}$ *for all* $a, b \in D$.
(c) Language Γ *admits* $\langle \sqcap, \sqcup \rangle$ *(or* $\langle \sqcap, \sqcup \rangle$ *is a multimorphism of* Γ*) if every function* $f \in \Gamma$ *of arity* n *satisfies*

$$f(x \sqcap y) + f(x \sqcup y) \leq f(x) + f(y) \qquad \forall x, y \in D^n \tag{20}$$

It has been shown in [3] that if Γ admits an STP multimorphism then $VCSP(\Gamma)$ can be solved in polynomial time. STP multimorphisms also appeared in the dichotomy result of [13]:

Theorem 16. *Suppose a finite-valued language* Γ *is* conservative*, i.e. it contains all possible unary cost functions* $u : D \to \{0, 1\}$*. Then* Γ *either admits an STP multimorphism or it is NP-hard.*

In this paper we prove the following (see [12]).

Theorem 17. *If a finite-valued language* Γ *admits an STP multimorphism then it also admits a submodularity multimorphism.*

This fact is already known; in particular, footnote 2 in [13] mentions that this result is implicitly contained in [3], and sketches a proof strategy. However, to our knowledge a formal proof has never appeared in the literature. This paper fills this gap. Our proof is different from the one suggested in [13], and inspired some of the proof techniques used in the main part of this paper.

Acknowledgements. I thank Andrei Krokhin for helpful discussions and for communicating the result of Raghavendra [16] about cyclic fractional polymorphisms.

References

1. Blake, A., Kohli, P., Rother, C. (eds.): Advances in Markov Random Fields for Vision and Image Processing. MIT Press (2011)
2. Cohen, D.A., Cooper, M.C., Jeavons, P.G.: An algebraic characterisation of complexity for valued constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 107–121. Springer, Heidelberg (2006)
3. Cohen, D., Cooper, M., Jeavons, P.: Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. Theoretical Computer Science 401(1), 36–51 (2008)
4. Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: The complexity of soft constraint satisfaction. Artificial Intelligence 170(11), 983–1016 (2006)

5. Cooper, M.C., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artif. Intell. 174(7-8), 449–478 (2010)
6. Dalmau, V., Pearson, J.: Closure functions and width 1 problems. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 159–173. Springer, Heidelberg (1999)
7. Feder, T., Vardi, M.: The computational structure of monotone monadic SNP a and constraint satisfaction: A study through Datalog and group theory. SIAM Journal on Computing 28(1), 57–104 (1998)
8. Huber, A., Krokhin, A., Powell, R.: Skew bisubmodularity and valued CSPs. In: SODA (2013)
9. Khot, S.: On the unique games conjecture (invited survey). In: Proceedings of the 25th Annual IEEE Conference on Computational Complexity (CCC 2010), pp. 99–121 (2010)
10. Kolmogorov, V.: Convergent tree-reweighted messages passing. PAMI 28(10), 1568–1583 (2006)
11. Kolmogorov, V., Schoenemann, T.: Generalized sequential tree-reweighted message passing. CoRR, abs/1205.6352 (2012)
12. Kolmogorov, V.: The power of linear programming for valued CSPs: a constructive characterization. ArXiv, abs/1207.7213v4 (2012)
13. Kolmogorov, V., Živný, S.: The complexity of conservative valued CSPs. In: SODA (2012)
14. Koster, A., van Hoesel, C.P.M., Kolen, A.W.J.: The partial constraint satisfaction problem: Facets and lifting theorems. Operation Research Letters 23(3-5), 89–97 (1998)
15. Kun, G., O'Donnell, R., Tamaki, S., Yoshida, Y., Zhou, Y.: Linear programming, width-1 CSPs, and robust satisfaction. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS 2012, pp. 484–495 (2012)
16. Raghavendra, P.: Approximating NP-hard Problems: Efficient Algorithms and their Limits. PhD Thesis (2009)
17. Savchynskyy, B., Schmidt, S., Kappes, J.H., Schnörr, C.: Efficient MRF energy minimization via adaptive diminishing smoothing. In: UAI (2012)
18. Schlesinger, M.I.: Syntactic analysis of two-dimensional visual signals in noisy conditions. Kibernetika 4, 113–130 (1976) (in Russian)
19. Sontag, D., Globerson, A., Jaakkola, T.: Introduction to dual decomposition for inference. In: Sra, S., Nowozin, S., Wright, S.J. (eds.) Optimization for Machine Learning. MIT Press (2011)
20. Thapper, J., Živný, S.: The power of linear programming for valued CSPs. In: FOCS (2012)
21. Thapper, J., Živný, S.: The complexity of finite-valued CSPs. ArXiv, abs/1210.2987 (2012); To appear in STOC 2013
22. Wainwright, M.J., Jaakkola, T.S., Willsky, A.S.: MAP estimation via agreement on (hyper)trees: Message-passing and linear-programming approaches. IEEE Transactions on Information Theory 51(11), 3697–3717 (2005)
23. Werner, T.: A linear programming approach to max-sum problem: A review. PAMI 29(7), 1165–1179 (2007)

Approximating Semi-matchings in Streaming and in Two-Party Communication*

Christian Konrad[1] and Adi Rosén[2]

[1] LIAFA, Université Paris Diderot - Paris 7, France
konrad@lri.fr
[2] CNRS and Univerité Paris Diderot - Paris 7, France
adiro@liafa.univ-paris-diderot.fr

Abstract. We study the communication complexity and streaming complexity of approximating unweighted semi-matchings. A semi-matching in a bipartite graph $G = (A, B, E)$, with $n = |A|$, is a subset of edges $S \subseteq E$ that matches all A vertices to B vertices with the goal usually being to do this as fairly as possible. While the term *semi-matching* was coined in 2003 by Harvey et al. [WADS 2003], the problem had already previously been studied in the scheduling literature under different names.

We present a deterministic one-pass streaming algorithm that for any $0 \leq \epsilon \leq 1$ uses space $\tilde{\mathrm{O}}(n^{1+\epsilon})$ and computes an $\mathrm{O}(n^{(1-\epsilon)/2})$-approximation to the semi-matching problem. Furthermore, with $\mathrm{O}(\log n)$ passes it is possible to compute an $\mathrm{O}(\log n)$-approximation with space $\tilde{\mathrm{O}}(n)$.

In the one-way two-party communication setting, we show that for every $\epsilon > 0$, deterministic communication protocols for computing an $\mathrm{O}(n^{\frac{1}{(1+\epsilon)c+1}})$-approximation require a message of size more than cn bits. We present two deterministic protocols communicating n and $2n$ edges that compute an $\mathrm{O}(\sqrt{n})$ and an $\mathrm{O}(n^{1/3})$-approximation respectively.

Finally, we improve on results of Harvey et al. [Journal of Algorithms 2006] and prove new links between semi-matchings and matchings. While it was known that an optimal semi-matching contains a maximum matching, we show that there is a hierachical decomposition of an optimal semi-matching into maximum matchings. A similar result holds for semi-matchings that do not admit length-two degree-minimizing paths.

1 Introduction

Semi-matchings. A *matching* in an unweighted bipartite graph $G = (A, B, E)$ can be seen as a one-to-one assignment matching the A vertices to B vertices. The usual aim is to find a matching that leaves as few A vertices without associations as possible. A *semi-matching* is then an extension of a matching, in that it is required that *all* A vertices are matched to B vertices. This, however, is generally not possible in an injective way, and therefore we now allow the matching of multiple A vertices to the same B vertex. Typical objectives here

* Research supported in part by ANR project RDAM. A full version of this paper can be found at: http://arxiv.org/abs/1304.6906

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 637–649, 2013.

are to minimize the maximal number of A vertices that are matched to the same B vertex, or to optimize with respect to even stronger balancing constraints. The term 'semi-matching' was coined by Harvey et al. [HLLT03] and also used in [FLN10], however, the problem had already previously been intensely studied in the scheduling literature [ECS73, Hor73, ANR95, Abr03, LL04]. We stick to this term since it nicely reflects the structural property of entirely matching one bipartition of the graph.

The most prominent application of the semi-matching problem is that of assigning a set of unit-length jobs to a set of identical machines with respect to assignment conditions expressed through edges between the two sets. The objective of minimizing the maximal number of jobs that a machine receives then corresponds to minimizing the *makespan* of the scheduling problem. Optimizing the cost function $\sum_{b \in B} \deg_S(b)(\deg_S(b) + 1)/2$, where $\deg_S(b)$ denotes the number of jobs that a machine b receives in the semi-matching S, corresponds to minimizing the *total completion time* of the jobs (optimizing with respect to this cost function automatically minimizes the maximal degree).

It is well known that matchings are of maximal size if they do not admit *augmenting paths* [Ber57]. Augmenting paths for matchings correspond to *degree-minimizing paths* for semi-matchings. They first appeared in [HLLT03] under the name of *cost-reducing-paths*, and they were used for the computation of a semi-matching that minimizes a certain cost function. We use the term 'degree-minimizing-path' since it is more appropriate in our setting. A degree-minimizing path starts at a B node of high degree, then alternates between edges of the semi-matching and edges outside the semi-matching, and ends at another B node of degree at least by two smaller than the degree of the starting point of the path. Flipping the semi-matching and non-semi-matching edges of the path then generates a new semi-matching such that the large degree of the start node of the path is decreased by 1, and the small degree of the end node of the path is increased by 1. An *optimal semi-matching* is defined in [HLLT03] to be one that does not admit any degree-minimizing paths. It was shown in [HLLT03] that such a semi-matching is also optimal with respect to a large set of cost functions, including the minimization of the maximal degree as well as the minimization of the total completion time. At present, the best existing algorithm for computing an optimal semi-matching [FLN10] runs in time $O(\sqrt{|V|}|E| \log |V|)$ where $V = A \cup B$. See [FLN10] for a broader overview about previous work on semi-matchings (including works from the scheduling literature).

In this paper, we study *approximation algorithms* for the semi-matching problem in different computational settings. The notion of approximation that we consider is with respect to the maximal degree: given a bipartite graph $G = (A, B, E)$ with $n = |A|$, we are interested in computing a semi-matching S such that $\deg \max S \leq c \cdot \deg \max S^*$, where S^* denotes an optimal semi-matching, $\deg \max$ denotes the maximal degree of a vertex w.r.t. a set of edges, and c is the approximation factor. This notion of approximation corresponds to approximating the makespan when the semi-matching is seen as a scheduling problem. This setting was already studied in e.g. [ANR95].

Streaming Algorithms and Communication Complexity. *Streaming Algorithms* fall into the category of massive data set algorithms. In many applications, the data that an algorithm is called upon to process is too large to fit into the computer's memory. In order to cope with this problem, a streaming algorithm sequentially scans the input while using a random access memory of size sublinear in the length of the input stream. Multiple passes often help to further decrease the size of the random access memory. *Graph streams* are widely studied in the streaming model, and in the last years matching problems have received particular attention [AG11, GKK12, KMM12, Kap13]. A graph stream is a sequence of the edges of the input graph with a priori no assumption on the order of the edges. Particular arrival orders of the edges are studied in the literature and allow the design of algorithms that depend on that order. Besides uniform random order [KMM12], the *vertex arrival order* [GKK12, Kap13] of edges of a bipartite graph is studied where edges incident to the same A node arrive in blocks. Deciding basic graph properties such as connectivity already requires $\Omega(|V|)$ space [FKM$^+$05], where V denotes the vertex set of a graph. Many works considering graph streams allow an algorithm to use $O(|V| \operatorname{polylog} |V|)$ space. This setting is usually called the *semi-streaming* setting.

Space lower bounds for streaming algorithms are often obtained via *Communication Complexity*. There is an inherent link between streaming algorithms and one-way k-party communication protocols. A streaming algorithm for a problem P with space s also serves as a one-way k-party communication protocol for P with communication cost $O(sk)$. Conversely, a lower bound on the size of any message of such a protocol is also a lower bound on the space requirements of a streaming algorithm. Determining the communication complexity of problems is in itself an important task, however, the previously discussed link to streaming algorithms provides an additional motivation.

Our Contributions. We initiate the study of the semi-matching problem in the streaming and the communication settings. We present a deterministic one-pass streaming algorithm that for any $0 \leq \epsilon \leq 1$ uses space $\tilde{O}(n^{1+\epsilon})$ and computes an $O(n^{(1-\epsilon)/2})$-approximation to the semi-matching problem (**Theorem 1**)[1]. Furthermore, we show that with $O(\log n)$ passes we can compute an $O(\log n)$-approximation with space $\tilde{O}(n)$ (**Theorem 2**).

In the two-party one-way communication setting, we show that for any $\epsilon > 0$, deterministic communication protocols that compute an $O(n^{\frac{1}{(1+\epsilon)c+1}})$-approximation to the semi-matching problem require a message of size at least cn bits (**Theorem 5**). We present two deterministic protocols communicating n and $2n$ edges that compute an $O(\sqrt{n})$-approximation and an $O(n^{1/3})$-approximation, respectively (**Theorem 3**).

While it was known that optimal semi-matchings contain a maximum matching [HLLT03], we show that there is a hierarchical decomposition of an optimal semi-matching into maximum matchings (**Lemma 10**). Similarly, we show that semi-matchings that do not admit length-two degree-minimizing paths can be

[1] We write $\tilde{O}(n)$ to denote $O(n \operatorname{polylog} n)$.

decomposed into maximal matchings (**Lemma 9**). The latter result allows us to prove that the maximal degree of a semi-matching that does not admit a length-two degree-minimizing path is at most $\lceil \log(n+1) \rceil$ times the maximal degree of an optimal semi-matching (**Theorem 6**).

A Semi-streaming Algorithm for Vertex Arrival Order. In [ANR95], the semi-matching problem is studied in the online model (seen as a scheduling problem). In this model, the A vertices arrive online together with their incident edges, and it has to be irrevocably decided to which B node an A node is matched. It is shown that the greedy algorithm matching an A node to the B node that currently has the smallest degree is $\lceil \log(n+1) \rceil$-competitive, and that this result is tight. This algorithm can also be seen as a one-pass $\lceil \log(n+1) \rceil$-approximation semi-streaming algorithm (meaning $\tilde{\mathrm{O}}(n)$ space) for the semi-matching problem when the input stream is in vertex arrival order. Note that our one-pass algorithm does not assume any order on the input sequence, and when allowing $\tilde{\mathrm{O}}(n)$ space it achieves an $\mathrm{O}(\sqrt{n})$-approximation.

Techniques. Our streaming algorithms are based on the following greedy algorithm. To keep the illustration simple, suppose that the input graph has a perfect matching. Fix a maximal degree $d_{\max}$ (for instance $d_{\max} = n^{1/4}$) and greedily add edges to a set S_1 such that the maximal degree of a B node in S_1 does not exceed $d_{\max}$, and the degree of any A node in S_1 is at most 1. This algorithm leaves at most $\mathrm{O}(n/d_{\max})$ A vertices unmatched in S_1. To match the yet unmatched vertices, we use a second greedy algorithm that we run in parallel to the first one. We fix a parameter d' appropriately (if $d_{\max} = n^{1/4}$ then we set $d' = n^{1/2}$) and for all vertices $a \in A$ we store arbitrary d' edges incident to a in a set E'. Then, we compute an optimal semi-matching S_2 of the unmatched vertices in S_1 and the B nodes only considering the edges in E'. We prove that such a semi-matching has bounded maximal degree (if $d_{\max} = n^{1/4}$ and $d' = n^{1/2}$ then this degree is $n^{1/4}$). The set $S_1 \cup S_2$ is hence a semi-matching of maximal degree $d_{\max} + \deg\max S_2$ and the space requirement of this algorithm is $\tilde{\mathrm{O}}(nd')$. In Section 3 we generalize this idea for any $0 \leq \epsilon \leq 1$ to obtain one-pass algorithms with approximation factors $\mathrm{O}(n^{(1-\epsilon)/2})$ using space $\tilde{\mathrm{O}}(n^{1+\epsilon})$, and a $\log(n)$-pass algorithm with approximation factor $\mathrm{O}(\log n)$ using space $\tilde{\mathrm{O}}(n)$.

In the two-party one-way communication setting, the edge set E of a bipartite graph $G = (A, B, E)$ is split among two players, Alice and Bob. Alice sends a message to Bob and Bob outputs a semi-matching of G. Our communication upper bounds make use of what we call a *c-semi-matching skeleton* (or simply c-skeleton). A c-skeleton of a bipartite graph $G = (A, B, E)$ is a subset of edges $S \subseteq E$ such that for any $A' \subseteq A : \deg\max \mathrm{semi}(A', B, S) \leq c \cdot \deg\max \mathrm{semi}(A', B, E)$ where $\mathrm{semi}(A', B, E')$ denotes an optimal semi-matching between A' and B using edges in E'. We show that if Alice sends a c-skeleton S of her subgraph to Bob, and Bob computes an optimal semi-matching using his edges and the skeleton, then the resulting semi-matching is a $c+1$-approximation. We show that there is an $\mathrm{O}(\sqrt{n})$-skeleton consisting of n edges, and that there is an $\mathrm{O}(n^{1/3})$-skeleton consisting of $2n$ edges. It turns out that an optimal semi-matching is an $\mathrm{O}(\sqrt{n})$-

skeleton, and we show how an $O(n^{1/2})$-skeleton can be improved to an $O(n^{1/3})$-skeleton by adding additional n edges. These skeletons are almost optimal: we show that for any $\epsilon > 0$, an $O(n^{\frac{1}{(1+\epsilon)c+1}})$-skeleton has at least cn edges. Inspired by the prior lower bound, we prove that for any $\epsilon > 0$, the deterministic one-way two-party communication complexity of approximating semi-matchings within a factor $O(n^{\frac{1}{(1+\epsilon)c+1}})$ is at least cn bits.

In order to prove our structure lemmas on semi-matchings, we make use of degree-minimizing paths. Our results on the decomposition of semi-matchings into maximum and maximal matchings directly relate the absence of degree-minimizing paths to the absence of augmenting paths in matchings.

Organization. After presenting notations and definitions in Section 2, we present our streaming algorithms in Section 3. We then discuss the one-way two-party communication setting in Section 4. We conclude with Section 5, where we present our results on the structure of semi-matchings. Due to space limitations, many of our proofs are omitted, however, all of them can be found in the full version of this paper.

2 Notations and Definitions

Let $G = (A, B, E)$ be a bipartite graph and let $n = |A|$. For ease of presentation, we assume that $|B|$ is upper-bounded by a polynomial in n. Let $e \in E$ be an edge connecting nodes $a \in A$ and $b \in B$. Then, we write $A(e)$ to denote the vertex a, $B(e)$ to denote the vertex b, and ab to denote e. Furthermore, for a subset $E' \subseteq E$, we define $A(E') = \bigcup_{e \in E'} A(e)$ (respectively $B(E')$). For subsets $A' \subseteq A$ and $B' \subseteq B$ we write $E'|_{A' \times B'}$ to denote the subset of edges of E' whose endpoints are all in $A' \cup B'$. We denote by $E'(a)$ the set of edges of $E' \subseteq E$ that have an endpoint in vertex a, and $E'(A')$ the set of edges that have endpoints in vertices of A', where $A' \subseteq A$ (similarly we define $E'(B')$ for $B' \subseteq B$).

For a node $v \in A \cup B$, the *neighborhood* of v is the set of nodes that are connected to v and we denote it by $\Gamma(v)$. For a subset $E' \subseteq E$, we write $\Gamma_{E'}(v)$ to denote the neighborhood of v in the graph induced by E'. Note that by this definition $\Gamma(v) = \Gamma_E(v)$. For a subset $E' \subseteq E$, we denote by $\deg_{E'}(v)$ the *degree* in E' of a node $v \in V$, which is the number of edges of E' with an endpoint in v. We define $\deg \max E' := \max_{v \in A \cup B} \deg_{E'}(v)$.

Matchings. A *matching* is a subset $M \subseteq E$ such that $\forall v \in A \cup B : \deg_M(v) \leq 1$. A *maximal matching* is a matching that is inclusion-wise maximal, i.e. it can not be enlarged by adding another edge of E to it. A *maximum matching* is a matching of maximal size. A *length p augmenting path* ($p \geq 3$, p odd) with respect to a matching M is a path $P = (v_1, \ldots, v_{p+1})$ such that $v_1, v_{p+1} \notin A(M) \cup B(M)$ and for $i \leq 1/2(p-1) : v_{2i}v_{2i+1} \in M$, and $v_{2i-1}v_{2i} \notin M$.

Semi-matchings. A *semi-matching* of G is a subset $S \subseteq E$ such that $\forall a \in A : \deg_S(a) = 1$. A *degree-minimizing path* $P = (b_1, a_1, \ldots, b_{k-1}, a_{k-1}, b_k)$ with respect to a semi-matching S is a path of length $2k$ ($k \geq 1$) such that for all $i \leq k : (a_i, b_i) \in S$, for all $i \leq k-1 : (a_i, b_{i+1}) \notin S$, and $\deg_S(b_1) >$

$\deg_S(b_2) \geq \deg_S(b_3) \geq \cdots \geq \deg(b_{k-1}) > \deg(b_k)$. An *optimal semi-matching* $S^* \subseteq E$ is a semi-matching that does not admit any degree-minimizing-paths. For $A' \subseteq A, B' \subseteq B, E' \subseteq E$, we denote by $\text{semi}(A', B', E')$ an optimal semi-matching in the graph $G' = (A', B', E')$, and we denote by $\text{semi}_2(A', B', E')$ a semi-matching that does not admit degree-minimizing paths of length two in G'.

Incomplete d-Bounded Semi-matchings. Let d be an integer. Then an *incomplete d-bounded semi-matching* of G is a subset $S \subseteq E$ such that $\forall a \in A : \deg_S(a) \leq 1$ and $\forall b \in B : \deg_S(b) \leq d$. For subsets $A' \subseteq A, B' \subseteq B, E' \subseteq E$, we write $\text{isemi}_d(A', B', E')$ to denote an incomplete d-bounded semi-matching of maximal size in the graph $G' = (A', B', E')$.

Approximation. We say that an algorithm (or communication protocol) is a c-approximation algorithm (resp. communication protocol) to the semi-matching problem if it outputs a semi-matching S such that $\deg \max S \leq c \cdot \deg \max S^*$, where S^* denotes an optimal semi-matching. We note that this measure was previously used for approximating semi-matching, e.g, in [ANR95].

3 Streaming Algorithms

To present our streaming algorithms, we describe an algorithm, $\text{ASEMI}(G, s, d, p)$ (Algorithm 1), that computes an incomplete $2dp$-bounded semi-matching in the graph G using space $\tilde{\text{O}}(s)$, and makes at most $p \geq 1$ passes over the input stream. If appropriate parameters are chosen, then the output is not only an incomplete semi-matching, but also a semi-matching. We run multiple copies of this algorithm with different parameters in parallel in order to obtain a one-pass algorithm for the semi-matching problem (Theorem 1). Using other parameters, we also obtain a $\log(n)$-pass algorithm, as stated in Theorem 2.

Algorithm 1. Approximating semi-matchings: $\text{ASEMI}(G, s, d, p)$

Require: $G = (A, B, E)$ is a bipartite graph
  $S \leftarrow \varnothing$
  repeat at most p times or until $|A(S)| = |A|$
    $S \leftarrow S \cup \text{INCOMPLETE}(G|_{(A \setminus A(S)) \times B}, s, d)$
  end repeat
  return S

$\text{ASEMI}(G, s, d, p)$ starts with an empty incomplete semi-matching S and adds edges to S by invoking $\text{INCOMPLETE}(G, s, d)$ (Algorithm 2) on the subgraph of the as yet unmatched A vertices in S and all B vertices. Each invocation of $\text{INCOMPLETE}(G, s, d)$ makes one pass over the input stream and returns a $2d$-bounded incomplete semi-matching while using space $\tilde{\text{O}}(s)$. Since we make at most p passes, the resulting incomplete semi-matching has a maximal degree of at most $2dp$.

$\text{INCOMPLETE}(G, s, d)$ collects edges greedily from graph G and puts them into an incomplete d-bounded semi-matching S_1 and a set E'. An edge e from the

Algorithm 2. Computing incomplete semi-matchings: INCOMPLETE(G, s, d)

Require: $G = (A, B, E)$ is a bipartite graph
  $k \leftarrow s/|A|$, $S_1 \leftarrow \varnothing$, $E' \leftarrow \varnothing$
  while $\exists$ an edge ab in stream **do**
    if $ab \notin A \times B$ **then continue**
    if $\deg_{S_1}(a) = 0$ and $\deg_{S_1}(b) < d$ **then** $S_1 \leftarrow S_1 \cup \{ab\}$
    if $\deg_{E'}(a) < k$ **then** $E' \leftarrow E' \cup \{ab\}$
  end while
  $S_2 \leftarrow \text{isemi}_d(E'|_{(A \setminus A(S_1)) \times B})$
  $S \leftarrow S_1 \cup S_2$
  return S

input stream is put into S_1 if $S_1 \cup \{e\}$ is still an incomplete d-bounded semi-matching. An edge $e = ab$ is added to E' if the degree of a in $E' \cup \{e\}$ is less or equal to a parameter k which is chosen to be $s/|A|$ in order to ensure that the algorithm does not exceed space $\tilde{\mathrm{O}}(s)$. The algorithm returns an incomplete $2d$-bounded semi-matching that consists of S_1 and S_2, where S_2 is an optimal incomplete d-bounded semi-matching between the A vertices that are not matched in S_1 and all B vertices, using only edges in E'.

We lower-bound the size of S_2 in Lemma 1 (proof omitted). We prove that for any bipartite graph $G = (A, B, E)$ and any $k > 0$, if we store for each $a \in A$ any $\min\{k, \deg_G(a)\}$ incident edges to a, then we can compute an incomplete d-bounded semi-matching of size at least $\min\{kd, |A|\}$ using only those edges, where d is an upper-bound on the maximal degree of an optimal semi-matching between A and B in G.

Lemma 1 is then used in the proof of Lemma 2, where we show a lower bound on the size of the output $S_1 \cup S_2$ of INCOMPLETE(G, s, d).

Lemma 1. *Let $G = (A, B, E)$ be a bipartite graph, let $k > 0$ and let $d \geq \deg\max\text{semi}(A, B, E)$. Let $E' \subseteq E$ such that for all $a \in A$: $\deg_{E'}(a) = \min\{k, \deg_E(a)\}$. Then there is an incomplete d-bounded semi-matching $S \subseteq E'$ such that $|S| \geq \min\{kd, |A|\}$.*

Lemma 2. *Let $G = (A, B, E)$ be a bipartite graph, let $s \geq |A|$ and let $d \geq \deg\max\text{semi}(A, B, E)$. Then* INCOMPLETE$(G, s, d)$ *(see Algorithm 2) uses $\tilde{\mathrm{O}}(s)$ space and outputs an incomplete $2d$-bounded semi-matching S such that $|S| \geq \min\{|A|\frac{d}{d+d^*} + \frac{ds}{|A|}, |A|\}$.*

Proof. The proof refers to the variables of Algorithm 2 and the values they take at the end of the algorithm. Furthermore, let $S^* = \text{semi}(A, B, E)$, $d^* = \deg\max S^*$, and let $A' = A \setminus A(S_1)$.

Firstly, we lower-bound $|S_1|$. Let $a \in A'$ and $b = S^*(a)$. Then $\deg_{S_1}(b) = d$ since otherwise a would have been matched in S_1. Hence, we obtain $|A(S_1)| \geq d|B(S^*(A'))| \geq d|A'|/d^*$, where the second inequality holds since the maximal degree in S^* is d^*. Furthermore, since $A' = A \setminus A(S_1)$ and $|S_1| = |A(S_1)|$, we obtain $|S_1| \geq |A|\frac{d}{d+d^*}$. We apply Lemma 1 on the graph induced by the

edge set $E'|_{A' \times B}$. We obtain that $|S_2| \geq \min\{ds/|A|, |A'|\}$ and consequently $|S| = |S_1| + |S_2| \geq \min\{|A|\frac{d}{d+d^*} + \frac{ds}{|A|}, |A|\}$.

Concerning space, the dominating factor is the storage space for the at most $k+1$ edges per A vertex, and hence space is bounded by $\tilde{\mathrm{O}}(k|A|) = \tilde{\mathrm{O}}(s)$. □

In the proof of Theorem 1 (proof omitted), for $0 \leq \epsilon \leq 1$ we show that ASEMI(G, $n^{1+\epsilon}$, $n^{(1-\epsilon)/2}d'$, 1) returns a semi-matching if d' is at least the maximal degree of an optimal semi-matching. Using a standard technique, we run $\log(n)+1$ copies of ASEMI for all $d' = 2^i$ with $0 \leq i \leq \log(n)$ and we return the best semi-matching, obtaining a 1-pass algorithm. We use the same idea in Theorem 2, where we obtain a $4\log n$-approximation algorithm that makes $\log(n)$ passes and uses space $\tilde{\mathrm{O}}(n)$ (proof omitted).

Theorem 1. *Let $G = (A, B, E)$ be a bipartite graph with $n = |A|$. For any $0 \leq \epsilon \leq 1$ there is a one-pass streaming algorithm using $\tilde{\mathrm{O}}(n^{1+\epsilon})$ space that computes a $4n^{(1-\epsilon)/2}$-approximation to the semi-matching problem.*

Theorem 2. *Let $G = (A, B, E)$ be a bipartite graph with $n = |A|$. There is a $\log(n)$-pass streaming algorithm using space $\tilde{\mathrm{O}}(n)$ that computes a $4\log n$-approximation to the semi-matching problem.*

4 Two-Party Communication Complexity

We now consider one-way two-party protocols which are given a bipartite graph $G = (A, B, E)$ as input, such that $E_1 \subseteq E$ is given to Alice and $E_2 \subseteq E$ is given to Bob. Alice sends a single message to Bob, and Bob outputs a valid semi-matching S for G. A central idea for our upper and lower bounds is what we call a *c-semi-matching skeleton* (or *c-skeleton*). Given a bipartite graph $G = (A, B, E)$, we define a c-semi-matching skeleton to be a subset of edges $S \subseteq E$ such that $\forall A' \subseteq A : \deg\max\mathrm{semi}(A', B, S) \leq c \cdot \deg\max\mathrm{semi}(A', B, E)$. We show how to construct an $\mathrm{O}(\sqrt{n})$-skeleton of size n, and an $\mathrm{O}(n^{1/3})$-skeleton of size $2n$. We show that if Alice sends a c-skeleton of her subgraph $G = (A, B, E_1)$ to Bob, then Bob can output a $c+1$-approximation to the semi-matching problem. Using our skeletons, we thus obtain one-way two party communication protocols for the semi-matching problem with approximation factors $\mathrm{O}(\sqrt{n})$ and $\mathrm{O}(n^{1/3})$, respectively (Theorem 3). Then, we show that for any $\epsilon > 0$, an $\mathrm{O}(n^{\frac{1}{(1+\epsilon)c+1}})$-skeleton requires at least cn edges. This renders our $\mathrm{O}(\sqrt{n})$-skeleton and our $\mathrm{O}(n^{1/3})$-skeleton tight up to a constant.

Upper Bound. Firstly, we discuss the construction of two skeletons. In Lemma 4, we show that an optimal semi-matching is an $\mathrm{O}(\sqrt{n})$-skeleton. We use the following key observation: Given a bipartite graph $G = (A, B, E)$, let $A' \subseteq A$ be such that A' has minimal expansion, meaning that $A' = \arg\min_{A'' \subseteq A} \frac{|\Gamma(A'')|}{|A''|}$. The maximal degree in a semi-matching is then clearly at least $\lceil \frac{|A'|}{|\Gamma(A')|} \rceil$ since all vertices of A' have to be matched to its neighborhood. However, it is also true that the maximal degree of a semi-matching *equals* $\lceil \frac{|A'|}{|\Gamma(A')|} \rceil$. A similar fact was

used in [GKK12] for fractional matchings, and also in [KRT01]. We state this fact in Lemma 3 (proof omitted).

Lemma 3. *Let $G = (A, B, E)$ with $|A| = n$, and let $d = \deg\max\text{semi}(A, B, E)$. Let $A' = \arg\min_{A'' \subseteq A} \frac{|\Gamma(A'')|}{|A''|}$ and let $\alpha = \frac{|\Gamma(A')|}{|A'|}$. Then: $d = \lceil \alpha^{-1} \rceil$.*

Lemma 4. *Let $G = (A, B, E)$ with $n = |A|$, and let $S = \text{semi}(A, B, E)$. Then: $\forall A' \subseteq A : \deg\max\text{semi}(A', B, S) < \sqrt{n}\,(\deg\max\text{semi}(A', B, E))^{1/2} + 1$.*

Proof. Let $A' \subseteq A$ be an arbitrary subset. Let $A'' = \arg\min_{A''' \subseteq A'} \frac{|\Gamma_S(A''')|}{|A'''|}$, and let $k = |\Gamma_S(A'')|$. Let $d = \deg\max\text{semi}(A', B, S)$. Then by Lemma 3, $d = \lceil \frac{|A''|}{k} \rceil$. Furthermore, since A'' is the set of minimal expansion in S, for all $b \in \Gamma_S(A'') : \deg_S(b) = d$, and hence $|A''| = kd$.

Let $d^* = \deg\max\text{semi}(A'', B, E)$. Then $d^* \leq \deg\max\text{semi}(A', B, E)$, since $A'' \subseteq A'$. It holds that $\forall x \in \Gamma_E(A'') \setminus \Gamma_S(A'') : \deg_S(x) \geq d - 1$ since otherwise there was a degree-minimizing path of length 2 in S. The sum of the degrees of the vertices in $\Gamma_E(A'')$ is upper-bounded by the number of A nodes. We obtain hence $(|\Gamma_E(A'')| - k)(d - 1) + kd \leq n$, and this implies that $|\Gamma_E(A'')| \leq \frac{n-k}{d-1}$. Clearly, $d^* \geq |A''|/|\Gamma_E(A'')|$, and using the prior upper bound on $|\Gamma_E(A'')|$ and the equality $|A''| = kd$, we obtain $d^* \geq \frac{kd(d-1)}{n-k}$ which implies that $d < \sqrt{n}\sqrt{d^*} + 1$ for any $k \geq 1$. □

In order to obtain an $O(n^{1/3})$-skeleton, for each $a \in A$ we add one edge to the $O(\sqrt{n})$-skeleton. Let $S = \text{semi}(A, B, E)$ be the $O(\sqrt{n})$-skeleton, let $B' = B(S)$ be the B nodes that are matched in the skeleton, and for all $b \in B'$ let $A_b = \Gamma_S(b)$ be the set of A nodes that are matched to b in S. Intuitively, in order to obtain a better skeleton, we have to increase the size of the neighborhood in the skeleton of all subsets of A, and in particular of the subsets A_b for $b \in B'$. We achieve this by adding additional optimal semi-matchings $S_b = \text{semi}(A_b, B, E)$ for all subsets A_b with $b \in B'$ to S, see Lemma 5.

Lemma 5. *Let $G = (A, B, E)$ be a bipartite graph with $n = |A|$. Let $S = \text{semi}(A, B, E)$, and for all $b \in B(S) : S_b = \text{semi}(\Gamma_S(b), B, E)$. Then: $\forall A' \subseteq A : \deg\max\text{semi}(A', B, S \cup \bigcup_{b \in B(S)} S_b) \leq \lceil 2n^{1/3} \deg\max\text{semi}(A', B, E) \rceil$.*

We mention that there are graphs for which adding further semi-matchings $S_{b_1 b_2} = \text{semi}(A_{b_1 b_2}, B, E)$ to our $O(n^{1/3})$-skeleton, where $A_{b_1 b_2}$ is the set of A vertices whose neighborhood in our $O(n^{1/3})$-skeleton is the set $\{b_1, b_2\}$, does not help to improve the quality of the skeleton. Finally, we state our main theorem.

Theorem 3. *Let $G = (A, B, E)$ with $n = |A|$ and $m = |B|$. Then there are two one-way two party deterministic communication protocols for the semi-matching problem, one with (1) message size $cn \log m$ and approximation factor $n^{1/2} + 2$, and one with (2) message size $2cn \log m$ and approximation factor $2n^{1/3} + 2$.*

Lower Bounds for Semi-matching-Skeletons. We present now a lower bound that shows that the skeletons of the previous subsection are essentially optimal. For an integer c, we consider the complete bipartite graph $K_{n,m}$ where

m is a carefully chosen value depending on c and n. We show in Lemma 6 (proof omitted) that for any subset of edges E' of $K_{n,m}$ such that for all $a \in A : \deg_{E'}(a) \leq c$, there is a subset $A' \subseteq A$ with $|A'| \leq m$ such that an optimal semi-matching that matches A' using edges in E' has a maximal degree of $\Omega(n^{\frac{1}{c+1}})$. Note that since $|A'| \leq m$, there is a matching in $K_{n,m}$ that matches all A' vertices. This implies that such an E' is only an $\Omega(n^{\frac{1}{c+1}})$-skeleton.

Lemma 6. *Let $G = (A, B, E)$ be the complete bipartite graph with $|A| = n$ and $|B| = (c!)^{\frac{1}{c+1}} n^{\frac{1}{c+1}}$ for an integer c. Let $E' \subseteq E$ be an arbitrary subset such that $\forall a \in A : \deg_{E'}(a) \leq c$. Then there exists an $A' \subseteq A$ with $|A'| \leq |B|$ and $\deg \max \text{semi}(A', B, E') \geq \frac{(c!)^{\frac{1}{c+1}}}{c} n^{\frac{1}{c+1}} > e^{-1.3} n^{\frac{1}{c+1}}$.*

We extend Lemma 6 now to edge sets of bounded cardinality without restriction on the maximal degree of an A node, and we state then our lower-bound result in Theorem 4.

Lemma 7. *Let $c > 0$ be an integer, let $\epsilon > 0$ be a constant, and let $c' = (1 + \epsilon)c$. Let $G = (A, B, E)$ be the complete bipartite graph with $|A| = n$ and $|B| = (c'!)^{\frac{1}{c'+1}} (\frac{\epsilon}{1+\epsilon} \cdot n)^{\frac{1}{c'+1}}$. Let $E' \subseteq E$ be an arbitrary subset of size at most cn. Then there exists an $A' \subseteq A$ with $|A'| \leq |B|$ and $\deg \max \text{semi}(A', B, E') > e^{-1.3} (\frac{\epsilon}{1+\epsilon} n)^{\frac{1}{c'+1}}$.*

Theorem 4. *Let $c > 0$ be an integer. Then for all $\epsilon > 0$, an $\mathrm{O}(n^{\frac{1}{(1+\epsilon)c+1}})$-semi-matching skeleton requires at least cn edges.*

One-Way, Two Party Communication Lower Bound. To prove a lower bound on the deterministic communication complexity we define a family of bipartite graphs. For given integers n and m, let $\mathcal{G}_1 = \{G_1(x) | x \in \{0,1\}^{n \times m}\}$ be defined as follows. Let $B_0 = \{b_1^0, \ldots, b_m^0\}$, $B_1 = \{b_1^1, \ldots, b_m^1\}$ and $A = \{a_1, \ldots, a_n\}$. Given $x \in \{0,1\}^{n \times m}$, let $E_x = \{(a_i, b_j^{x_{i,j}}) \,|\, 1 \leq i \leq n, 1 \leq j \leq m\}$ (i.e, the entries of the matrix x determine if there is an edge (a_i, b_j^0) or an edge (a_i, b_j^1) for all i, j). Then, we define $G_1(x) = (A, B_0 \cup B_1, E_x)$. From Lemma 7 we immediately obtain the following lemma.

Lemma 8. *Let $c > 0$ be an integer, let $\epsilon > 0$ be a constant, and let $c' = (1+\epsilon)c$. Let n be a sufficiently large integer, and let $m = (c'!)^{\frac{1}{c'+1}} (\frac{\epsilon}{1+\epsilon} \cdot n)^{\frac{1}{c'+1}}$. Let $G = (A, B_0 \cup B_1, E)$ be a graph $G \in \mathcal{G}_1$, and let $E' \subseteq E$ be such that $|E'| \leq cn$. Then there exists a set of nodes $A' \subseteq A$ with $|A'| \leq m$ and $\deg \max \text{semi}(A', B_0 \cup B_1, E') > 1/2 e^{-1.3} (\frac{\epsilon}{1+\epsilon} n)^{\frac{1}{c'+1}}$.*

We further define a second family of bipartite graphs $\mathcal{G}_2$ on the sets of nodes A and C, $|A| = |C| = n$. For a set $A' \subseteq A$ we define the graph $G_2(A')$ to be an arbitrary matching from all the nodes of A' to nodes of C. The family of graphs $\mathcal{G}_2$ is defined as $\mathcal{G}_2 = \{G_2(A') | A' \subseteq A\}$.

Our lower bound will be proved using a family of graphs $\mathcal{G}$. Slightly abusing notation, the family of graphs $\mathcal{G}$ is defined as $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$. That is, the graphs

in $\mathcal{G}$ are all graphs $G = (A, B_0 \cup B_1 \cup C, E_1 \cup E_2)$ built from a graph $G_1 = (A, B_0 \cup B_1, E_1) \in \mathcal{G}_1$ and a graph $G_2 = (A, C, E_1) \in \mathcal{G}_2$ where the set of nodes A is the same for G_1 and G_2. We now prove our lower bound.

Theorem 5. *Let $c > 0$ be an integer and let $\epsilon > 0$ be an arbitrarily small constant. Let $\mathcal{P}$ be a β-approximation one-way two-party protocol for semi matching that has communication complexity at most α. If $\beta \leq \gamma = 1/2\frac{1}{e^{1.3}}(\frac{\epsilon}{\epsilon+1}n)^{\frac{1}{(1+\epsilon)c+1}}$, then $\alpha > cn$, where n is the number of nodes to be matched.*

Proof. Take n sufficiently large. Let $c' = (1+\epsilon)c$ and let $m = (c'!)^{\frac{1}{c'+1}}(\frac{\epsilon}{1+\epsilon} \cdot n)^{\frac{1}{c'+1}}$. We consider as possible inputs the graphs in $\mathcal{G}$ (for n and m). Given an input graph, Alice will get as input all edges between A and $B_0 \cup B_1$ (i.e., a graph in $\mathcal{G}_1$) and Bob will get all edges between A and C (i.e., a graph in $\mathcal{G}_2$)

Assume towards a contradiction that the communication complexity of $\mathcal{P}$ is at most cn. Then there is a set of graphs $\mathcal{G}^* \subseteq \mathcal{G}_1$, $|\mathcal{G}^*| \geq 2^{nm-cn}$, such that on all graphs in $\mathcal{G}^*$ Alice sends the same message to Bob. Consider the set $X^* \subseteq \{0,1\}^{n \times m}$ such that $\mathcal{G}^* = \{G_1(x) \,|\, x \in X^*\}$, Since there is a one-to-one correspondence between $\mathcal{G}^*$ and X^*, $|X^*| \geq 2^{nm-cn}$, and there are at most cn entries which are constant over all matrices in X^*, otherwise $|X^*| < 2^{nm-cn}$. This means that there are at most cn edges that exist in all graphs in $\mathcal{G}^*$. Let E' be the set of all these edges.

Consider now the graph $G = (A, B_0 \cup B_1, E')$. Since $|E'| \leq cn$, by Lemma 8 there exists a set $A' \subseteq A$ with $|A'| \leq m$ and $\deg\max\mathrm{semi}(A', B_0 \cup B_1, E') > \gamma$. We now define $G_2^* \in \mathcal{G}_2$ to be $G_2^* = G_2(A \setminus A')$.

Now observe that on any of $G \in \mathcal{G}^* \times \{G_2^*\} \subseteq \mathcal{G}$, $\mathcal{P}$ gives the same output semi-matching S. S can include, as edges matching the nodes in A', only edges from E', since for any other edge there exists an input in $\mathcal{G}^* \times \{G_2^*\}$ in which that edge does not exist and $\mathcal{P}$ would not be correct on that input. It follows (by Lemma 8) that the maximum degree of S is greater than γ. On the other hand, since $|A'| \leq m$, there is a perfect matching in any graph in $\mathcal{G}^* \times \{G_2^*\}$. The approximation ratio of $\mathcal{P}$ is therefore greater than γ. A contradiction. □

5 The Structure of Semi-matchings

We now present our results concerning the structure of semi-matchings. All proofs of this section are deferred to the full version of this paper. Firstly, we show in Lemma 9 that a semi-matching that does not admit length-two degree-minimizing paths can be decomposed into maximal matchings. In Lemma 10, we show that if a semi-matching does not admit *any* degree-minimizing paths, then there is a similar decomposition into maximum matchings. Lemma 9 is then used to prove that semi-matchings that do not admit length-two degree-minimizing paths approximate optimal semi-matchings within a factor $\lceil \log(n+1) \rceil$. To this end, in the full version of this paper we show that the first d^* maximal matchings of the decomposition of such a semi-matching match at least $1/2$ of the A vertices, where d^* is the maximal degree of an optimal semi-matching. In Theorem 6,

we then apply this result $\lceil \log(n+1) \rceil$ times, showing that the maximal degree of a semi-matching that does not admit length-two degree-minimizing paths is at most $\lceil \log(n+1) \rceil$ times the maximal degree of an optimal semi-matching.

Lemma 9. *Let $S = \text{semi}_2(A, B, E)$ be a semi-matching in G that does not admit a length-two degree-minimizing path, and let $d = \deg\max S$. Then S can be partitioned into d matchings $M_1, \ldots, M_d$ such that $\forall i : M_i$ is a maximal matching in $G|_{A_i \times B_i}$, where $A_1 = A$, $B_1 = B$, and for $i > 1 : A_i = A \setminus \bigcup_{1 \leq j < i} A(M_j)$ and $B_i = B(M_{j-1})$.*

Lemma 10. *Let $S^* = \text{semi}(A, B, E)$ be a semi-matching in G that does not admit degree-minimizing paths of any length, and let $d^* = \deg\max S^*$. Then S^* can be partitioned into d^* matchings $M_1, \ldots, M_{d^*}$ such that $\forall i : M_i$ is a maximum matching in $G|_{A_i \times B_i}$, where $A_1 = A$, $B_1 = B$, and for $i > 1 : A_i = A \setminus \bigcup_{1 \leq j < i} A(M_j)$ and $B_i = B(M_{j-1})$.*

Theorem 6. *Let $S = \text{semi}_2(A, B, E)$ be a semi-matching of G that does not admit a length-two degree-minimizing path. Let $S^* = \text{semi}(A, B, E)$ be an optimal semi-matching in G. Then $\deg\max S \leq \lceil \log(n+1) \rceil \deg\max S^*$.*

References

[Abr03] Abraham, D.: Algorithmics of two-sided matching problems. Master's thesis, University of Glasgow (2003)

[AG11] Ahn, K.J., Guha, S.: Linear programming in the semi-streaming model with application to the maximum matching problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 526–538. Springer, Heidelberg (2011)

[ANR95] Azar, Y., Naor, J.S., Rom, R.: The competitiveness of on-line assignments. J. Algorithms 18(2), 221–237 (1995)

[Ber57] Berge, C.: Two Theorems in Graph Theory. Proc. of the National Academy of Sciences of the United States of America 43(9), 842–844 (1957)

[ECS73] Eruno, J., Coffman Jr., E.G., Sethi, R.: Scheduling independent tasks to reduce mean finishing-time. In: Proc. of the 4th ACM Symposium on Operating System Principles, SOSP 1973, pp. 102–103 (1973)

[FKM+05] Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: Graph distances in the streaming model: the value of space. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005 (2005)

[FLN10] Fakcharoenphol, J., Laekhanukit, B., Nanongkai, D.: Faster algorithms for semi-matching problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 176–187. Springer, Heidelberg (2010)

[GKK12] Goel, A., Kapralov, M., Khanna, S.: On the communication and streaming complexity of maximum bipartite matching. In: Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012 (2012)

[HLLT03] Harvey, N.J.A., Ladner, R.E., Lovász, L., Tamir, T.: Semi-matchings for bipartite graphs and load balancing. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 294–306. Springer, Heidelberg (2003)

[Hor73] Horn, W.A.: Minimizing average flow time with parallel machines. Operations Research, 846–847 (1973)

[Kap13] Kapralov, M.: Better bounds for matchings in the streaming model. In: Proc. of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013 (2013)

[KMM12] Konrad, C., Magniez, F., Mathieu, C.: Maximum matching in semi-streaming with few passes. In: Gupta, A., Jansen, K., Rolim, J., Servedio, R. (eds.) APPROX/RANDOM 2012. LNCS, vol. 7408, pp. 231–242. Springer, Heidelberg (2012)

[KRT01] Kleinberg, J., Rabani, Y., Tardos, É.: Fairness in routing and load balancing. Journal of Computer and System Sciences 63(1), 2–20 (2001)

[LL04] Lin, Y., Li, W.: Parallel machine scheduling of machine-dependent jobs with unit-length. European Journal of Operational Research 156(1), 261–266 (2004)

Full-Fledged Real-Time Indexing for Constant Size Alphabets

Gregory Kucherov[1,2] and Yakov Nekrich[3,⋆]

[1] Laboratoire d'Informatique Gaspard Monge, Université Paris-Est & CNRS, Marne-la-Vallée, Paris, France
Gregory.Kucherov@univ-mlv.fr
[2] Department of Computer Science, Ben-Gurion University of the Negev, Be'er Sheva, Israel
[3] Department of Electrical Engineering & Computer Science, University of Kansas
yakov.nekrich@googlemail.com

Abstract. In this paper we describe a data structure that supports pattern matching queries on a dynamically arriving text over an alphabet of constant size. Each new symbol can be prepended to T in $O(1)$ expected worst-case time. At any moment, we can report all occurrences of a pattern P in the current text in $O(|P| + k)$ time, where $|P|$ is the length of P and k is the number of occurrences. This resolves, under assumption of constant size alphabet, a long-standing open problem of existence of a real-time indexing method for string matching (see [2]).

1 Introduction

Two main versions of the string matching problem differ in which of the two components – pattern P or text T – is provided first in the input (or is considered as fixed) and can then be preprocessed before processing the other component. The framework when the text has to be preprocessed is usually called *indexing*, as it can be viewed as constructing a text index supporting matching queries.

Real-time variants of the string matching problem are about as old as the string matching itself. In the 70s, existence of real-time string matching algorithms was first studied for Turing machines. For example, it has been shown that the language $\{P\#T\}$ where P occurs in T can be recognized by a Turing machine, while the language $\{T\#P\}$ cannot [7]. In the realm of the RAM model, the real-time variant of pattern-preprocessing string matching has been extensively studied, leading to very efficient solutions (see e.g. [3] and references therein). The indexing variant, however, still has important unsolved questions.

In the real-time indexing problem, we have to maintain an indexing data structure for a text that arrives online, by spending $O(1)$ worst-case time on each new character; a string matching query must be answered in $O(|P|)$ time for a query string P. Back in the 70s, Slisenko [15] claimed a solution to the

⋆ This work was done while this author was at Laboratoire d'Informatique Gaspard Monge, Université Paris-Est & CNRS

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 650–660, 2013.

real-time indexing problem, but its complex and voluminous full description made it unacknowledged by the scientific community, and the problem remained to be considered open for many years. In 1994, Kosaraju [11] presented another solution which, however, did not support repetitive matching queries on different portions of arriving text, but assumed that the text is entirely read before the matching query is made. In 2008, Amir and Nor [2] proposed another algorithm that fixes this drawback and allows queries to be made at any moment of the text scan.

All the three existing real-time indexing solutions [15,11,2] support only existential queries asking whether the pattern occurs in the text, but are unable to report occurrences of the pattern. Designing a real-time text indexing algorithm that would support queries on all occurrences of a pattern is stated in [2] as the most important remaining open problem. The algorithms of [11,2] assume a constant size alphabet and are both based on constructions of "incomplete" suffix trees which can be built real-time but can only answer existential queries. To output all occurrences of a pattern, a fully-featured suffix tree is needed, however a real-time suffix tree construction, first studied in [1], is in itself an open question. The best currently known algorithm [4] spends $O(\log \log n)$ worst-case time on each character, and a truly real-time construction seems unlikely to exist. Therefore, a suffix tree alone seems to be insufficient to solve the real-time indexing problem.

In this paper, we propose the first real-time text indexing solution that supports reporting all pattern occurrences, under the assumption of constant size alphabet. The general idea is to maintain several data structures, three in our case, each supporting queries for different pattern lengths. Our method employs the suffix tree construction technique recently proposed by Kopelowitz [9]. Similar to [9] and to previous real-time indexing solutions [11,2], we assume that the text is read right-to-left, or otherwise the pattern needs to be reversed before executing the query. We use the word RAM computation model; the same model is also used in e.g., [4,9].

The paper is organized as follows. In Section 2.1, we describe auxiliary data structures and Kopelowitz' technique that are essential for our algorithm. In Section 3, we describe the three data structures for different pattern lengths that constitute a basis of our solution. These data structures, however, do not provide a fully real-time algorithm. Then in Section 4, we show how to "fix" the solution of Section 3 in order to obtain a fully real-time algorithm.

Throughout the paper, Σ is an alphabet of constant size σ. Since the text T is read right-to-left, it will be convenient for us to enumerate symbols of T from the end, i.e. $T = t_n \dots t_1$ and substring $t_{i+\ell} t_{i+\ell-1} \dots t_i$ will be denoted $T[i+\ell..i]$. $T[i..]$ denotes suffix $T[i..1]$. Throughout this paper, we reserve k to denote the number of objects (occurrences of a pattern, elements in a list, etc) in the query answer.

2 Preliminaries

In this Section, we describe main algorithmic tools used by our algorithms.

2.1 Range Reporting and Predecessor Queries on Colored Lists

We use data structures from [13] for searching in dynamic colored lists.

Colored Range Reporting in a List. Let elements of a dynamic linked list $\mathcal{L}$ be assigned positive integer values called *colors*. A colored range reporting query on a list $\mathcal{L}$ consists of two integers $col_1 < col_2$ and two pointers ptr_1 and ptr_2 that point to elements e_1 and e_2 of $\mathcal{L}$. An answer to a colored range reporting query consists of all elements $e \in \mathcal{L}$ occurring between e_1 and e_2 (including e_1 and e_2) such that $col_1 \leq \texttt{col}(e) \leq col_2$, where $\texttt{col}(e)$ is the color of e. The following result on colored range reporting has been proved by Mortensen [13].

Lemma 1 ([13]). *Suppose that* $\texttt{col}(e) \leq \log^f n$ *for all* $e \in \mathcal{L}$ *and some constant* $f \leq 1/4$. *We can answer color range reporting queries on* $\mathcal{L}$ *in* $O(\log\log m + k)$ *time using an* $O(m)$*-space data structure, where* m *is the number of elements in* $\mathcal{L}$. *Insertion of a new element into* $\mathcal{L}$ *is supported in* $O(\log\log m)$ *time.*

Note that the bound $f \leq 1/4$ follows from the description in [12]: the data structure in [13] uses Q-heaps [6] to answer certain queries on the set of colors in constant time.

Colored Predecessor Problem. The colored predecessor query on a list $\mathcal{L}$ consists of an element $e \in \mathcal{L}$ and a color col. The answer to a query (e, col) is the closest element $e' \in \mathcal{L}$ which precedes e such that $\texttt{col}(e) = col$. The following Lemma is also proved in [13]; we also refer to [8], where a similar problem is solved.

Lemma 2 ([13]). *Suppose that* $\texttt{col}(e) \leq \log^f n$ *for all* $e \in \mathcal{L}$ *and some constant* $f \leq 1/4$. *There exists an* $O(m)$ *space data structure that answers colored predecessor queries on* $\mathcal{L}$ *in* $O(\log\log m)$ *time and supports insertions in* $O(\log\log m)$ *time, where* m *is the number of elements in* $\mathcal{L}$.

2.2 On-Line Indexing for Alphabets of Small Size

Kopelowitz [9] described an online indexing method that works for an arbitrarily large alphabet A. We describe below a simplified version of his algorithm, adapted to our purposes, for the case when the alphabet size $|A| \leq \log^{1/4} n$. For a current text T, Kopelowitz' algorithm maintains a list $\mathcal{S}$ of its lexicographically sorted suffixes and a suffix tree $\mathcal{T}$.[1] Besides, the following auxiliary data structures are used. For any symbol $a \in A$ that occurs in T at least once, we store a in a data structure $\mathcal{A}$. Since $\mathcal{A}$ contains at most $\log^{1/4} n$ elements, we can search in $\mathcal{A}$ in $O(1)$ time using Q-heaps [6]. For every a in $\mathcal{A}$, we store a pointer $last(a)$ to the last (lexicographically largest) suffix of T that starts with a. Furthermore, every suffix $T[i..]$ in $\mathcal{S}$ is colored with color t_{i+1}, i.e., the color of $T[i..]$ is the symbol that precedes the starting position of $T[i..]$ in T. We maintain the structure $\mathcal{D}$ for colored predecessor queries on $\mathcal{S}$, as in Lemma 2. For each $T[i..]$ in $\mathcal{S}$, we also store a pointer to the suffix $T[i+1..]$ in $\mathcal{S}$. Finally, we store a data

[1] In subsequent sections we will consider $\mathcal{S}$ to be a part of $\mathcal{T}$.

structure for weighted level ancestor queries on $\mathcal{T}$: for any leaf v of $\mathcal{T}$ and for any integer q, we can find the lowest ancestor u of v such that the string depth of u is smaller than q. The data structure from [10] uses linear space, supports dynamic tree updates in expected $O(\log \log n)$ time, and answers the weighted level ancestor queries in worst-case $O(\log \log n)$ time.

The algorithm of Kopelowitz [9] consists of three phases. During the first phase we prepend a symbol t_{n+1} to the current text $T = t_n \dots t_1$ and find the position of the new suffix $t_{n+1}T$ in the lexicographically ordered list of suffixes. To do this, we first check if t_{n+1} occurs at least once in T. We query $\mathcal{A}$ for the largest symbol $a \leq t_{n+1}$; if $a \neq t_{n+1}$, then the suffix $t_{n+1}T$ should be inserted after $last(a)$ into $\mathcal{S}$. If $a = t_{n+1}$, at least one suffix of T starts with t_{n+1}. Using $\mathcal{D}$, we look for the predecessor of $T[n..]$ in $\mathcal{S}$ colored with t_{n+1}. Let $T[j..]$ denote such a predecessor of $T[n..]$. Then $T[j+1..]$ starts with symbol t_{n+1}, and it is easy to check that $T[j+1..]$ is the lexicographically largest suffix that precedes $t_{n+1}T$.

When we know suffixes $S' = T[i'..]$ and $S'' = T[i''..]$ of T that respectively precede and follow $t_{n+1}T$, we can find the longest common prefixes $\ell' = lcp(t_{n+1}T, S')$ and $\ell'' = lcp(t_{n+1}T, S'')$ in $O(1)$ time using the data structure of [5]. If $\ell' > \ell''$, we find the leaf v of $\mathcal{T}$ that holds S' and the lowest ancestor u of v with string depth at most ℓ'; u is found using the dynamic weighted level ancestor structure from [10]. If the string depth of u equals ℓ', we create a new child of u that holds the new suffix $t_{n+1}T$. Otherwise, let w be a child of u which is an ancestor of v. We split the edge from u to w and create a new node u'. Then, we create a new child of u' that holds t_oT. The case $\ell'' \geq \ell'$ is symmetric. When the new suffix is inserted into the suffix tree, we update all the auxiliary data structures during the third phase. We refer to [9] for a detailed description of the algorithm.

The only difference between the described procedure and the original algorithm of Kopelowitz [9] is that our method assumes an alphabet of size $|A| \leq \log^{1/4} n$, which allows us to employ the colored predecessor data structure to search for the position of suffix $t_{n+1}T$ during the first phase. The algorithm in [9] works for an arbitrarily large alphabet, and therefore requires more complicated data structures.

3 Fast Off-Line Solution

In this section we describe the main part of our algorithm. The algorithm updates the index by reading the text in the right-to-left order. However, the algorithm we describe now will not be on-line, as it will have to access symbols to the left of the currently processed symbol. Another "flaw" of the algorithm is that it will support pattern matching queries only with an additional exception: we will be able to report all occurrences of a pattern except for those with start positions among a small number of most recently processed symbols of T. In the next section we will show how to fix these issues and turn our algorithm into a fully real-time indexing solution that reports all occurrences of a pattern.

The algorithm distinguishes between three types of query patterns depending on their length: *long patterns* contain at least $(\log\log n)^2$ symbols, *medium-size patterns* contain between $(\log^{(3)} n)^2$ and $(\log\log n)^2$ symbols, and *short patterns* contain less than $(\log^{(3)} n)^2$ symbols[2]. For each of the three types of patterns, the algorithm will maintain a separate data structure supporting queries in $O(|P|+k)$ time for matching patterns of the corresponding type.

3.1 Long Patterns

To match long patterns, we maintain a *sparse suffix tree* $\mathcal{T}_L$ storing only suffixes that start at positions $q \cdot d$ for $q \geq 1$ and $d = \log\log n/(4\log\sigma)$. Suffixes stored in $\mathcal{T}_L$ are regarded as strings over a meta-alphabet of size $\sigma^d = \log^{1/4} n$. This allows us to use the method of Section 2.2 to maintain $\mathcal{T}_L$. Recall that the method maintains a list of sorted suffixes that we denote $\mathcal{L}_L$.

Using $\mathcal{T}_L$ we can find occurrences of a pattern P that start at positions qd for $q \geq 1$, but not occurrences starting at positions $qd+\delta$ for $1 \leq \delta < d$. To be able to find all occurrences, we maintain an additional list $\mathcal{L}_E$ defined as follows.

The list $\mathcal{L}_E$ contains copies of all nodes of $\mathcal{T}_L$ as they occur during the Euler tour of $\mathcal{T}_L$. Thus, $\mathcal{L}_E$ contains one element for each leaf and two elements for each internal node of $\mathcal{T}_L$. The first copy of an internal node u precedes the copies of all nodes in the subtree of u, and the second copy of u occurs immediately after the copies of all descendants of u. To simplify the presentation, we will not distinguish between elements of $\mathcal{L}_E$ and suffix tree nodes that they represent. If a node of $\mathcal{L}_E$ is a leaf that corresponds to a suffix $T[i..]$, we mark it with the meta-symbol $\overleftarrow{T}[i,d] = t_{i+1}t_{i+2}\dots t_{i+d}$ which is interpreted as the color of the leaf for the suffix $T[i..]$. Colors are ordered by lexicographic order of underlying strings. If $S = s_1 \dots s_j$ is a string with $j < d$, then S defines an interval of colors, denoted $[minc(S), maxc(S)]$, corresponding to all strings of length d with prefix S. Recall that there are $\log^{1/4} n$ different colors. On list $\mathcal{L}_E$, we maintain the data structure of Lemma 1 for colored range reporting queries.

After reading character t_i where $i = qd$ for $q \geq 1$, we add the suffix $T[i..]$, viewed as a string over the meta-alphabet of cardinality $\log^{1/4} n$, to $\mathcal{T}_L$ according to the algorithm described in Section 2.2. In addition, we have to update the list $\mathcal{L}_E$, i.e. to insert to $\mathcal{L}_E$ the new leaf holding the suffix $T[i..]$ marked with the color $t_{i+1}t_{i+2}\dots t_{i+d}$. (Note that here the algorithms "looks ahead" for the forthcoming d letters of T.) If a new internal node has been inserted into $\mathcal{T}_L$, we also update the list $\mathcal{L}_L$ accordingly. (Details are left out and can be found e.g. in [12].)

Since the meta-alphabet size is only $\log^{1/4} n$, the navigation in $\mathcal{T}_L$ from a node to a child can be supported in $O(1)$ time. Observe that the children of any internal node $v \in \mathcal{T}_L$ are naturally ordered by the lexicographic order of edge labels. We store the children of v in a data structure $\mathcal{P}_v$ which allows us to find in time $O(1)$ the child whose edge label starts with a string (meta-symbol)

[2] Henceforth, $\log^{(3)} n = \log\log\log n$.

$S = s_1 \dots s_d$. Moreover, we can also compute in time $O(1)$ the "smallest" and the "largest" child of v whose edge label starts with a string $S = s_1 \dots s_j$ with $j \leq d$. $\mathcal{P}_v$ will also support adding a new edge to $\mathcal{P}_v$ in $O(1)$ time. Data structure $\mathcal{P}_v$ can be implemented using e.g. atomic heaps [6]; since all elements in $\mathcal{P}_v$ are bounded by $\log^{1/4} n$, we can also implement $\mathcal{P}_v$ as described in [14].

We now consider a long query pattern $P = p_1 \dots p_m$ and show how the occurrences of P are computed. An occurrence of P is said to be a δ-occurrence if it starts in T at a position $j = qd + \delta$, for some q. For each δ, $0 \leq \delta \leq d-1$, we find all δ-occurrences as follows. First we "spell out" $P_\delta = p_{\delta+1} \dots p_m$ in $\mathcal{T}_L$ over the meta-alphabet, i.e. we traverse $\mathcal{T}_L$ proceeding by blocks of up to d letters of Σ. If this process fails at some step, then P has no δ-occurrences. Otherwise, we spell out P_δ completely, and retrieve the closest explicit descendant node v_δ, or a range of descendant nodes $v_\delta^l, v_\delta^{l+1}, \dots, v_\delta^r$ in the case when P_δ spells to an explicit node except for a suffix of length less than d. The whole spelling step takes time $O(|P|/d + 1)$.

Now we jump to the list $\mathcal{L}_E$ and retrieve the first occurrence of v_δ (or v_δ^l) and the second occurrence of v_δ (or v_δ^r) in $\mathcal{L}_E$. A leaf u of $\mathcal{T}$ corresponds to a δ-occurrence of P if and only if u occurs in the subtree of v_δ (or the subtrees of $v_\delta^l, \dots, v_\delta^r$) and the color of u belongs to $[minc(p_\delta \dots p_1), maxc(p_\delta \dots p_1)]$. In the list $\mathcal{L}_E$, these leaves occur precisely within the interval we computed. Therefore, all δ-occurrences of P can be retrieved in time $O(\log\log n + k_\delta)$ by a colored range reporting query (Lemma 1), where k_δ is the number of δ-occurrences. Summing up over all δ, all occurrences of a long pattern P can be reported in time $O(d(|P|/d + \log\log n) + k) = O(|P| + d\log\log n + k) = O(|P| + k)$, as $d = \log\log n/(4\log\sigma)$, $\sigma = O(1)$ and $|P| \geq (\log\log n)^2$.

3.2 Medium-Size Patterns

Now we show how to answer matching queries for patterns P where $(\log^{(3)} n)^2 \leq |P| < (\log\log n)^2$. In a nutshell, we apply the same method as in Section 3.1 with the main difference that the sparse suffix tree will store only *truncated suffixes* of length $(\log\log n)^2$, i.e. prefixes of suffixes bounded by $(\log\log n)^2$ symbols. We store truncated suffixes starting at positions spaced by $\log^{(3)} n = \log\log\log n$ symbols. The total number of different truncated suffixes is at most $\sigma^{(\log\log n)^2}$. This small number of suffixes will allow us to search and update the data structures faster compared to Section 3.1. We now describe the details of the construction.

We store all truncated suffixes that start at positions qd', for $q \geq 1$ and $d' = \log^{(3)} n$, in a tree $\mathcal{T}_M$. $\mathcal{T}_M$ is organized in the same way as the standard suffix tree; that is, $\mathcal{T}_M$ is a compressed trie for substrings $T[qd'..qd' - (\log\log n)^2 + 1]$, where these substrings are regarded as strings over the meta-alphabet $\Sigma^{d'}$.[3] Observe that the same truncated suffix can occur several times. Therefore, we

[3] For simplicity we assume that $\log^{(3)} n$ and $\log\log n$ are integers and $\log^{(3)} n$ divides $\log\log n$. If this is not the case, we can find d' and d that satisfy these requirements such that $\log\log n \leq d \leq 2\log\log n$ and $\log^{(3)} n \leq d' \leq 2\log^{(3)} n$.

augment each leaf v with a list of *colors* $Col(v)$ corresponding to left contexts of the corresponding truncated suffix S. More precisely, if $S = T[qd'..qd' - (\log\log n)^2 + 1]$ for some $q \geq 1$, then $\overleftarrow{T}[qd', d']$ is added to $Col(v)$. Note that the number of colors is bounded by $\sigma^{\log^{(3)} n}$. Futhermore, for each color col in $Col(v)$, we store all positions $i = qd'$ of T such that S occurs at i and $\overleftarrow{T}[i, d'] = col$. As in Section 3.1, we store a list $\mathcal{L}_M$ that contains colored elements corresponding to the Euler tour traversal of $\mathcal{T}_M$. For each internal node, $\mathcal{L}_M$ contains two elements. For every leaf v and for each value col in its color list $Col(v)$, $\mathcal{L}_M$ contains a separate element colored with col. Observe that since the size of $\mathcal{L}_M$ is bounded by $O(\sigma^{(\log\log n)^2 + \log^{(3)} n})$, updates of $\mathcal{L}_M$ can be supported in $O(\log\log(\sigma^{(\log\log n)^2})) = O(\log^{(3)} n)$ time, and colored reporting queries on $\mathcal{L}_M$ can be answered in $O(\log^{(3)} n + k)$ time (see Lemma 1).

Truncated suffixes are added to $\mathcal{T}_M$ using a method similar to that of Section 3.1. After reading a symbol $t_{qd'}$ for some $q \geq 1$, we add $S_{\text{new}} = T[qd'..qd' - (\log\log n)^2 + 1]$ colored with $\overleftarrow{T}[qd', d']$ to the tree $\mathcal{T}_M$. To find the place of S_{new} in the list of leaves of $\mathcal{T}_M$, here we compare truncated suffixes directly rather than using predecessor queries, as in Kopelowitz's algorithm (Section 2.2). Observe that every truncated suffix can be viewed as an integer in the range $[1..U]$ for $U = \sigma^{(\log\log n)^2}$. We store current truncated suffixes in the van Emde Boas data structure $\mathcal{V}$. Using $\mathcal{V}$, we can find the largest $S_{\text{prev}} \leq S_{\text{new}}$ and the smallest $S_{\text{next}} \geq S_{\text{new}}$ in $\mathcal{T}_M$ in $O(\log\log U) = O(\log^{(3)} n)$ time. Let $\ell' = lcp(S_{\text{prev}}, S_{\text{new}})$, $\ell'' = lcp(S_{\text{next}}, S_{\text{new}})$, and $\ell = \max(\ell', \ell'')$. Observe that lcp values can be computed in $O(1)$ time using standard bit operations. Once ℓ is known, we update the tree $\mathcal{T}_M$ spending $O(\log\log|\mathcal{T}_M|) = O(\log^{(3)} n)$ expected time on the weighted level ancestor query. Finally, we update $\mathcal{L}_M$: if $\mathcal{L}_M$ already contains a leaf with string value S_{new} and color $\overleftarrow{T}[qd', d']$, we add qd' to the list of its occurrences, otherwise we insert a new element into $\mathcal{L}_M$ and initialize its location list to qd'. Altogether, the addition of a new truncated suffix S_{new} requires $O(\log^{(3)} n)$ time.

A query for a pattern $P = p_1 \dots p_m$, such that $(\log^{(3)} n)^2 \leq m < (\log\log n)^2$, is answered in the same way as in Section 3.1. For each $\rho = 0, \dots, \log^{(3)} n - 1$, we find locus nodes $v^l_\rho, \dots, v^r_\rho$ (possibly with $v^l_\rho = v^r_\rho$) of $P_\rho = p_{\rho+1} \dots p_m$. Then, we find all elements in $\mathcal{L}_M$ occurring between the first occurrence of v^l_ρ and the second occurrence of v^r_ρ and colored with a color col that belongs to $[minc(p_\rho \dots p_1), maxc(p_\rho \dots p_1)]$. For every such element, we traverse the associated list of occurrences: if a position i is in the list, then P occurs at position $(i + \rho)$. The total time needed to find all occurrences of a medium-size pattern P is $O(d'(|P|/d' + \log^{(3)} n) + k) = O(|P| + (\log^{(3)} n)^2 + k) = O(|P| + k)$ since $|P| \geq (\log^{(3)} n)^2$.

3.3 Short Patterns

Finally, we describe our indexing data structure for patterns P with $|P| < (\log^{(3)} n)^2$. We maintain the tree $\mathcal{T}_S$ of truncated suffixes of length $\Delta = (\log^{(3)} n)^2$ seen so far in the text. For every position i of T, $\mathcal{T}_S$ contains the substring

$T[i..i-\Delta+1]$. $\mathcal{T}_S$ is organized as a compacted trie. We support queries and updates on $\mathcal{T}_S$ using tabulation. There are $O(2^{\sigma^\Delta})$ different trees, and $O(\sigma^\Delta)$ different queries can be made on each tree. Therefore, we can afford explicitly storing all possible trees $\mathcal{T}_S$ and tabulating possible tree updates. Each internal node of a tree stores pointers to its lefmost and rightmost leaves, the leaves of a tree are organized in a list, and each leaf stores the encoding of the corresponding string Q.

The *update table* $\mathbf{T}_u$ stores, for each tree $\mathcal{T}_S$ and for any string Q, $|Q| = \Delta$, a pointer to the tree $\mathcal{T}'_S$ (possibly the same) obtained after adding Q to $\mathcal{T}_S$. Table $\mathbf{T}_u$ uses $O(2^{\sigma^\Delta}\sigma^\Delta) = o(n)$ space. The *output table* $\mathbf{T}_o$ stores, for every string Q of length Δ, the list of positions in the current text T where Q occurs. $\mathbf{T}_o$ has $\sigma^\Delta = o(n)$ entries and all lists of occurrences take $O(n)$ space altogether.

When scanning the text, we maintain the encoding of the string Q of Δ most recently read symbols of T. The encoding is updated after each symbol using bit operations. After reading a new symbol, the current tree $\mathcal{T}_S$ is updated using table $\mathbf{T}_u$ and the current position is added to the entry $\mathbf{T}_o[Q]$. Updates take $O(1)$ time.

To answer a query P, $|P| < \Delta$, we find the locus u of P in the current tree $\mathcal{T}_S$, retrieve the leftmost and rightmost leaves and traverse the leaves in the subtree of u. For each traversed leaf v_l with label Q, we report the occurrences stored in $\mathbf{T}_o[Q]$. The query takes time $O(|P| + k)$.

4 Real-Time Indexing

The indexes for long and medium-size patterns, described in Sections 3.1 and 3.2 respectively, do not provide real-time indexing solutions for several reasons. The index for long patterns, for example, requires to look ahead for the forthcoming d symbols when processing symbols t_i for $i = qd$, $q \geq 1$. Furthermore, for such i, we are unable to find occurrences of query patterns P starting at positions $t_{i-1} \dots t_{i-d+1}$ before processing t_i. A similar situation holds for medium-size patterns. Another issue is that in our previous development we assumed the length n of T to be known, whereas this may of course not be the case in the real-time setting. In this Section, we show how to fix these issues in order to turn the indexes real-time. Firstly we show how the data structures of Sections 3.1 and 3.2 can be updated in a real-time mode. Then, we describe how to search for patterns that start among most recently processed symbols. We describe our solutions to these issues for the case of long patterns, as a simple change of parameters provides a solution for medium-size patterns too. Finally, we will show how we can circumvent the fact that the length of T is not known in advance.

In the algorithm of Section 3.1, the text is partitioned into blocks of length d, and the insertion of a new suffix $T[i..]$ is triggered only when the leftmost symbol t_i of a block is reached. The insertion takes time $O(d)$ and assumes the knowledge of the forthcoming block $t_{i+d} \dots t_{i+1}$. To turn this algorithm real-time, we apply a standard deamortization technique. We distribute the cost of

the insertion of suffix $T[i-d..]$ over d symbols of the block $t_{i+d} \dots t_{i+1}$. This is correct, as by the time we start reading the block $t_{i+d} \dots t_{i+1}$, we have read the block $t_i \dots t_{i-d+1}$ and therefore have all necessary information to insert suffix $T[i-d..]$. In this way, we spend $O(1)$ expected time per symbol to update all involved data structures.

Now assume we are reading a block $t_{i+d} \dots t_{i+1}$, i.e. we are processing some symbol $t_{i+\delta}$ for $1 \leq \delta < i$. At this point, we are unable to find occurrences of a query pattern P starting at $t_{i+\delta} \dots t_{i+1}$ as well as within the two previous blocks, as they have not been indexed yet. This concerns up to $(3d-1)$ most recent symbols. We then introduce a separate procedure to search for occurrences that start in $3d$ leftmost positions of the already processed text. This can be done by simply storing T in a compact form T_c where every $\log_\sigma n$ consecutive symbols are packed into one computer word[4]. Thus, T_c uses $O(|T|/\log_\sigma n)$ words of space. Using T_c, we can test whether $T[j..j-|P|+1] = P$, for any pattern P and any position j, in $O(\lceil |P|/\log_\sigma n \rceil) = o(|P|/d) + O(1)$ time. Therefore, checking $3d$ positions takes time $o(|P|) + O(d) = O(|P|)$ for a long pattern P.

We now describe how we can apply our algorithm in the case when the text length is not known beforehand. In this case, we assume $|T|$ to take increasing values $n_0 < n_1 < \dots$, as long as the text T keeps growing. Here, n_0 is some appropriate initial value and $n_i = 2n_{i-1}$ for $i \geq 1$.

Suppose now that n_i is the currently assumed value of $|T|$. After we reach character $t_{n_i/2}$, during the processing of the next $n_i/2$ symbols, we keep building the index for $|T| = n_i$ and, in parallel, rebuild all the data structures under assumption that $|T| = n_{i+1} = 2n_i$. In particular, if $\log\log(2n_i) \neq \log\log n_i$, we build a new index for long patterns, and if $\log^{(3)}(2n_i) \neq \log^{(3)} n_i$, we build a new index for meduim-size and short patterns. If $\log_\sigma(2n_i) \neq \log_\sigma n_i$, we also construct a new compact representation T_c introduced earlier in this section. Altogether, we distribute the construction cost of the data structures for $T[n_i..1]$ under assumption $|T| = 2n_i$ over the processing of $t_{n_i/2+1} \dots t_{n_i}$. Since $O(n_i) = O(n_i/2)$, processing these $n_i/2$ symbols remains real-time. By the time t_{n_i} has been read, all data structures for $|T| = 2n_i$ have been built, and the algorithm proceeds with the new value $|T| = n_{i+1}$. Observe finally that the intervals $[n_i/2+1, n_i]$ are all disjoint, therefore the overhead per letter incurred by the procedure remains constant. In conclusion, the whole algorithm remains real-time. We finish with our main result.

Theorem 1. *There exists a data structure storing a text T that can be updated in $O(1)$ worst-case expected time after prepending a new symbol to T. This data structure reports all occurrences of a pattern P in the current text T in $O(|P|+k)$ time, where k is the number of occurrences.*

5 Conclusions

In this paper we presented the first real-time indexing data structure that supports reporting all pattern occurrences in optimal time $O(|P| + k)$. As in the

[4] In fact, it would suffice to store $3d-1$ most recently read symbols in compact form.

previous works on this topic [11,2,4], we assume that the input text is over an alphabet of constant size. It may be possible to extend our result to alphabets of poly-logarithmic size.

Our algorithm spends a constant *expected* worst-case time for updating the data structure when a new text symbol arrives. The expectation comes only from the updates of the weighted level ancestor structure [10], which, in turn, comes from the updates for the dynamic predecessor problem (y-fast tries). We feel that one can get rid of the expectation, however we have not found a solution to this so far.

Acknowledgements. GK has been supported by the Marie-Curie Intra-European fellowship for carrier development. We thank the anonymous reviewers for helpful comments.

References

1. Amir, A., Kopelowitz, T., Lewenstein, M., Lewenstein, N.: Towards real-time suffix tree construction. In: Consens, M., Navarro, G. (eds.) SPIRE 2005. LNCS, vol. 3772, pp. 67–78. Springer, Heidelberg (2005)
2. Amir, A., Nor, I.: Real-time indexing over fixed finite alphabets. In: Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008), pp. 1086–1095 (2008)
3. Breslauer, D., Grossi, R., Mignosi, F.: Simple real-time constant-space string matching. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 173–183. Springer, Heidelberg (2011)
4. Breslauer, D., Italiano, G.F.: Near real-time suffix tree construction via the fringe marked ancestor problem. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) SPIRE 2011. LNCS, vol. 7024, pp. 156–167. Springer, Heidelberg (2011)
5. Franceschini, G., Grossi, R.: A General Technique for Managing Strings in Comparison-Driven Data Structures. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 606–617. Springer, Heidelberg (2004)
6. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. 48(3), 533–551 (1994)
7. Galil, Z.: String matching in real time. J. ACM 28(1), 134–149 (1981)
8. Giyora, Y., Kaplan, H.: Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. ACM Transactions on Algorithms 5(3) (2009)
9. Kopelowitz, T.: On-line indexing for general alphabets via predecessor queries on subsets of an ordered list. In: Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), pp. 283–292 (2012)
10. Kopelowitz, T., Lewenstein, M.: Dynamic weighted ancestors. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 565–574 (2007)
11. Kosaraju, S.R.: Real-time pattern matching and quasi-real-time construction of suffix trees (preliminary version). In: Proc. 26th Annual ACM Symposium on Theory of Computing (STOC 1994), pp. 310–316. ACM (1994)

12. Kucherov, G., Nekrich, Y., Starikovskaya, T.: Cross-document pattern matching. In: Kärkkäinen, J., Stoye, J. (eds.) CPM 2012. LNCS, vol. 7354, pp. 196–207. Springer, Heidelberg (2012)
13. Mortensen, C.W.: Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), pp. 618–627 (2003)
14. Navarro, G., Nekrich, Y.: Top-k document retrieval in optimal time and linear space. In: Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), pp. 1066–1077 (2012)
15. Slisenko, A.: String-matching in real time: Some properties of the data structure. In: Winkowski, J. (ed.) MFCS 1978. LNCS, vol. 64, pp. 493–496. Springer, Heidelberg (1978)

Arithmetic Circuit Lower Bounds via MaxRank*

Mrinal Kumar[1], Gaurav Maheshwari[2], and Jayalal Sarma M.N.[2]

[1] Department of Computer Science, Rutgers University, Piscataway, NJ 08855, USA
[2] Department of Computer Science & Engineering, IIT Madras, Chennai 36, India

Abstract. We introduce the polynomial coefficient matrix and identify maximum rank of this matrix under variable substitution as a complexity measure for multivariate polynomials. We use our techniques to prove super-polynomial lower bounds against several classes of non-multilinear arithmetic circuits. In particular, we obtain the following results :

- As our first main result, we prove that any homogeneous depth-3 circuit for computing the product of d matrices of dimension $n \times n$ requires $\Omega(n^{d-1}/2^d)$ size. This improves the lower bounds in [9] for $d = \omega(1)$.
- As our second main result, we show that there is an explicit polynomial on n variables and degree at most $\frac{n}{2}$ for which any depth-3 circuit C of product dimension at most $\frac{n}{10}$ (dimension of the space of affine forms feeding into each product gate) requires size $2^{\Omega(n)}$. This generalizes the lower bounds against diagonal circuits proved in [14]. Diagonal circuits are of product dimension 1.
- We prove a $n^{\Omega(\log n)}$ lower bound on the size of product-sparse formulas. By definition, any multilinear formula is a product-sparse formula. Thus, this result extends the known super-polynomial lower bounds on the size of multilinear formulas [11].
- We prove a $2^{\Omega(n)}$ lower bound on the size of partitioned arithmetic branching programs. This result extends the known exponential lower bound on the size of ordered arithmetic branching programs [7].

1 Introduction

Arithmetic circuits are a fundamental model of computation for polynomials. Establishing the limitations of polynomial sized arithmetic circuits is a central open question in the area of algebraic complexity(see [17] for a detailed survey). One of the recent surprises in the area was the result due to Agrawal and Vinay [2] where they show that if a polynomial in n variables of degree d (linear in n) can be computed by arithmetic circuits of size $2^{o(n)}$, then it can also be computed by depth-4 circuits of size $2^{o(n)}$. The parameters of this result was further tightened by Koiran [8]. These results explained the elusiveness of proving lower bounds against even depth-4 circuits. For depth-3 circuits, the best known general result (over finite fields) is an exponential lower bound due to Grigoriev

* The full version of the paper is available as a technical report at the ECCC. Technical report No. ECCC-TR-13-028. See `http://eccc.hpi-web.de/report/2013/028/`

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 661–672, 2013.

and Karpinski [5] and Grigoriev and Razborov [4]. Over infinite fields, obtaining such strong lower bounds is a long-standing open problem. Lower bounds for restricted classes of depth-3 and depth-4 circuits are studied in [1,9,16] .

One class of models which has been extensively studied is when the gates are restricted to compute multilinear polynomials. Super-polynomial lower bounds are known for the size of multilinear formulas computing the permanent or determinant polynomial [12]. However, even under this restriction proving super-polynomial lower bounds against arbitrary multilinear arithmetic circuits is an open problem (see [17] and references there in). The parameter identified by [11], which showed the limitations of multilinear formulas, was the rank of a matrix associated with the circuit - namely the partial derivatives matrix[1]. The method showed that there exists a partition of variables into two sets such that the rank of the partial derivatives matrix of any polynomial computed by the model is upper bounded by a function of the size of the circuit. But there are explicit polynomials for which the rank of the partial derivatives matrix is high. This program has been carried out for several classes of multilinear polynomials and several variants of multilinear circuits [3,7,10,11,12,13]. However, the partial derivatives matrix, in the form that was studied, was known to yield lower bounds only for multilinear circuits.

In this work, we generalize this framework to prove lower bounds against several classes of non-multilinear arithmetic circuits. This generalization also shows that the multilinearity restriction in the above proof strategy can possibly be eliminated from the circuit model side. Hence it can also be seen as an approach towards proving lower bounds against the general arithmetic circuits.

We introduce a variant of the partial derivatives matrix where the entries will be polynomials instead of constants - which we call the *polynomial coefficient matrix*. Instead of rank of the partial derivatives matrix, we analyze the max-rank - the maximum rank of the polynomial coefficient matrix[2] under any substitution for the variables from the underlying field. We first prove how the max-rank changes under arithmetic operations. These tools are combined to prove upper bounds on max-rank of various restrictions of arithmetic circuits.

In [9], it was proved that any homogeneous depth-3 circuit for multiplying d $n \times n$ matrices (Iterated Matrix Multiplication, IMM_d^n) requires $\Omega\left(n^{d-1}/d!\right)$ size. We use our techniques to improve this result in terms of the lower bound. Our methods are completely different from [9] and this demonstrates the power of this method beyond the reach of the original partial derivatives matrix method due to Raz [11]. As our first main result, we prove the following.

Theorem 1. *Any homogeneous depth-3 circuit for computing the product of d matrices of dimension $n \times n$ requires $\Omega(n^{d-1}/2^d)$ size.*

[1] An exponential sized matrix associated with the multilinear polynomial with respect to a partition of the variables into two sets. See Section 2 for the formal definition.

[2] When it is clear from the context, we drop the matrix as well as the partition. By the term, max-rank of a polynomial, we denote the maximum rank of the polynomial coefficient matrix corresponding to the polynomial with respect to the partition in the context.

Notice that compared to the bounds in [9], our bounds are stronger when $d = \omega(1)$. Very recently, Gupta et al. [6] studied the model of homogeneous circuits and proved a strong lower bound parameterized by the bottom fan-in. They studied depth-4 circuits ($\Sigma\Pi\Sigma\Pi$) and showed that if the fan-in of the bottom level product gate of the circuits is t, then any homogeneous depth-4 circuit computing the permanent (and the determinant) of $n \times n$ matrices must have size $2^{\Omega(\frac{n}{t})}$. In particular, this implies a $2^{\Omega(n)}$ lower bound for any depth-3 homogeneous circuit computing the permanent (and the determinant) of $n \times n$ matrices (n^2 variables). However, we remark that Theorem 1 is addressing the iterated matrix multiplication polynomial and hence is not directly subsumed by the above result. Moreover, the techniques used in [6] are substantially different from ours.

We apply our method to depth-3 circuits where space of the affine forms feeding into each product gate in the circuit is of limited dimension. Formally, a depth-3 $\Sigma\Pi\Sigma$ circuit C is said to be of product dimension r if for each product gate P in C, where $P = \Pi_{i=1}^{d} L_i$, where L_i is an affine form for each i, the dimension of the span of the set $\{L_i\}_{i\in[d]}$ is at most r. As our second main result, we prove exponential lower bounds on the size (in fact, the top fan in) of depth-3 circuits of bounded product dimension for computing an explicit polynomial.

Theorem 2. *There is an explicit polynomial on n variables and degree $\leq \frac{n}{2}$ for which any $\Sigma\Pi\Sigma$ circuit C of product dimension at most $\frac{n}{10}$ requires size $2^{\Omega(n)}$.*

In [14], the author studies diagonal circuits, which are depth-3 circuits where each product gate is an exponentiation gate. Clearly, such a product gate can be visualized as a product gate with the same affine form being fed into it multiple times. Thus, these circuits are of product dimension 1, and our lower bound result generalizes size lower bounds against diagonal circuits.

Note that the product dimension of a depth-3 circuit is different from the dimension of the span of all affine forms computed at the bottom sum gates of a $\Sigma\Pi\Sigma$ circuit. It can be easily seen that, when this parameter, which we refer to as the total dimension of the circuit, when bounded, the model non-universal.

For our next result, we generalize the model of syntactic multilinear formulas to product-sparse formulas. We formally define product-sparse formulas and full max-rank polynomials in Section 2. These formulas can compute non-multilinear polynomials as well. We show the following theorem regarding this model using our methods.

Theorem 3. *Let X be a set of $2n$ variables and let $f \in \mathbb{F}[X]$ be a full max-rank polynomial. Let Φ be any (s, d)-product-sparse formula of size $n^{\epsilon \log n}$, for a constant ϵ. If $sd = o(n^{1/8})$, then f cannot be computed by Φ.*

As our fourth result, we define partitioned arithmetic branching programs which are generalizations of ordered ABPs. We prove an exponential lower bound for partitioned ABPs extending results in [7].

Theorem 4. *Let X be a set of $2n$ variables and $\mathbb{F}$ be a field. For any full max-rank homogeneous polynomial f of degree n over X and $\mathbb{F}$, the size of any partitioned ABP computing f must be $2^{\Omega(n)}$.*

2 Preliminaries

In this section, we define some of the models we study. For more detailed account of models and the results we refer the reader to the survey [17].

An arithmetic circuit Φ over the field $\mathbb{F}$ and the set of variables $X = \{x_1, x_2, \ldots, x_n\}$ is a directed acyclic graph $G = (V, E)$. The vertices of G with in-degree 0 are called *input* gates and are labelled by variables in X or constants from the field $\mathbb{F}$. The vertices of G with out-degree 0 are called *output* gates. Rest all vertices are referred to as internal vertices. Every internal vertex is either a plus gate or a product gate. We will study arithmetic circuits with a single output gate. Thus, the polynomial computed by the arithmetic circuit is the polynomial associated with the output gate. The size of Φ is defined to be the number of gates in Φ. For a vertex $v \in V$, we denote the set of variables that occur in the subgraph rooted at v by X_v.

We consider depth restricted circuits. A $\Sigma\Pi\Sigma$ circuit is a levelled depth-3 circuit with a plus gate at the top, multiplication gates at the middle level and plus gates at the bottom level. The fan-in of the top plus gate is referred to as top fan-in. A $\Sigma\Pi\Sigma$ circuit is said to be *homogeneous* if the plus gate at the bottom level compute homogeneous linear forms only.

An important restricted model of arithmetic circuits is multilinear circuits. A polynomial $f \in \mathbb{F}[X]$ is called *multilinear* if the degree of every variable in f is at most one. An arithmetic circuit is called *multilinear* if the polynomial computed at every gate is multilinear. An arithmetic circuit is called *syntactic multilinear* if for every product gate v with children v_1 and v_2, $X_{v_1} \cap X_{v_2} = \phi$. An arithmetic circuit is called an *arithmetic formula* if the underlying undirected graph is acyclic i.e. fan-out of every vertex is at most one.

Let Φ be a formula defined over the set of variables X and a field $\mathbb{F}$. For a product gate v in Φ with children v_1 and v_2, let us define the following properties:

Disjoint v is said to be *disjoint* if $X_{v_1} \cap X_{v_2} = \phi$.

Sparse v is said to be *s-sparse* if the number of monomials in the polynomial computed by at least one of its input gates is at most 2^s.

For a node v, let us define the product-sparse depth of v to be equal to the maximum number of non-disjoint product gates in any path from a leaf to v.

Definition 1. *A formula is said to be a (s, d)-product-sparse if every product gate v is either disjoint or s-sparse, where d is the product-sparse depth of the root node.*

Clearly, any syntactic multilinear formula is a $(s, 0)$-product-sparse formula for any s. Thus, proving lower bounds for product-sparse formulas will be a strengthening of known results.

An Arithmetic Branching Program (ABP) B is a levelled graph $G(V, E)$ in which V can be partitioned into levels $L_0, L_1, \ldots, L_d$ such that $L_0 = \{s\}$ and $L_d = \{t\}$ and edges can only go between consecutive levels. s and t are called the *source* and *sink* respectively. The weight function w assigns affine forms to E. For a path p, extend the weight function by $w(p) = \prod_{e \in p} w(e)$. B computes

the polynomial $\sum_p w(p)$ where p runs over all source-sink paths. B is said to be *homogeneous* if all edge labels are homogeneous linear forms and naturally computes a homogeneous polynomial. For any $i, j \in V$, $P_{i,j}$ denotes all paths from i to j in G, $X_{i,j}$ denotes the variables occuring in those paths and $f_{i,j}$ denotes the polynomial $\sum_{p \in P_{i,j}} w(p)$.

Definition 2. *Let B be a homogeneous ABP over a field $\mathbb{F}$ and set of variables $X = \{x_1, x_2, \ldots, x_{2n}\}$. B is said to be π-partitioned for a permutation $\pi : [2n] \to [2n]$ if there exists an $i = 2\alpha n$ for some constant α such that the following condition is satisfied, $\forall v \in L_i$:*

- *Either, $X_{s,v} \subseteq \{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\}$ and $|X_{v,t}| \leq 2n(1-\alpha)$.*
- *Or, $X_{v,t} \subseteq \{x_{\pi(n+1)}, x_{\pi(n+2)}, \ldots, x_{\pi(2n)}\}$ and $|X_{s,v}| \leq 2n(1-\alpha)$*

We say that B is partitioned with respect to the level L_i. B is said to be a partitioned ABP if it is π-partitioned for some $\pi : [2n] \to [2n]$.

We now introduce the main tool used in the paper and prove its properties. Let $Y = \{y_1, y_2, \ldots, y_m\}$ and $Z = \{z_1, z_2, \ldots, z_m\}$ be two sets of variables. Let $f \in \mathbb{F}[Y, Z]$ be a multilinear polynomial. Define L_f to be the $2^m \times 2^m$ *partial derivatives matrix* as follows: for monic multilinear monomials $p \in \mathbb{F}[Y], q \in \mathbb{F}[Z]$, define $L_f(p, q)$ to be the coefficient of the monomial pq in f. Let us denote the rank of L_f by $\text{rank}(L_f)$. We extend the partial derivatives matrix to non-multilinear polynomials.

Definition 3 (Polynomial Coefficient Matrix). *For $f \in \mathbb{F}[Y, Z]$, define M_f to be the $2^m \times 2^m$* polynomial coefficient matrix *with each entry from $\mathbb{F}[Y, Z]$ defined as follows. For monic multilinear monomials p and q in Y and Z respectively, $M_f(p, q) = G$ if and only if f can be uniquely written as $f = pq(G) + Q$, where $G, Q \in \mathbb{F}[Y, Z]$ such that G does not contain any variable other than those present in p and q, Q does not have any monomial m which is divisible by pq and which contains only variables that are present in p and q.*

Observe that we can write, $f = \sum_{p,q} M_f(p, q)pq$ and for a multilinear polynomial f, M_f is same as L_f. For any function $S : Y \cup Z \to \mathbb{F}$, let us denote by $M_f|_S$ the matrix obtained by substituting each variable x by $S(x)$ at each entry in M_f. Let us define $\text{max-rank}(M_f) = \max_{S:Y \cup Z \to \mathbb{F}} \{\text{rank}(M_f|_S)\}$. The following proposition bounds the max-rank of the matrix (similar bounds on the rank of partial derivatives matrix for some cases have been proved in [13]). We defer the proof to the full version of the paper.

Proposition 1. *Let $f, g \in \mathbb{F}[Y, Z]$, $h \in \mathbb{F}[Y]$ and $w \in F[Z]$.*

1.1 If f contains variables $Y' \subseteq Y$ and $Z' \subseteq Z$ only, then $\text{max-rank}(M_f) \leq 2^a$ *where $a = \min\{|Y'|, |Z'|\}$.*

1.2 $\text{max-rank}(M_{f+g}) \leq \text{max-rank}(M_f) + \text{max-rank}(M_g)$.

1.3 Let $Y_1, Y_2 \subseteq Y$ and $Z_1, Z_2 \subseteq Z$ such that $Y_1 \cap Y_2 = \phi$ and $Z_1 \cap Z_2 = \phi$. If $f \in \mathbb{F}[Y_1, Z_1]$ and $g \in \mathbb{F}[Y_2, Z_2]$, then $\text{max-rank}(M_{fg}) = \text{max-rank}(M_f) \cdot \text{max-rank}(M_g)$.

1.4 max-rank$(M_{fh}) \leq$ max-rank(M_f) *and* max-rank$(M_{fw}) \leq$ max-rank(M_f).
1.5 If g *is a linear form, then* max-rank$(M_{fg}) \leq 2 \cdot$ max-rank(M_f).
1.6 If g *can be expressed as* $\sum_{i\in[r]} h_i w_i$ *where* $h_i \in \mathbb{F}[Y]$ *and* $w_i \in \mathbb{F}[Z]$, *then* max-rank$(M_{fg}) \leq r \cdot$ max-rank(M_f).
1.7 If g *has* r *monomials, then* max-rank$(M_{fg}) \leq r \cdot$ max-rank(M_f).

Full Rank Polynomials: Let $X = \{x_1, \cdots, x_{2n}\}, Y = \{y_1, \cdots, y_n\}$ and $Z = \{z_1, \cdots, z_n\}$ be sets of variables and $f \in \mathbb{F}[X]$. f is said to be a *full rank* polynomial if for any partition $A : X \to Y \cup Z$, rank$(L_{f^A}) = 2^n$, where f^A is the polynomial obtained from f after substituting every variable x by $A(x)$. We say that f is a *full max-rank* polynomial if max-rank$(M_{f^A}) = 2^n$ for any partition A. Any full rank polynomial is also a full max-rank polynomial. Many full rank polynomials have been studied in the literature [7,11,12].

3 Lower Bounds against Homogeneous Depth-3 Circuits

Let Φ be a homogeneous $\Sigma\Pi\Sigma$ circuit with top fan-in k defined over the set of variables X and field $\mathbb{F}$ computing a homogeneous polynomial $f = \sum_{i=1}^{k} P_i$, where $P_i = \prod_{j=1}^{deg(P_i)} l_{i,j}$, each $l_{i,j}$ is a linear form and $deg(P_i)$ is the fan-in of the i^{th} multiplication gate. For a partition $A : X \to Y \cup Z$, denote by Φ^A the circuit obtained after replacing every variable x by $A(x)$ and the corresponding polynomial by f^A. We prove the following upper bound on the max-rank(M_{f^A}).

Lemma 1. *Let* Φ *be a homogeneous* $\Sigma\Pi\Sigma$ *circuit as defined above and the degree of* f *be* d. *Then, for any partition* $A : X \to Y \cup Z$, max-rank$(M_{f^A}) \leq k \cdot 2^d$.

Proof. Let us denote by $l_{i,j}^A$ and P_i^A the polynomials obtained after substitution of x by $A(x)$ in the polynomials $l_{i,j}$ and P_i respectively.
Since each $l_{i,j}$ is a homogeneous linear form, a multiplication gate P_i computes a homogeneous polynomial of degree $deg(P_i)$. Thus if $deg(P_i) \neq d$ then the multiplication gate P_i does not contribute any monomial in the output polynomial f. Hence, it can be assumed without loss of generality that $deg(P_i) = d$ for all $i \in [k]$.

Since $l_{i,j}$ is a homogeneous linear form, max-rank$(M_{l_{i,j}^A}) \leq 2$. Thus, using Proposition 1.5,$\forall i \in [k]$: max-rank$(M_{P_i^A}) \leq 2^d$. Hence, using Proposition 1.2, max-rank$(M_{f^A}) \leq \sum_{i\in[k]}$ max-rank$(M_{P_i^A}) \leq k \cdot 2^d$.

In [9], it was proved that any homogeneous $\Sigma\Pi\Sigma$ circuit for multiplying d $n \times n$ matrices requires $\Omega(n^{d-1}/d!)$ size. We prove a better lower bound using our techniques. Formally, let $X^1, X^2, \ldots, X^d$ be disjoint sets of variables of size n^2 each, with $X = \cup_{i\in[d]} X^i$. The variables in X^i will be denoted by x_{jk}^i for $j, k \in [n]$. We will be looking at the problem of multiplying d $n \times n$ matrices $A^1, A^2, \ldots, A^d$

where $(j,k)^{th}$ entry of matrix A^i, denoted by A^i_{jk}, is defined to be equal x^i_{jk} for all $i \in [d]$ and $j,k \in [n]$. The output polynomial, that we are interested in, is the $(1,1)^{th}$ entry of $\prod_{i\in[d]} A^i$ denoted by f. We also refer to f by IMM^n_d. f is clearly a homogeneous multilinear polynomial of degree d. Moreover, any monomial in f contains one variable each from the sets $X^1, X^2, \ldots, X^d$.

We first prove an important lemma below.

Lemma 2. *For the polynomial f as defined above, there exists a bijective partition $B : X \to Y \cup Z$ such that* $\text{max-rank}(M_{f^B}) = n^{d-1}$.

Proof. We fix some notations first. For $i < j$, let us denote the set $\{i, i+1, \ldots, j\}$ by $[i,j]$. Let us also denote the pair $((k,i),(k+1,j))$ by e_{ijk} for any i,j,k. Construct a directed graph $G(V,E)$ on the set of vertices $V = [0,d] \times [1,n]$ and consisting of edges $E = \{e_{ijk} \mid k \in [0,d-1], i,j \in [1,n]\}$. Note that the edges e_{ijk} and e_{jik} are two distinct edges for fixed values of i,j,k when $i \neq j$. Let us also define a weight function $w : E \to X$ such that $w(e_{ijk}) = x^{k+1}_{ij}$.

It is easy to observe that the above graph encodes the matrices $A_1, A_2, \ldots, A_d$. The weights on the edges are the variables in the matrices. For example, a variable x^{k+1}_{ij} in the matrix A_{k+1} is the weight of the edge e_{ijk}. Let us denote the set of paths in G from the vertex $(0,1)$ to the vertex $(d,1)$ by $\mathcal{P}$. Let us extend the weight function and define $w(p) = \prod_{e\in p} w(e)$ for any $p \in \mathcal{P}$. Since, all paths in $\mathcal{P}$ are of length equal to d, the weights corresponding to each of these paths are monomials of degree d.

Let us define the partition $B : X \to Y \cup Z$ as follows: all the variables in odd numbered matrices are assigned variables in Y and all the variables in even numbered matrices are assigned variables in Z. Let us denote the variable assigned by B to x^{2k-1}_{ij} by y^{2k-1}_{ij} and the variable assigned to x^{2k}_{ij} by z^{2k}_{ij}.

It follows from the matrix multiplication properties that for any path $p \in \mathcal{P}$, the monomial $w(p)$ is a monomial in the output polynomial. Each such path is uniquely specified once we specify the odd steps in the path. Now, specifying odd steps in the path corresponds to specifying a variable from each of the odd numbered matrices. To count number of such ways, let us first consider the case when d is even. There are $d/2$ odd numbered matrices and we have n^2 ways to choose a variable from each of these $d/2$ matrices except for the first matrix for which we can only choose a variable from the 1^{st} row since our output polynomial is the $(1,1)^{th}$ entry. Thus, there are n^{d-1} number of ways to specify one variable each from the odd numbered matrices, the number of such paths is also n^{d-1}. We get the same count for the case when d is odd using a similar argument. Since once the odd steps are chosen, there is only one way to choose the even steps, all these n^{d-1} monomials give rise to non-zero entries in different rows and columns in the matrix M_{f^B}. Hence, the matrix is an identity block of dimension n^{d-1} upto a permutation of rows and columns and thus it has rank n^{d-1}.

Theorem 5. *Any homogeneous $\Sigma\Pi\Sigma$ circuit for computing the product of d $n \times n$ matrices requires $\Omega(n^{d-1}/2^d)$ size.*

Proof. Let Φ be a homogeneous $\Sigma\Pi\Sigma$ circuit computing f. Then, using Lemma 1, for any partition A, max-rank$(M_{f^A}) \leq k \cdot 2^d$. From Lemma 2, we know that there exists a partition B such that max-rank$(M_{f^B}) = n^{d-1}$. Hence, $k \geq n^{d-1}/2^d$.

It is worth noting that there exists a depth-2 circuit of size n^{d-1} computing IMM_d^n polynomial. As observed in Lemma 2, there are n^{d-1} monomials in the IMM_d^n polynomial. Hence, the sum of monomials representation for IMM_d^n will have top fan-in equal to n^{d-1}. We remark that when the number of matrices is a constant, the upper and lower bounds for IMM_d^n polynomial asymptotically match.

4 Lower Bounds against Depth-3 Circuits of Bounded Product Dimension

If a depth-3 circuit is not homogeneous, the fan-in of a product gate can be arbitrarily larger than the degree of the polynomial being computed. Hence the techniques in the previous section fails to give non-trivial size lower bounds. In this section, we study depth-3 circuits with bounded product dimension - where the affine forms feeding into every product gate are from a linear vector space of small dimension and prove exponential size lower bounds for such circuits.

We will first prove an upper bound on the max-rank of the polynomial coefficient matrix for the polynomial computed by a depth-3 circuit of product dimension r, parameterized by r. Let C be a $\Sigma\Pi\Sigma$ circuit of product dimension r and top fan in k. Let P^j be the product gates in C for $j \in [k]$, given by $P^j = \Pi_{i=1}^s L_i^j$. Without loss of generality, let us assume that the vectors $L_1^j, L_2^j, \ldots, L_r^j$ form a basis for the span of $\{L_1^j, L_2^j, \ldots, L_s^j\}$. Let l_i^j be the homogeneous part of L_i^j for each i. So, clearly the set $\{l_i^j\}_{i\in[r']}$ spans the set $\{l_i^j\}_{i\in[s]}$, where $r' \leq r$. To simplify the notation, we will refer to r' as r. In the following presentation, we will always use d to refer to the degree of the homogeneous polynomial computed by the circuit under consideration. Now, let us express each l_i^j as a linear combination of $\{l_i^j\}_{i\in[r]}$. Let us expand the product P^j into a sum of product of homogeneous linear forms coming from $\{l_i^j\}_{i\in r}$. Let P_d^j be the slice of P^j of degree exactly d, for each $j \in [k]$.

Observation 6. *Let $C_d = \Sigma_{i\in[k]} P_d^i$. If C computes a homogeneous polynomial of degree d, then C_d computes the same polynomial.*

Proof. We know that, $C = \Sigma_{i\in[k]} P^i$. Now, writing each product gate as a sum of product of homogeneous linear forms as described in the paragraph above, we get $C = \Sigma_{i\in[k]} \Sigma_{j\in[d]} P_j^i$. Now, equating the degree d parts of the polynomial in both sides of the equality, we obtain $C_d = \Sigma_{i\in[k]} P_d^i$. If C computes a homogeneous polynomial of degree d, $C = C_d$ and the lemma follows.

We know that for each $P_d^j = \Sigma_i \Pi_{u=1}^d l_{\alpha_{iu}}$ where $\alpha_{iu} \in [r]$ and $j \in [k]$. We now use the following lemma to simplify the inner product terms $\Pi_{u=1}^d l_{\alpha_{iu}}$ in the expression for P_d^j.

Lemma 3. *([15]) Any monomial of degree d can be written as a linear combination of d^{th} power of some 2^d linear forms. Further, each of the 2^d linear forms in the expression corresponds to $\Sigma_{x\in S}x$ for a subset S of $[d]$.*

By applying to each product term $\Pi_{u=1}^{d} l_{\alpha_{iu}}$ in P_d^j, we obtain the following:

Lemma 4. *If $P_d^j = \Sigma_i \Pi_{u=1}^{d} l_{\alpha_{iu}}$ where $\alpha_{iu} \in [r]$, then $P_d^j = \Sigma_{q=1}^{v} c_q {L_q}^d$ for some homogeneous linear forms L_q, constants c_q and $v \leq \binom{d+r}{r}$.*

Proof. Consider any product term in the sum of products expansion P_d^j as described, say $S = \Pi_{u=1}^{d} l_{\alpha_{iu}}$. From Lemma 3, we know that S can be written as $S = \Sigma_{t=1}^{2^d} {L_t}^d$, where for every subset U of $[d]$, there is a $\beta \in [2^d]$ such that $L_\beta = \Sigma_{u\in U} l_{\alpha_{iu}}$. In general, each L_t can be written as $L_t = \Sigma_{i\in[r]} \gamma_i l_i$ for non-negative integers γ_i satisfying $\Sigma_{i\in[r]}\gamma_i \leq d$. Now, each of the product terms in P_d^j can be expanded in a similar fashion into d^{th} powers of linear forms, each from the set $\{\Sigma_{i\in[r]}\gamma_i l_i : \gamma_i \in \mathbb{Z}^{\geq 0} \wedge \Sigma_{i\in[r]}\gamma_i \leq d\}$. The number of distinct such linear forms is at most $\binom{d+r}{r}$. Hence, the lemma follows.

We now bound the max-rank of the power of a homogeneous linear form which in turn will give us a bound for P_d^j due to the subadditivity of max-rank.

Lemma 5. *Given a linear form l and any positive integer t, the* max-rank *of l^t is at most $t+1$ for any partition of the set X of variables into Y and Z.*

Proof. Partition the linear form l into two parts, $l = l_y + l_z$, where l_y consists of all variables in l from the set Y and l_z consists of the variables which come from the set Z. By the binomial theorem, $l^t = \Sigma_{i=0}^{t} \binom{t}{i} l_y^i l_z^{t-i}$. Now, l_y^i is a polynomial just in Y variables and hence its max-rank can be bounded above by 1, and multiplication by l_z^{t-i} does not increase the max-rank any further, by Proposition 1.4. Hence, the max-rank of each term in the sum is at most 1 and there are at most $t+1$ terms, so, by using the subadditivity of max-rank, we get an upper bound of $t+1$ on the max-rank of the sum.

The following lemma gives an upper bound on the max-rank of P_d^j and follows from Lemma 4, Lemma 5 and the subadditivity of max-rank.

Lemma 6. *The* max-rank *of P_d^j is at most $(d+1)\binom{d+r}{r}$ for any partition of the set X of variables into Y and Z.*

Now we are ready to prove the theorem.

Theorem 7. *There is an explicit polynomial in n variables and degree $\frac{n}{2}$ for which any $\Sigma\Pi\Sigma$ circuit C of product dimension at most $\frac{n}{10}$ requires size $2^{\Omega(n)}$.*

Proof. We describe the explicit polynomial $Q(X)$ first. Fix an equal sized partition A of X into Y and Z. Order all subsets of Y and Z of size exactly $\frac{n}{4}$ in any order, say $S_1, S_2, \ldots, S_w$ and $T_1, T_2, \ldots, T_w$, where $w = \binom{\frac{n}{2}}{\frac{n}{4}}$. Let us define the polynomial $Q^A(Y,Z)$ for the partition A as follows: $Q^A(Y,Z) = \Sigma_{i=1}^{w} \Pi_{y\in S_i} \Pi_{z\in T_i} yz$.

We obtain the polynomial $Q(X)$ by replacing variables in Y and Z in $Q^A(Y,Z)$ by $A^{-1}(Y)$ and $A^{-1}(Z)$ respectively.

Now we prove the size lower bound. The polynomial coefficient matrix of Q with respect to the partition Y and Z is simply the diagonal submatrix, and the rank is at least $\binom{\frac{n}{2}}{\frac{n}{4}} \geq \frac{2^{\frac{n}{2}}}{\sqrt{n}}$. Since C computes the polynomial, the top fan in k should be at least $\frac{\frac{2^{\frac{n}{2}}}{\sqrt{n}}}{\binom{d+r}{r}(d+1)}$. For $d = \frac{n}{2}$, and $r = \frac{n}{10}$, we have a lower bound of 2^{cn}, for a constant $c > 0$.

5 Lower Bounds against Product-Sparse Formulas

Let $Y = \{y_1, y_2, \ldots, y_m\}$ and $Z = \{z_1, z_2, \ldots, z_m\}$. Let Φ be a (s,d)-product-sparse formula defined over the field $\mathbb{F}$ and the variables $Y \cup Z$. For a node v, let us denote by Φ_v the sub-circuit rooted at v, and denote by Y_v and Z_v, the set of variables in Y and Z that appear in Φ_v respectively. Let us define, $a(v) = \min\{|Y_v|, |Z_v|\}$ and $b(v) = (|Y_v| + |Z_v|)/2$. We say that a node v is k-unbalanced if $b(v) - a(v) \geq k$. Let γ be a simple path from a leaf to the node v. We say that γ is k-unbalanced if it contains at least one k-unbalanced node. We say that γ is central if for every u, u_1 on the path γ such that there is an edge from u_1 to u in Φ, $b(u) \leq 2b(u_1)$. v is said to be k-weak if every central path that reaches v is k-unbalanced.

We prove that if v is k-weak then the max-rank of the matrix M_v can be bounded. The proof goes via induction on $|\Phi_v|$ and follows the same outline as that of [12]. It only differs in the case of non-disjoint product gates which we include in full detail below. The proofs of the rest of cases is easy to see.

Lemma 7. *Let Φ be a (s,d)-product-sparse formula over the set of variables $Y \cup Z$, and let v be a node in Φ. Denote the product-sparse depth of v by $d(v)$. If v is k-weak,* $\text{max-rank}(M_v) \leq 2^{s \cdot d(v)} \cdot |\Phi_v| \cdot 2^{b(v)-k/2}$.

Proof. Consider the case when v is a s-sparse product gate with children v_1 and v_2. Without loss of generality it can be assumed that v is not disjoint.

Let us suppose that the product-sparse depth of v is d. Without loss of generality, assume that v_2 computes a sparse polynomial having at most 2^s number of monomials. Thus using Proposition 1.7, $\text{max-rank}(M_v) \leq 2^s \cdot \text{max-rank}(M_{v_1})$ Clearly, product-sparse depth of v_1 is at most $d-1$. Consider the following cases: Case 1 : If $b(v) \leq 2b(v_1)$, then v_1 is also k-weak. Therefore, by induction hypothesis, $\text{max-rank}(M_{v_1}) \leq 2^{s(d-1)} \cdot |\Phi_{v_1}| \cdot 2^{b(v_1)-k/2} \leq 2^{s(d-1)} \cdot |\Phi_v| \cdot 2^{b(v)-k/2}$. Thus, $\text{max-rank}(M_v) \leq 2^{sd} \cdot |\Phi_v| \cdot 2^{b(v)-k/2}$. Case 2 : If $b(v) > 2b(v_1)$, then $b(v_1) < b(v)/2 < b(v) - k/2$ since $b(v) \geq k$. Therefore using Proposition 1.1, $\text{max-rank}(M_{v_1}) \leq 2^{a(v_1)} \leq 2^{b(v_1)} < 2^{b(v)-k/2}$. Therefore, $\text{max-rank}(M_v) \leq 2^s \cdot 2^{b(v)-k/2} \leq 2^{sd} \cdot |\Phi_v| \cdot 2^{b(v)-k/2}$.

Now, to prove a lower bound for (s,d)-product-sparse formulas computing a full max-rank polynomial, we only need to show that there exists a partition that makes the formula k-weak with suitable values of s, d and k.

In [11], Raz proved that for syntactic multilinear formulas of size at most $n^{\epsilon \log n}$, where ϵ is a small enough universal constant, there exists such a partition that makes the formula k-weak for $k = n^{1/8}$. We observe that this result also holds for product-sparse formulas, the proof given in [11] is not specific to just syntactic multilinear formulas and holds for any arithmetic formula. With above lemma, the following theorem is easy to derive.

Theorem 8. *Let X be a set of $2n$ variables and let $f \in \mathbb{F}[X]$ be a full max-rank polynomial. Let Φ be any (s,d)-product-sparse formula of size $n^{\epsilon \log n}$ for a constant ϵ (same as in [11]). If $sd = o(n^{1/8})$, then f cannot be computed by Φ.*

6 Lower Bounds against Partitioned Arithmetic Branching Programs

In the preliminaries section, we defined partitioned arithmetic branching programs which are a generalization of ordered ABPs. By definition, any polynomial computed by a partitioned ABP is homogenous. In [7], a full rank homogenous polynomial was constructed. Thus, to prove lower bounds for partitioned ABP, we only need to upper bound the max-rank of the polynomial coefficient matrix for any polynomial being computed by a partitioned ABP. Now we prove such an upper bound and use it to prove exponential lower bound on the size of partitioned ABPs, thus extending result in [7].

Theorem 9. *Let X be a set of $2n$ variables and $\mathbb{F}$ be a field. For any full max-rank homogenous polynomial f of degree n over X and $\mathbb{F}$, the size of any partitioned ABP computing f must be $2^{\Omega(n)}$.*

Proof. Let B be a π-partitioned ABP computing f for a permutation $\pi : [2n] \to [2n]$. Let $L_0, L_1, \ldots, L_n$ be the levels of B. Consider any partition A that assigns all n y-variables to $\{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\}$ and all n z-variables to $\{x_{\pi(n+1)}, x_{\pi(n+2)}, \ldots, x_{\pi(2n)}\}$. Let us denote by f^A the polynomial obtained from f after substituting each variable x by $A(x)$. Let B is partitioned with respect to the level L_i for $i = 2\alpha n$. We can write, $f = f_{st} = \sum_{v \in L_i} f_{s,v} f_{v,t}$. Consider a node $v \in L_i$. By definition, there are following two cases:
Case 1: $X_{s,v} \subseteq \{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\}$ and $|X_{v,t}| \leq 2n(1-\alpha)$. Thus, $f^A_{s,v} \in \mathbb{F}[Y]$. Hence, using Proposition 1.4 and 1.1,
$\text{max-rank}(M_{f^A_{s,v} f^A_{v,t}}) \leq \text{max-rank}(M_{f^A_{v,t}}) \leq 2^{|X_{v,t}|/2} \leq 2^{n(1-\alpha)}$
Case 2: $X_{v,t} \subseteq \{x_{\pi(n+1)}, x_{\pi(n+2)}, \ldots, x_{\pi(2n)}\}$ and $|X_{s,v}| \leq 2n(1-\alpha)$. Thus, $f^A_{v,t} \in \mathbb{F}[Z]$. Hence, again using Proposition 1.4 and 1.1,
$\text{max-rank}(M_{f^A_{s,v} f^A_{v,t}}) \leq \text{max-rank}(M_{f^A_{s,v}}) \leq 2^{|X_{s,v}|/2} \leq 2^{n(1-\alpha)}$ Thus, in any case, $\text{max-rank}(M_{f^A_{s,v} f^A_{v,t}}) \leq 2^{n(1-\alpha)}$ for all $v \in L_i$. Using Proposition 1.2, $\text{max-rank}(M_{f^A}) \leq |L_i| \cdot 2^{n(1-\alpha)}$. Since f is a full max-rank polynomial, we get $|L_i| \geq 2^{\alpha n}$.

References

1. Agrawal, M., Saha, C., Saptharishi, R., Saxena, N.: Jacobian Hits Circuits: Hitting-sets, Lower Bounds for Depth-d Occur-k Formulas & Depth-3 Transcendence Degree-k Circuits. In: Proceedings of ACM Symposium on Theory of Computing (STOC), pp. 599–614 (2012)
2. Agrawal, M., Vinay, V.: Arithmetic Circuits: A Chasm at Depth Four. In: Proceedings of Symposium on Foundations of Computer Science (FOCS), pp. 67–75 (2008)
3. Dvir, Z., Malod, G., Perifel, S., Yehudayoff, A.: Separating Multilinear Branching Programs and Formulas. In: Proceedings of Symposium on Theory of Computing (STOC), pp. 615–624 (2012)
4. Grigoriev, D., Razborov, A.: Exponential Complexity Lower Bounds for Depth 3 Arithmetic Circuits in Algebras of Functions over Finite Fields. In: Proceedings of Symposium on Foundations of Computer Science (FOCS), pp. 269–278 (1998)
5. Grigoriev, D., Karpinski, M.: An Exponential Lower Bound for Depth 3 Arithmetic Circuits. In: Proceedings of Symposium on Theory of Computing (STOC), pp. 577–582 (1998)
6. Gupta, A., Kamath, P., Kayal, N., Saptharishi, R.: Approaching the chasm at depth four. To Appear in Conference on Computational Complexity (2013)
7. Jansen, M.J.: Lower Bounds for Syntactically Multilinear Algebraic Branching Programs. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 407–418. Springer, Heidelberg (2008)
8. Koiran, P.: Arithmetic Circuits: The Chasm at Depth Four Gets Wider. Theor. Comput. Sci. 448, 56–65 (2012)
9. Nisan, N., Wigderson, A.: Lower Bounds on Arithmetic Circuits via Partial Derivatives. In: Proceedings of Symposium on Foundations of Computer Science (FOCS), pp. 16–25 (1995)
10. Raz, R., Yehudayoff, A.: Lower Bounds and Separations for Constant Depth Multilinear Circuits. In: Proceedings of Conference on Computational Complexity, pp. 128–139 (June 2008)
11. Raz, R.: Separation of Multilinear Circuit and Formula Size. Theory of Computing 2(1), 121–135 (2006)
12. Raz, R.: Multi-linear Formulas for Permanent and Determinant are of Super-polynomial Size. Journal of ACM 56, 8:1–8:17 (2009)
13. Raz, R., Shpilka, A., Yehudayoff, A.: A Lower Bound for the Size of Syntactically Multilinear Arithmetic Circuits. SIAM Journal of Computing 38(4), 1624–1647 (2008)
14. Saxena, N.: Diagonal Circuit Identity Testing and Lower Bounds. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 60–71. Springer, Heidelberg (2008)
15. Shpilka, A.: Affine Projections of Symmetric Polynomials. In: Proceedings of Conference on Computational Complexity, pp. 160–171 (2001)
16. Shpilka, A., Wigderson, A.: Depth-3 Arithmetic Circuits over Fields of Characteristic Zero. Computational Complexity 10(1), 1–27 (2001)
17. Shpilka, A., Yehudayoff, A.: Arithmetic Circuits: A Survey of Recent Results and Open Questions. Foundations and Trends in Theoretical Computer Science 5(3-4), 207–388 (2010)

Model Checking Lower Bounds for Simple Graphs

Michael Lampis*

KTH Royal Institute of Technology
mlampis@kth.se

Abstract. A well-known result by Frick and Grohe shows that deciding FO logic on trees involves a parameter dependence that is a tower of exponentials. Though this lower bound is tight for Courcelle's theorem, it has been evaded by a series of recent meta-theorems for other graph classes. Here we provide some additional non-elementary lower bound results, which are in some senses stronger. Our goal is to explain common traits in these recent meta-theorems and identify barriers to further progress.

More specifically, first, we show that on the class of threshold graphs, and therefore also on any union and complement-closed class, there is no model-checking algorithm with elementary parameter dependence even for FO logic. Second, we show that there is no model-checking algorithm with elementary parameter dependence for MSO logic even restricted to paths (or equivalently to unary strings), unless EXP=NEXP. As a corollary, we resolve an open problem on the complexity of MSO model-checking on graphs of bounded max-leaf number. Finally, we look at MSO on the class of colored trees of depth d. We show that, assuming the ETH, for every fixed $d \geq 1$ at least $d+1$ levels of exponentiation are necessary for this problem, thus showing that the $(d+1)$-fold exponential algorithm recently given by Gajarský and Hliněný is essentially optimal.

1 Introduction

Algorithmic meta-theorems are general statements establishing tractability for a whole class of problems (often defined by expressibility in a certain logic) in some class of inputs (usually a family of graphs). The most famous theorem in this area is one due to Courcelle [2] stating that all problems expressible in monadic second-order logic (MSO_2) are linear-time solvable on graphs of bounded treewidth. Much work has been devoted in recent years to proving stronger and stronger meta-theorems in this spirit, often extending Courcelle's theorem to other graph classes (see e.g. [3,7,5] or [12,13] for some great surveys).

The most often cited drawback of Courcelle's theorem has to do with the "hidden constant" in the algorithm's linear running time. It is clear that the running time must somehow depend on the input formula and the graph's treewidth, but the dependence given in Courcelle's theorem is in the worst case a tower of exponentials whose height grows with the size of the formula. Unfortunately,

* Research supported by ERC Grant 226203. See http://arxiv.org/abs/1302.4266 for full version containing omitted proofs.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 673–683, 2013.

this cannot be avoided: Frick and Grohe [8] proved that the parameter dependence has to be non-elementary even if one restricts the problem severely by just looking at properties expressible in first-order logic on trees.

This lower bound result, though quite devastating, has proven very fruitful and influential: several papers have appeared recently with the explicit aim of proving meta-theorems which evade it, and thus achieve a much better dependence on the parameters. Specifically, in [15] algorithmic meta-theorems with an elementary parameter dependence are shown for vertex cover, max-leaf number and the newly defined neighborhood diversity. A meta-theorem for twin cover was shown by Ganian [10]. In addition, meta-theorems were shown for tree-depth by Gajarský and Hliněný [9] and for the newly defined shrub-depth (which generalizes neighborhood diversity and twin cover) by Ganian et al. [11].

Thus, together with improved meta-theorems, these papers give a new crop of graph complexity measures, some more general than others. It becomes a natural question how much progress we can hope to achieve this way, that is, how far this process of defining more and more general "graph widths" can go on before hitting some other natural barrier that precludes an elementary parameter dependence. Is simply avoiding the class of all trees enough?

This is exactly the question we try to answer in this paper. Towards this end we try to give hardness results for graph families which are as simple as possible. Perhaps most striking among them is a result showing that not only is avoiding all trees not enough but in fact it is necessary to avoid the much smaller class of uncolored paths if one hopes for an elementary parameter dependence. As an example application, this almost immediately rules out the existence of meta-theorems with elementary parameter dependence in any induced-subgraph-closed graph class with unbounded diameter and any edge-subdivision-closed graph class. This explains why all recently shown meta-theorems we mentioned work on classes which are closed under induced subgraphs but have bounded diameter and are not closed under edge subdivisions.

Our results can be summarized as follows. First, a non-elementary lower bound for model checking FO logic on threshold graphs is shown. In a sense, this is a natural analogue of the lower bound for trees to the realm of clique-width, since threshold graphs are known to have the smallest possible clique-width. The proof is relatively simple and consists mostly of translating a similar lower bound given in [8] for FO model checking on binary words. However, the main interest of this result is that as a corollary we show that the complexity of FO model checking is non-elementary for any graph class closed under disjoint union and complement. This explains why, though some of the recent meta-theorems work on complement-closed graph classes (e.g. neighborhood diversity, shrub-depth) and some work on union-closed graph classes (e.g. tree-depth), no such meta-theorem has been shown for a class that has both properties.

Our second result is that model checking MSO logic on uncolored paths (or equivalently on unary strings) has a non-elementary parameter dependence. This is the most technically demanding of the results of this paper, and it is proved under the assumption EXP$\neq$NEXP. The proof consists of simulating the workings

of a non-deterministic Turing machine via an MSO question on a path. Though the idea of simulating Turing machines has appeared before in similar contexts [14], because the graphs we have here are very restricted we face a number of significant new challenges. The main tool we use to overcome them, which may be of independent interest, is an MSO formula construction that compares the sizes of ordered sets while using an extremely small number of quantifiers. In the end, this result strengthens both non-elementary MSO lower bounds given in [8] (for trees and for binary strings), modulo a slightly stronger complexity assumption. It also resolves the complexity of MSO model checking for max-leaf number, which was left open in [15]. As an added corollary, we give an alternative, self-contained proof of a result from [3], stating that MSO_2 model checking is not in XP for cliques unless EXP=NEXP.

Finally, we study one of the recent positive results in this area by considering the problem of model-checking MSO logic on rooted colored trees of height d. This is an especially interesting problem, since the $(d+1)$-fold exponential algorithm of [9] is the main tool used in the meta-theorems of both [9] and [11]. We show that, assuming the ETH, any algorithm needs at least $d+1$ levels of exponentiation, and therefore the algorithm of [9] is essentially optimal.

2 Preliminaries

The basic problem we are concerned with is model-checking: We are given a formula ϕ (in some logic) and a structure S (usually a graph or a string) and must decide if $S \models \phi$, that is, if S satisfies the property described by ϕ.

Due to space constraints, we do not give a full definition of FO and MSO logic here (see e.g. [8]). Let us just briefly describe some conventions. We use lower-case letters to denote singleton (FO) variables, and capitals to denote set variables. When the input is a graph, we assume the existence of an $E(x,y)$ predicate encoding edges; when it's a string a $\prec$ predicate encodes a total ordering; when it's a rooted tree a $C(x,y)$ predicate encodes that x is a child of y. Sometimes the input also has a set of colors (also called labels). For each color c we are given a unary predicate $P_c(x)$. When the input is an uncolored graph that consists of a single path it is possible to simulate the $\prec$ predicate by picking one endpoint of the path arbitrarily (call it s) and saying that $x \prec y$ if all paths from s to y contain x. Thus, model-checking MSO logic on uncolored paths is at least as hard as it is on unary strings. In most of the paper when we talk about MSO logic for graphs we mean MSO_1, that is, with quantification over vertex sets only. An exception is Corollary 4 which talks about MSO_2 logic, which allows edge set quantifiers. We will implicitly assume that our MSO formulas are allowed to use simple set operations (such as union and intersection) as these can be implemented with FO formulas of constant size.

A graph is a threshold graph ([1]) if it can be constructed from K_1 by repeatedly adding *union* vertices (not connected to any previous vertex) and *join* vertices (connected to all previous vertices), one at a time. Thus, a threshold graph can be described by a string over the alphabet $\{u, j\}$. A graph is a cograph if it is K_1, or it is a disjoint union of cographs, or it is the complement

of a cograph. It is not hard to see that threshold graphs are cographs. From the definition it follows that any class of graphs that contains K_1 and is closed under disjoint union and complement contains all cographs; if it is closed under the union and join operations it contains all threshold graphs.

All logarithms are base two. We define $\exp^{(k)}(n)$ as follows: $\exp^{(0)}(n) = n$ and $\exp^{(k+1)}(n) = 2^{exp^{(k)}(n)}$. Then $\log^{(k)} n$ is the inverse of $\exp^{(k)}(n)$. Finally, $\log^* n$ is the minimum i such that $\log^{(i)} n \leq 1$.

3 Threshold Graphs

As mentioned, Frick and Grohe [8] showed that there is no FPT model-checking algorithm for FO logic on trees with an elementary dependence on the formula size, under standard complexity assumptions. In many senses this is a great lower bound result, because it matches the tower of exponentials that appears in the running time of Courcelle's theorem, while looking both at a much simpler logic (FO rather than MSO_2) and at the class of graphs with the smallest possible treewidth, namely trees.

Courcelle, Makowsky and Rotics [3] have given an extension of Courcelle's theorem to MSO_1 logic for clique-width. The parameter dependence is again a tower of exponentials and, since trees have cliquewidth at most 3 ([4]), we already know that this cannot be avoided even for graphs of constant clique-width. Here we will slightly strengthen this result, showing that the non-elementary dependence cannot be avoided even on cographs, the class of graphs that has the smallest possible clique-width (that is, clique-width 2) without being trivial. We will heavily rely on a lower bound, due again to Frick and Grohe, on the complexity of model checking on binary strings.

One interesting consequence of the lower bound we give for cographs is that it precludes the existence of an FPT algorithm with elementary parameter dependence for any graph class that satisfies two simple properties: closure under disjoint unions and closure under complement. The reason for this is that if a class is closed under both of these operations and it contains the single-vertex graph, then it must contain all cographs (we will also show that the assumption that K_1 is in the class is not needed). This observation helps to explain why, though some of the recent elementary model-checking algorithms which have appeared work on union-closed graph classes, and some work on complement-closed graph classes, no such algorithms are known for classes with both properties.

The proof we present here is relatively simple and it relies on the following theorem.

Theorem 1 ([8]). *Unless FPT=AW[*], for any constant c and any elementary function f there is no model-checking algorithm for FO logic on binary words which given a formula ϕ and a word w decides if $w \models \phi$ in time at most $f(\phi)|w|^c$.*

We will reduce this problem to FO model checking on threshold graphs. This is quite natural, since the definition of threshold graphs gives a straightforward correspondence between graphs and strings.

Theorem 2. *Unless FPT=AW[∗], for any constant c and any elementary function f there is no model-checking algorithm for FO logic on connected threshold graphs which given a formula ϕ and such a graph G decides if $G \models \phi$ in time at most $f(\phi)|G|^c$.*

Proof (sketch). We can encode an arbitrary binary string with a threshold graph by encoding each 0 by two vertices uj and each 1 by three vertices ujj. We also need to encode a FO formula for strings to one for graphs. Existential quantification in the string is simulated by existential quantification in the graph where we ask that the vertex selected is a union vertex (this can be expressed in FO logic). The predicates $x \prec y$ and $P_1(x)$ can be implemented by checking if there are join vertices connected to x but not y in the first case, and checking for two join vertices connected to x but not later union vertices in the second. □

Corollary 1. *Let $\mathcal{C}$ be a graph class that is closed under disjoint union and complement, or under disjoint union and join. Unless FPT=AW[∗], for any constant c and any elementary function f there is no model-checking algorithm for FO logic on $\mathcal{C}$ which given a formula ϕ and a graph $G \in \mathcal{C}$ decides if $G \models \phi$ in time at most $f(\phi)|G|^c$.*

4 Paths, Unary Strings

The main result of this section is a reduction proving that, under the assumption that EXP≠NEXP, there is no FPT model-checking algorithm for MSO logic with an elementary parameter dependence on the formula even on graphs that consist of a single path, or equivalently, on unary strings. As a consequence, this settles the complexity of MSO model-checking on graphs with bounded max-leaf number, a problem left open in [15], since paths have the smallest possible max-leaf number. Until now a similar result was known only for the much richer class of binary strings (or equivalently colored paths), under the weaker assumption that P≠NP [8]. It is somewhat surprising that we are able to extend this result to uncolored paths, because in this case the size of the input is exponentially blown-up compared to a reasonable encoding. One would expect this to make the problem easier, but in fact, it only makes it more complicated to establish hardness.

Indeed, one of the main hurdles in proving a lower bound for MSO on unary strings, or paths, is information-theoretic. Normally, one would start with an NP-hard problem, and reduce to a model-checking instance with a very small formula ϕ. But, because the path we construct can naturally be stored with a number of bits that is logarithmic in its size, in order to encode n bits of information from the original instance into the new instance we need to construct a path of exponential size. Thus, a polynomial-time reduction seems unlikely and this is the reason we end up using the assumption that EXP≠NEXP, instead of P≠NP.

Our approach is to start from the prototypical NEXP-complete problem: given n bits of input for a non-deterministic Turing machine that runs in time 2^{n^k}, does the machine accept? We will use the input path to simulate the machine's tape

and then ask for a subset of the vertices of this path that corresponds to cells in the tape where 1 is written. Thus, what we need at this point is an MSO formula that checks if the chosen vertices encode a correct accepting computation.

Of course, to describe a machine's computation in MSO logic a significant amount of machinery will be needed. We note that, though the approach of simulating a Turing machine with an MSO formula has been used before (e.g. [14]), the problem here is significantly more challenging for two reasons: first, unlike previous cases the input here is uncolored, so it is harder to encode arbitrary bits; and second, there are (obviously) no grid-like minors in our graph, so it's harder to encode the evolution of a machine's tape, and in particular to identify vertices that correspond to the same tape cell in different points in time.

Our main building block to overcome these problems is an MSO construction which compares the sizes of paths (or generally, ordered sets) of size n with very few (roughly $2^{O(\log^* n)}$) quantifiers. This construction may be of independent interest in the context of the counting power of MSO logic. We first describe how to build this formula, then use it to obtain other basic arithmetic operations (such as exponentiation and division) and finally explain how they all fit together to give the promised result.

4.1 Measuring Long Paths with Few Quantifiers

To keep the presentation simple we will concentrate on the model-checking problem on unary strings; formulas for MSO on paths can easily be constructed as explained in section 2. We therefore assume that there is a predicate $\prec$ which gives a total ordering of all elements.

Let us now develop our basic tool, which will be an MSO formula $eq_L(P_1, P_2)$, where P_1, P_2 are free set variables. The desired behavior of the formula is that if $|P_1| = |P_2|$ and $|P_1| \leq L$ then the formula will be true, while on the other hand whenever the formula is true it must be the case that $|P_1| = |P_2|$. In other words, the formula will always correctly identify equal sets with size up to L, and it will never identify two unequal sets as equal (it may however be false for two equal sets larger than L). Our main objective is to achieve this with as few quantifiers as possible.

We will work inductively. It should be clear that for very small values of L (say $L = 4$) it is possible to compare sets of elements with size at most L with a constant number of set and vertex quantifiers and we can simply make the formula false if one set has more than 4 elements. So, suppose that we have a way to construct the desired formula for some L. We will show how to use it to make the formula $eq_{L'}$, where $L' \geq L \cdot 2^L$. If our recursive definition of $eq_{L'}$ uses a constant number of copies of eq_L then in the end we will have $|eq_L| = 2^{O(\log^* L)}$, because for each level of exponentiation we blow up the size of a formula by a constant factor. This will be sufficiently small to rule out a non-elementary parameter dependence.

Let us now give a high-level description of the idea, by concentrating first on the set P_1. We will select a subset of P_1, call it Q_1, and this naturally divides P_1 into sections, which are defined as maximal sets of vertices of P_1, consecutive in

the ordering, with the property that either all or none of their vertices belong in Q_1. We will make sure that all sections have length L, except perhaps the last, which we call the remainder (see Figure 1). It is not hard to see that this structure can be imposed if the predicate eq_L is available. We do the same for P_2 and now we need to verify that the two remainders have the same length (easy with eq_L) and that we have the same number of sections on P_1 and P_2.

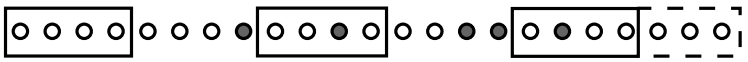


Fig. 1. An example of the counting structure imposed on a set for $L = 4$. Assume that the elements of the set are displayed here in the ordering given by $\prec$. We select a set of elements (indicated by boxes) to create sections of elements of size L, and a remainder set (indicated by a dashed box). We then select some elements from each section appropriately (indicated by solid grey filling) so that we simulate counting in binary. For $L = 4$ this method can count up to length $L2^L = 64$ before overflowing.

We need to count the number of sections this structure creates on P_1 (which may be up to 2^L). We select another subset of P_1, call it B_1. The intuition here is that selecting B_1 corresponds to writing a binary number on each section, by interpreting positions selected in B_1 as 1 and the rest as 0. We will now need to make sure that each section encodes the binary number that is one larger than the number encoded by the immediately preceding section. This is achievable by using eq_L to locate the elements that represent the same bit positions. We also make sure that there was no overflow in the counting and that counting started from zero, that is, all sections have some vertex not in B_1 and the first has no vertices in B_1.

Finally, assuming that the above counting structure is correctly imposed on both P_1 and P_2 all that is left is to take the last sections on both and compare them. If the same binary number is encoded on both then $|P_1| = |P_2|$.

Lemma 1. *Let $L > 2$ be a power of two. Then we can define a formula $eq_L(P_1, P_2)$ such that the formula is true if and only if $|P_1| = |P_2| < L \log L$. Furthermore $|eq_L| = 2^{O(\log^* L)}$.*

Before we go on, we will also need formulas to perform some slightly more complicated arithmetic operations than simply counting. In particular, we will need a formula $exp_L(P_1, P_2)$, which will be true if $|P_2| = 2^{|P_1|}$. The trick we use for this is shown in Figure 2.

Finally, we will need the following MSO formulas $root^{(k)}(P_1, P_2)$ which checks if $|P_2| = |P_1|^k$, assuming k is a power of two; $div(P_1, P_2)$ which checks is $|P_1|$ divides $|P_2|$; and $mod_L(P_1, P_2, R)$ which is true if $|P_2| \bmod |P_1| = |R|$ and $|P_1| \leq L$.

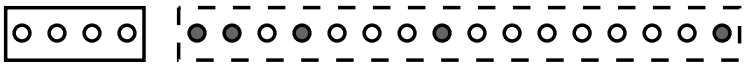


Fig. 2. An example where the set on the left has size 4 and we verify that the set on the right has size 2^4. First we select a subset of elements on the right of size one more than the set on the left. Then we ensure that distances between consecutive elements are doubled at each step and the first and last element are selected.

4.2 Hardness for Unary Strings and Paths

Theorem 3. *Let f be an elementary function and c a constant. If there exists an algorithm which, given a unary string w of length n and an MSO formula ϕ decides if $w \models \phi$ in time $f(|\phi|)n^c$ then EXP=NEXP.*

Proof (sketch). Suppose we are given a non-deterministic Turing machine running in time $T = 2^{n^k}$ for k a power of two, and therefore using at most T cells of its tape, when given n bits of input. We are also given the n bits of input, which we interpret as a binary number $I \leq 2^n$. We must decide if the machine accepts using the hypothetical model-checking algorithm for unary words.

We construct a path of size $T^2(2I + 1)$. Using *div* we locate a sub-path of size I (finding the largest odd divisor of the path) and a sub-path of size T^2, which we divide into sections of size T using *root*. These sections will represent snapshots of the tape during the machine's execution. We ask for a subset of the elements of the path that encodes the tape cells on which the machine writes 1. Now we need to check two things: first, that the bits at the beginning of the tape correspond to the input. This can be done with *exp* and some arithmetic on the size of I. Second, that the bits selected encode a correct computation. This is done by checking all pairs of elements from consecutive snapshots that correspond to the same cell. These are identified using eq_L (their distance is exactly T). □

Corollary 2. *Let f be an elementary function and c a constant. If there exists an algorithm which, given a path P on n vertices and an MSO formula ϕ decides if $P \models \phi$ in time $f(|\phi|)n^c$ then EXP=NEXP.*

Corollary 3. *Let f be an elementary function, c a constant, and $\mathcal{C}$ a class of graphs closed under edge sub-divisions. If there exists an algorithm which, given a graph $G \in \mathcal{C}$ on n vertices and an MSO formula ϕ decides if $G \models \phi$ in time $f(|\phi|)n^c$ then EXP=NEXP. The same is true if $\mathcal{C}$ is closed under induced subgraphs and, for all $d > 0$ contains a graph with diameter d.*

Finally, we can extend the ideas given above to obtain an alternative, self-contained proof of a result given in [3]: MSO_2 model-checking on cliques is not in XP, unless EXP=NEXP. In [3] this is proved under the equivalent assumption $\text{P}_1 \neq \text{NP}_1$ (the P≠NP assumption for unary languages). That proof relies on the work of Fagin on graph spectra [6].

Here we can simply reuse the ideas of Theorem 3 by observing two basic facts: first, with an appropriate MSO_2 formula we can select a set of edges in the given clique that induces a spanning path. Therefore, we can assume we have the same structure as in the case of paths. Second, the eq_L predicate can be constructed in constant size, since two disjoint sets of vertices are equal if and only if there exists a perfect matching between them in the clique (and this is MSO_2-expressible).

Corollary 4. *If there exists an algorithm which, given a clique K_n on n vertices and an MSO_2 formula ϕ decides if $K_n \models \phi$ in $n^{f(|\phi|)}$, for any function f, then EXP=NEXP.*

5 Tree-Depth

In this section we give a lower bound result that applies to the model-checking algorithm for trees of bounded height given by Gajarský and Hliněný [9]. We recall here the main result:

Theorem 4 ([9]). *Let T be a rooted t-colored tree of height $h \geq 1$, and let ϕ be an MSO sentence with r quantifiers. Then $T \models \phi$ can be decided by an FPT algorithm in time $O\left(\exp^{(h+1)}\left(2^{h+5}r(t+r)\right) + |V(T)|\right)$.*

Theorem 4 is the main algorithmic tool used to obtain the recent elementary model-checking algorithms for tree-depth and shrub-depth given in [9] and [11], since in both cases the strategy is to interpret the graph into a colored tree of bounded height.

The running time given in Theorem 4 is an elementary function of the formula ϕ, but non-elementary in the height of the tree. Though we would very much like to avoid that, it is not hard to see that the dependence on at least one of the parameters must be non-elementary, since allowing h to grow eventually gives the class of all trees so the lower bound result of Frick and Grohe should apply.

It is less obvious however what the height of the exponentiation tower has to be exactly, as a function of h, the height of the tree. The fact that we know that the height of the tower must be unbounded (so that we eventually get a non-elementary function) does not preclude an algorithm that runs in time $\exp^{(\sqrt{h})}(|\phi|)$ or, less ambitiously, $\exp^{(h/2)}(|\phi|)$, or even $\exp^{(h-5)}(|\phi|)$. Recall that we are trying to determine the number of levels of exponentiation in the running time here, so shaving off even an additive constant would be a non-negligible improvement.

We show that even such an improvement is probably impossible, and Theorem 4 determines precisely the complexity of MSO model-checking on colored trees of height h, at least in the sense that it gives exactly the correct level of exponentiations. We establish this fact assuming the ETH, by combining lower bound ideas which have appeared in [8] and [15]. More specifically, the main technical obstacle is comparing indices, or in other words, counting economically in our construction. For this, we use the tree representation of numbers of [8] pruned

to height $h-1$. We then use roughly $\log^{(h)} n$ colors to differentiate the leaves of the constructed trees.

Theorem 5. *If for some constant $h \geq 1$ there exists a model-checking algorithm for t-colored rooted trees of height h that runs in $\exp^{(h+1)}(o(t)) \cdot poly(n)$ time for trees with n vertices then the Exponential Time Hypothesis fails.*

6 Conclusions and Open Problems

We have proved non-elementary lower bounds for FO logic on cographs and MSO logic on uncolored paths. The hope is that, since these lower bounds concern very simple graph families, they can be used as "sanity checks" guiding the design of future graph widths. We have also given a lower bound for MSO logic on colored trees of bounded height. It would be interesting to see if this can be extended to uncolored trees.

Finally, let us mention that a promising direction in this area that we did not tackle here is that of alternative logics, besides FO and MSO variants. One example is the meta-theorems given by Pilipczuk [16] for a kind of modal logic. The algorithmic properties of such logics are still mostly unexplored but they may be a good way to evade the lower bounds given in [8] and this paper.

References

1. Chvátal, V., Hammer, P.L.: Aggregation of inequalities in integer programming. Annals of Discrete Mathematics 1, 145–162 (1977)
2. Courcelle, B.: The monadic second-order logic of graphs. i. Recognizable sets of finite graphs. Inf. Comput. 85(1), 12–75 (1990)
3. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory Comput. Syst. 33(2), 125–150 (2000)
4. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Mathematics 101(1-3), 77–114 (2000)
5. Dawar, A., Grohe, M., Kreutzer, S., Schweikardt, N.: Approximation schemes for first-order definable optimisation problems. In: LICS, pp. 411–420. IEEE Computer Society (2006)
6. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. Proceedings American Mathematical Society (1974)
7. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. J. ACM 48(6), 1184–1206 (2001)
8. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. Ann. Pure Appl. Logic 130(1-3), 3–31 (2004)
9. Gajarský, J., Hliněný, P.: Faster deciding mso properties of trees of fixed height, and some consequences. In: D'Souza, D., Kavitha, T., Radhakrishnan, J. (eds.) FSTTCS. LIPIcs, vol. 18, pp. 112–123. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
10. Ganian, R.: Twin-cover: Beyond vertex cover in parameterized algorithmics. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 259–271. Springer, Heidelberg (2012)

11. Ganian, R., Hliněný, P., Nešetřil, J., Obdržálek, J., de Mendez, P.O., Ramadurai, R.: When trees grow low: Shrubs and fast MSO_1. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 419–430. Springer, Heidelberg (2012)
12. Grohe, M.: Logic, graphs, and algorithms. Electronic Colloquium on Computational Complexity (ECCC) 14(091) (2007)
13. Hliněný, P., Il Oum, S., Seese, D., Gottlob, G.: Width parameters beyond treewidth and their applications. Comput. J. 51(3), 326–362 (2008)
14. Kreutzer, S.: On the parameterized intractability of monadic second-order logic. Logical Methods in Computer Science 8(1) (2012)
15. Lampis, M.: Algorithmic meta-theorems for restrictions of treewidth. Algorithmica 64(1), 19–37 (2012)
16. Pilipczuk, M.: Problems parameterized by treewidth tractable in single exponential time: A logical approach. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 520–531. Springer, Heidelberg (2011)

The Complexity of Proving That a Graph Is Ramsey

Massimo Lauria[1], Pavel Pudlák[2], Vojtěch Rödl[3], and Neil Thapen[2]

[1] Royal Institute of Technology, Stockholm
lauria@kth.se
[2] Academy of Sciences of the Czech Republic
{pudlak,thapen}@math.cas.cz
[3] Emory University, Atlanta
rodl@mathcs.emory.edu

Abstract. We say that a graph with n vertices is c-Ramsey if it does not contain either a clique or an independent set of size $c \log n$. We define a CNF formula which expresses this property for a graph G. We show a superpolynomial lower bound on the length of resolution proofs that G is c-Ramsey, for *every* graph G. Our proof makes use of the fact that every Ramsey graph must contain a large subgraph with some of the statistical properties of the random graph.

Introduction

The proof of the existence of c-Ramsey graphs, i.e., graphs that have no clique or independent set of size $c \log n$, was one of the first applications of the probabilistic method in combinatorics [12]. The problem, posed by Erdős, of constructing such graphs explicitly is still open [9]. In this paper we are interested in the problem: how hard is it to *certify* that a graph G of size n is c-Ramsey. By definition, c-Ramsey graphs are in **coNP**; finding polynomial certificates would put this set in **NP**. We believe that this is not possible and our result is a first step in this direction. We show that there is no resolution proof that G is c-Ramsey with length shorter than $n^{\Omega(\log n)}$. Let us stress that this is not a worst-case result: for a fixed constant $c \geq 2$, the lower bound $n^{\Omega(\log n)}$ holds for *every* c-Ramsey graph G. The brute force approach to checking that G satisfies the property takes time $n^{O(\log n)}$ and can be turned into a resolution proof. Hence our result is asymptotically optimal.

We prove our result for one natural formalization of the c-Ramsey property, a formalization that we call *binary*. For an alternative formalization, the *unary* formalization, we derive the same lower bound only for tree-like resolution proofs (see Section 1.1).

Since most SAT solvers used in practice are essentially proof search algorithms for resolution [19], our lower bound on resolution proof size shows that the verification problem is hard for quite a large class of algorithms. Also note that, while it does not follow from the resolution lower bound that there is no algorithm which will construct a Ramsey graph in polynomial time, it does follow

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 684–695, 2013.

that, given an algorithm, there is no polynomial-size resolution proof that the algorithm decides if a graph is c-Ramsey.

The finite Ramsey theorem states that for any k, there is some N such that every graph of size at least N contains a clique or independent set of size k. We write $r(k)$ for the least such N. Computing the actual value of $r(k)$ is challenging, and so far only a few values have been discovered. In fact, even to determine the asymptotic behavior of the function $r(k)$ is a longstanding open problem, in spite of the nontrivial results proved about it, see e.g., [12,23,10].

A c-Ramsey graph is a witness that $r(c \log n) > n$, so proving that a graph is Ramsey is in some sense proving a lower bound for $r(k)$. Previously, proof complexity has focused on upper bounds for $r(k)$. Krishnamurthy and Moll [18] proved partial results on the complexity of proving the exact upper bound, and conjectured that the propositional formalization of the exact upper bound is hard in general. Krajíček later proved an exponential lower bound on the length of bounded depth Frege proofs of the tautology proposed by Krishnamurthy and Moll [17]. The upper bound $r(k) \leq 4^k$ has short proofs in a relatively weak fragment of sequent calculus, in which every formula in a proof has small constant depth [17,21]. Recently Pudlák [22] has shown a lower bound on proofs of $r(k) \leq 4^k$ in resolution. We discuss this in more detail in Section 1. There are also results known about the off-diagonal Ramsey numbers $r(k, s)$ where cliques of size k and independent sets of size s are considered. See [13,1,14,7] for estimates and [8] for resolution lower bounds.

The paper is organized as follows. In Section 1 we formally state our main result, mention some open problems, and then outline the method we will use. In Section 2 we apply this to prove a simple version of our main theorem, restricted to the case when G is a random graph. In Section 3 we prove the full version. This will use one extra ingredient, a result from [20] that every Ramsey graph G has a large subset with some of the statistical density properties of the random graph.

1 Definitions and Results

Resolution [6] is a system for refuting propositional CNFs, that is, propositional formulas in conjunctive normal form. A resolution refutation is a sequence of disjunctions, which in this context we call *clauses*. Resolution has a single inference rule: from two clauses $A \vee x$ and $B \vee \neg x$ we can infer the new clause $A \vee B$ (which is a logical consequence). A *resolution refutation* of a CNF ϕ is a derivation of the empty clause from the clauses of ϕ. For an unsatisfiable formula ϕ we define $L(\phi)$ to be the length, that is, the number of clauses, of the shortest resolution refutation of ϕ. If ϕ is satisfiable we consider $L(\phi)$ to be infinite.

Let $c > 0$ be a constant, whose value will be fixed for the rest of the paper.

Definition 1 (c-Ramsey graph). *We say that a graph with n vertices is* c-Ramsey *if there is no set of $c \log n$ vertices which form either a clique or an independent set.*

We now describe how we formalize the c-Ramsey property in a way suitable for the resolution proof system. Given a graph G on $n = 2^k$ vertices, we will define

a formula Ψ_G in conjunctive normal form which is satisfiable if and only there is a homogeneous set (a set inducing an empty or complete subgraph) of size ck in G, that is, if and only if G is not Ramsey. We identify the vertices of G with the binary strings of length k. In this way we can use an assignment to k propositional variables to determine a vertex.

The formula Ψ_G has variables to represent an injective mapping from a set of ck "indices" to the vertices of G, and asserts that the vertices mapped to form either a clique or an independent set. It has a single extra variable y to indicate which of these two cases holds.

In more detail, for each $i \in [ck]$ we have k variables $x_1^i, \ldots, x_k^i$ which we think of as naming, in binary, the vertex of G to which i is mapped. We have an additional variable y, so there are ck^2+1 variables in total. To simplify notation we will write propositional literals in the form "$x_b^i = 1$", "$x_b^i \neq 0$", "$x_b^i = 0$" and "$x_b^i \neq 1$". The first and the second are aliases for the literal x_b^i. The third and the fourth are aliases for literal $\neg x_b^i$.

Thus the formula Ψ_G expressing that G is c-Ramsey is a conjunction of clauses asserting the following:

1. **The map is injective.** For each vertex $v \in V(G)$, represented as $v_1 \cdots v_k$ in binary, and each pair of distinct $i, j \in [ck]$, we have the clause

$$\bigvee_{b=1}^{k} (x_b^i \neq v_b) \vee \bigvee_{b=1}^{k} (x_b^j \neq v_b).$$

   These clauses guarantee that no two indices i and j map to the same vertex v.

2. **If $y = 0$, then the image of the mapping is an independent set.** For each pair of distinct vertices $u, v \in V(G)$, represented respectively as $u_1 \ldots u_k$ and $v_1 \ldots v_k$, and each pair of distinct $i, j \in [ck]$, if $\{u, v\} \in E(G)$ we have the clause

$$y \vee \bigvee_{b=1}^{k} (x_b^i \neq u_b) \vee \bigvee_{b=1}^{k} (x_b^j \neq v_b).$$

   These clauses guarantee that, if $y = 0$, then no two indices are mapped to two vertices with an edge between them.

3. **If $y = 1$, then the image of the mapping is a clique.** For each pair of distinct vertices $u, v \in V(G)$, represented respectively as $u_1 \ldots u_k$ and $v_1 \ldots v_k$, and each pair of distinct $i, j \in [ck]$, if $\{u, v\} \notin E(G)$ we have the clause

$$\neg y \vee \bigvee_{b=1}^{k} (x_b^i \neq u_b) \vee \bigvee_{b=1}^{k} (x_b^j \neq v_b).$$

   These clauses guarantee that, if $y = 1$, then no two indices are mapped to two vertices without an edge between them.

The formula Ψ_G has $\binom{ck}{2}\left(n+\binom{n}{2}\right)$ clauses in total, and so is unusual in that the number of clauses is exponentially larger than the number of variables. However the number of clauses is polynomial in the number n of vertices of G.

If G is Ramsey, then Ψ_G is unsatisfiable and has only $c\log^2 n+1$ variables. So we can refute Ψ_G in quasipolynomial size by a brute-force search through all assignments:

Proposition 1. *If G is c-Ramsey, the formula Ψ_G has a (treelike) resolution refutation of size $n^{O(\log n)}$.*

At this point, we should recall the formalization of the Ramsey theorem that is more usually studied in proof complexity. This is the family RAM_n of propositional CNFs, where RAM_n has one variable for each distinct pair of points in $[n]$ and asserts that the graph represented by these variables is $\frac{1}{2}$-Ramsey. Hence RAM_n is satisfiable if and only if any $\frac{1}{2}$-Ramsey graph on n vertices exists. In contrast, our formula Ψ_G is satisfiable if and only if our particular graph G is not c-Ramsey.

Put differently, a refutation of RAM_n is a proof that $r(k)\leq 2^{2k}$. This was recently shown to require exponential size (in n) resolution refutations [22]. On the other hand a refutation of Ψ_G is a proof that G is c-Ramsey, and hence that G witnesses that $r(k)>2^{\frac{k}{c}}$.

We now state our main result. We postpone the proof to Section 3.

Theorem 1. *Let G be any graph with n vertices. Then $L(\Psi_G)\geq n^{\Omega(\log n)}$.*

If G is not c-Ramsey then this is trivial, since Ψ_G is satisfiable and therefore $L(\Psi_G)$ is infinite by convention. If G is c-Ramsey, then by Proposition 1 this bound is tight and we know that $L(\Psi_G)=n^{\Theta(\log n)}$.

1.1 Open Problems

Our formalization Ψ_G, apart from being natural, is motivated by the τ-tautologies introduced by Krajíček [15]. Although it has similar properties we are not able to show that it is a τ-tautology.

We call Ψ_G the *binary encoding* because the vertices of the graph are represented by strings of propositional variables. One can also consider the *unary encoding* Ψ_G^u in which a vertex of the graph is determined by a single propositional variable. More precisely, the mapping from an index i to the vertices of G is represented by n variables $\{p_v^i : v\in V(G)\}$ and we have clauses asserting that for each i, exactly one of the variables p_v^i is true. Otherwise the structure of Ψ_G^u is similar to that of Ψ_G. As before, if G is a c-Ramsey graph we have the brute-force upper bound $L(\Psi_G^u)=n^{O(\log n)}$. But we are not able to prove a superpolynomial lower bound on resolution size. However we are able to prove such a lower bound if we restrict to treelike resolution, as a corollary of our main theorem (we are grateful to Leszek Kołodziejczyk for pointing out this simple proof).

Theorem 2. *Let G be any c-Ramsey graph with n vertices. Then Ψ_G^u requires treelike resolution refutations of size $n^{\Omega(\log n)}$.*

Proof. (Sketch) Suppose we have a small treelike resolution refutation of the unary formula Ψ_G^u. We can produce from it an at most polynomially larger treelike Res(k) refutation of the binary formula Ψ_G as follows. Replace each variable p_v^i asserting that index i is mapped to vertex v with the conjunction $\bigwedge_{b=1}^{k} x_b^i = v_b$. The substitution instance of Ψ_G^u is then almost identical to the Ψ_G, except for the additional clauses asserting that every index maps to exactly one vertex; but these are easy to derive in treelike Res(k).

It is well known that every treelike depth $d+1$ Frege proof can be made into a DAG-like depth d Frege proof with at most polynomial increase in size [16]. In particular, we can turn our treelike Res(k) refutation of Ψ_G into a resolution refutation. The lower bound then follows from Theorem 1. □

Lower bounds for DAG-like resolution would have interesting consequences for various areas of proof complexity [3,11]. The problem of proving a superpolynomial lower bound on Ψ_G^u is related to the following open problem (rephrased from [5]): consider a random graph G distributed according to $\mathcal{G}(n, n^{-(1+\epsilon)\frac{2}{k-1}})$ for some $\epsilon > 0$. Does every resolution proof that there is no k-clique in G require size $n^{\Omega(k)}$?

Another natural problem is to extend our lower bound to proof systems stronger than resolution. A superpolynomial lower bound on the proofs of Ψ_G in Res(log) (resolution with logarithmic size conjunctions in clauses) would imply a superpolynomial lower bound on general resolution proofs of Ψ_G^u.

1.2 Resolution Width and Combinatorial Games

The *width* of a clause is the number of literals it contains. The width of a CNF ϕ is the width of its widest clause. Similarly the width of a resolution refutation Π is the width of its widest clause. The width of refuting an unsatisfiable CNF ϕ is the minimum width of Π over all refutations Π of ϕ. We will denote it by $W(\phi)$.

A remarkable result about resolution is that it is possible to prove a lower bound on the proof length by proving a lower bound on the proof width.

Theorem 3 ([4]). *For any CNF ϕ with m variables and width k,*

$$L(\phi) \geq 2^{\Omega\left(\frac{(W(\phi)-k)^2}{m}\right)}.$$

Now consider a game played between two players, called the Prover and the Adversary. The Prover claims that a CNF ϕ is unsatisfiable and the Adversary claims to know a satisfying assignment. At each round of the game the Prover asks for the value of some variable and the Adversary has to answer. The Prover saves the answer in memory, where each variable value occupies one memory location. The Prover can also delete any saved value, in order to save memory. If the deleted variable is asked again, the Adversary is allowed to answer differently. The Prover wins when the partial assignment in memory falsifies a clause of ϕ. The Adversary wins if he has a strategy to play forever.

If ϕ is in fact unsatisfiable, then the Prover can always eventually win, by asking for the total assignment. If ϕ is satisfiable, then there is an obvious winning strategy for the Adversary (answering according to a fixed satisfying assignment). However, even if ϕ is unsatisfiable, it may be that the Prover cannot win the game unless he uses a large amount of memory. Indeed, it turns out that the smallest number of memory locations that the Prover needs to win the game for an unsatisfiable ϕ is related to the width of resolution refutations. (We only need one direction of this relationship – for a converse see [2].)

Lemma 1. *Given an unsatisfiable CNF ϕ, it holds that $W(\phi) + 1$ memory locations are sufficient for the Prover in order to win the game against any Adversary.*

1.3 The Clique Formula

For any graph G, the formula $\Psi_G\restriction_{y=1}$ is satisfiable if and only if G has a clique of size ck. We will call this restricted formula Clique(G). Dually, $\Psi_G \restriction_{y=0}$ is equivalent to Clique($\bar{G}$). Since fixing a variable in a resolution refutation results in a refutation for the corresponding restricted formula, we have

$$\max\left\{L(\text{Clique}(G)), L(\text{Clique}(\bar{G}))\right\} \leq L(\Psi_G).$$

Furthermore we can easily construct a refutation of Ψ_G from refutations of $\Psi_G\restriction_{y=1}$ and $\Psi_G\restriction_{y=0}$. In this way we get

$$L(\Psi_G) \leq L(\text{Clique}(\bar{G})) + L(\text{Clique}(G)) + 1.$$

We can now describe our high-level approach. To lower-bound $L(\Psi_G)$ it is enough to lower-bound $L(\text{Clique}(G))$, which we will do indirectly by exhibiting a good strategy for the Adversary in the game on Clique(G). This game works as follows: the Adversary claims to know ck strings in $\{0,1\}^k$ which name ck vertices in G which form a clique. The Prover starts with no knowledge of these strings but can query them, one bit at a time, and can also forget bits to save memory. The Prover wins if at any point there are two fully-specified strings for which the corresponding vertices are not connected by an edge in G.

We will give a strategy for the Adversary which will beat any Prover limited to ϵk^2 memory for a constant $\epsilon > 0$. It follows by Lemma 1 that Clique(G) is not refutable in width ϵk^2. The formula Clique(G) has ck^2 variables and has width $2k$. Hence applying Theorem 3 we get

$$L(\Psi_G) \geq L(\text{Clique}(G)) \geq 2^{\Omega\left(\frac{(\epsilon k^2 - 2k)^2}{ck^2}\right)} \geq 2^{\Omega(k^2)} \geq n^{\Omega(\log n)}.$$

1.4 Other Notation

We will consider simple graphs with $n = 2^k$ vertices. We identify the vertices with the binary strings of length k. For any vertex $v \in G$ we denote its binary representation by $v_1 \cdots v_k$.

A *pattern* is a partial assignment to k variables. Formally, it is a string $p = p_1 \cdots p_k \in \{*, 0, 1\}^k$, and we say that p *is consistent with* v if for all $i \in [k]$ either $p_i = v_i$ or $p_i = *$. The *size* $|p|$ of p is the number of bits set to 0 or 1. The *empty pattern* is a string of k stars.

For any vertex $v \in V(G)$ we let $N(v)$ be the set $\{u \big| \{v, u\} \in E(G)\}$ of neighbors of v. Notice that $v \notin N(v)$. For any $U \subseteq V(G)$ we let $N(U)$ be the set of vertices of G which neighbor every point in U, that is, $\bigcap_{v \in U} N(U)$. Notice that $U \cap N(U) = \emptyset$.

2 Lower Bounds for the Random Graph

We consider random graphs on n vertices given by the usual distribution $\mathcal{G}(n, \frac{1}{2})$ in the Erdős-Rényi model.

Theorem 4. *If G is a random graph, then with high probability $L(\Psi_G) = n^{\Omega(\log n)}$.*

We will use the method outlined in Section 1.3 above, so to prove the theorem it is enough to give a strategy for the Adversary in the game on Clique(G) which forces the Prover to use a large amount of memory. This is Lemma 3 below. We first prove a lemma which captures the property of the random graph which we need.

Lemma 2. *For a random graph G, with high probability, the following property P holds. Let $U \subseteq V(G)$ with $|U| \leq \frac{1}{3}k$ and let p be any pattern with $|p| \leq \frac{1}{3}k$. Then p is consistent with at least one vertex in $N(U)$.*

Proof. Fix such a set U and such a pattern p. The probability that an arbitrary vertex $v \notin U$ is in $N(U)$ is at least $2^{-\frac{1}{3}k} = n^{-\frac{1}{3}}$. The pattern p is consistent with at least $n^{\frac{2}{3}} - |U|$ vertices outside U. The probability that no vertex consistent with p is in $N(U)$ is hence at most

$$\left(1 - n^{-\frac{1}{3}}\right)^{n^{\frac{2}{3}} - |U|} \leq e^{-(1-o(1))n^{\frac{1}{3}}}.$$

We can bound the number of such sets U by $n^{\frac{1}{3}k} \leq n^{\log n}$ and the number of patterns p by $3^k \leq n^2$, so by the union bound property P fails to hold with probability at most $2^{-\Omega(n^{\frac{1}{3}})}$. □

Lemma 3. *Let G be any graph with property P. Then there is an Adversary strategy in the game on* Clique(G) *which wins against any Prover who uses at most $\frac{1}{9}k^2$ memory locations.*

Proof. For each index $i \in [ck]$, we will write p^i for the pattern representing the current information in the Prover's memory about the ith vertex. The Adversary's strategy is to answer queries arbitrarily (say with 0) as long as the index i being queried has $|p^i| < \frac{1}{3}k - 1$. If $|p^i| = \frac{1}{3}k - 1$, the Adversary privately *fixes* the ith vertex to be some particular vertex v^i of G consistent with p^i, and then

answers queries to i according to v^i until, through the Prover forgetting bits, $|p^i|$ falls below $\frac{1}{3}k$ again, at which point the Adversary considers the ith vertex no longer to be fixed.

If the Adversary is able to guarantee that the set of currently fixed vertices always forms a clique, then the Prover can never win. So suppose we are at a point in the game where the Adversary has to fix a vertex for index i, that is, where the Prover is querying a bit for i and $|p^i| = \frac{1}{3}k - 1$. Let $U \subseteq V(G)$ be the set of vertices that the Adversary currently has fixed. It is enough to show that there is some vertex consistent with p^i which is connected by an edge in G to every vertex in U. But by the limitation on the size of the Prover's memory, no more than $\frac{1}{3}k$ vertices can be fixed at any one time. Hence $|U| \leq \frac{1}{3}k$ and the existence of such a vertex follows from property P. □

3 Lower Bounds for Ramsey Graphs

We prove Theorem 1, that for any c-Ramsey graph G on n vertices, $L(\Psi_G) \geq n^{\Omega(\log n)}$. As in the previous section we will do this by showing, in Lemma 5 below, that the Adversary has a strategy for the game on Clique(G) which forces the Prover to use a lot of memory.

Definition 2. *Given sets $A, B \subseteq V(G)$ we define their mutual* density *by*

$$d(A,B) = \frac{e(A,B)}{|A||B|}$$

where we write $e(A,B)$ for the number of edges in G with one end in A and the other in B. For a single vertex v we will write $d(v,B)$ instead of $d(\{v\},B)$.

Our main tool in our analysis of Ramsey graphs is the statistical property shown in Corollary 1 below, which plays a role analogous to that played by Lemma 2 for random graphs. We use the following result proved in [20, Case II of Theorem 1]:

Lemma 4 ([20]). *There exists constants $\beta > 0$, $\delta > 0$ such that if G is a c-Ramsey graph, then there is a set $S \subseteq V(G)$ with $|S| \geq n^{\frac{3}{4}}$ such that, for all $A, B \subseteq S$, if $|A|, |B| \geq |S|^{1-\beta}$ then $\delta \leq d(A,B) \leq 1 - \delta$.*

Now fix a c-Ramsey graph G. Let S, β and δ be as in the above lemma, and let $m = |S|$. Notice that since our goal is to give an Adversary strategy for the formula Clique(G), we will only use the lower bound $\delta \leq d(A,B)$ from the lemma.

Corollary 1. *Let $X, Y_1, Y_2, \ldots, Y_r \subseteq S$ be such that $|X| \geq rm^{1-\beta}$ and $|Y_1|, \ldots, |Y_r| \geq m^{1-\beta}$. Then there exists $v \in X$ such that $d(v, Y_i) \geq \delta$ for each $i = 1, \ldots, r$.*

Proof. For $i = 1, \ldots, r$ let

$$X_i = \{u \in X \mid d(u, Y_i) < \delta\}.$$

By Lemma 4, each $|X_i| < m^{1-\beta}$. Hence $X \setminus \bigcup_i X_i$ is non-empty and we can take v to be any vertex in $X \setminus \bigcup_i X_i$. □

The next lemma implies our main result, Theorem 1.

Lemma 5. *There is a constant $\epsilon > 0$, independent of n and G, such that there exists a strategy for the Adversary in the game on* Clique(G) *which wins against any Prover who is limited to $\epsilon^2 k^2$ memory locations.*

Proof. Let $\epsilon > 0$ be a constant, whose precise value we will fix later. As in the proof of Lemma 3, the Adversary's replies when queried about the ith vertex will depend on the size of p^i, the pattern representing the current information known to the Prover about the ith vertex. If $|p^i| < \epsilon k - 1$ the Adversary can reply in a somewhat arbitrary way (see below), but if $|p^i| = \epsilon k - 1$ then the Adversary will fix a value v^i for the ith vertex, consistent with p^i, and will reply according to v^i until $|p^i|$ falls back below ϵk, at which point the vertex is no longer fixed. By the limitation on the Prover's memory, no more than ϵk vertices can be fixed simultaneously, which will allow the Adversary to ensure that the set of currently fixed vertices always forms a clique.

Let S, β and δ be as in Lemma 4 and let $m = |S|$. We will need to use Corollary 1 above to make sure that the Adversary can find a v^i with suitable density properties when fixing the ith vertex. But here there is a difficulty which does not arise with the random graph. Corollary 1 only works for subsets of the set S, and S may be distributed very non-uniformly over the vertices of G. In particular, through some sequence of querying and forgetting bits for i, the Prover may be able to force the Adversary into a position where the set of vertices consistent with a small p^i has only a very small intersection with S, so that it is impossible to apply Corollary 1.

Let α be a constant with $0 < \alpha < \beta$, whose precise value we will fix later. We write C_p for the set of vertices of G consistent with a pattern p. We write $P_{\epsilon k}$ for the set of patterns p with $|p| \leq \epsilon k$. To avoid the problem in the previous paragraph, we will construct a non-empty set $S^* \subseteq S$ with the property that, for every $p \in P_{\epsilon k}$, either

$$C_p \cap S^* = \emptyset \quad \text{or} \quad |C_p \cap S^*| > m^{1-\alpha}.$$

In the second case we will call the pattern p *active*. The Adversary can then focus on the set S^*, in the sense that he will pretend that his clique is in S^* and will ignore the vertices outside S^*.

We construct S^* in a brute-force way. We start with $S_0 = S$ and define a sequence of subsets $S_0, S_1, \ldots$ where each $S_{t+1} = S_t \setminus C_p$ for the lexicographically first $p \in P_{\epsilon k}$ for which $0 < |S_t \cap C_p| \leq m^{1-\alpha}$, if any such p exists. We stop as soon as there is no such p, and let S^* be the final subset in the sequence. To show that S^* is non-empty, notice that at each step at most $m^{1-\alpha}$ elements are

removed. Furthermore there are at most $|P_{\epsilon k}|$ steps, since a set of vertices C_p may be removed at most once. Recall that $n = 2^k$ and $m \geq n^{\frac{3}{4}}$. We have

$$|P_{\epsilon k}| = \sum_{i=0}^{\epsilon k} 2^i \binom{k}{i} \leq 2^{\epsilon k} \sum_{i=0}^{\epsilon k} \binom{k}{i} \leq n^{\epsilon} n^{H(\epsilon)},$$

where $H(x)$ is the binary entropy function $-x \log x - (1-x)\log(1-x)$, and we are using the estimate $\sum_{i=0}^{\epsilon k} \binom{k}{i} \leq 2^{kH(\epsilon)}$ which holds for $0 < \epsilon < 1$. Then

$$|S^*| \geq |S| - |P_{\epsilon k}| \cdot m^{1-\alpha} \geq n^{\frac{3}{4}} - \epsilon k \cdot n^{\epsilon + H(\epsilon)} n^{\frac{3}{4}(1-\alpha)},$$

so, for large n, S^* is non-empty as long as we choose α and ϵ satisfying

$$\tfrac{3}{4}\alpha > \epsilon + H(\epsilon). \qquad (\star)$$

Notice that if S^* is non-empty then in fact $|S^*| > m^{1-\alpha}$, since S^* must intersect at least the set C_p where p is the empty pattern.

We can now give the details of the Adversary's strategy. The Adversary maintains the following three conditions, which in particular guarantee that the Prover will never win.

1. For each index i, if $|p^i| < \epsilon k$ then p^i is active, that is, $C_{p^i} \cap S^* \neq \emptyset$.
2. For each index i, if $|p^i| \geq \epsilon k$ then the ith vertex is fixed to some $v^i \in C_{p^i} \cap S^*$; furthermore the set U of currently fixed vertices v^j forms a clique.
3. For every active $p \in P_{\epsilon k}$ and every $U' \subseteq U$, we have

$$|C_p \cap S^* \cap N(U')| \geq |C_p \cap S^*| \cdot \delta^{|U'|}.$$

(Recall that $0 < \delta < 1$ is the constant from Lemma 4.) These are true at the start of the game, because no vertices are fixed and each p^i is the empty pattern.

Suppose that, at a turn in the game, the Prover queries a bit for an index i for which he currently has information p^i. If $|p^i| < \epsilon k - 1$, then by condition 1 there is at least one vertex v in $C_{p^i} \cap S^*$. The Adversary chooses an arbitrary such v and replies according to the bit of v. If $|p^i| \geq \epsilon k$, then a vertex $v^i \in C_{p^i}$ is already fixed, and the Adversary replies according to the bit of v^i.

If $|p^i| = \epsilon k - 1$, then the Adversary must fix a vertex v^i for i in a way that satisfies conditions 2 and 3. To preserve condition 2, v^i must be connected to every vertex in the set U of currently fixed vertices. To preserve condition 3, it is enough to choose v^i such that

$$d(v^i, C_p \cap S^* \cap N(U')) \geq |C_p \cap S^* \cap N(U')| \cdot \delta$$

for every active p in $P_{\epsilon k}$ and every $U' \subseteq U$. To find such a v^i we will apply Corollary 1, with one set Y for each pair of a suitable p and U'. We put

$$\begin{aligned} X &= C_{p^i} \cap N(U) \cap S^* \\ Y_{(p,U')} &= C_p \cap N(U') \cap S^* \text{ for each active } p \in P_{\epsilon k} \text{ and each } U' \subseteq U \\ r &= |\{\text{pairs } (p, U')\}| \leq |P_{\epsilon k}| \cdot 2^{|U|}. \end{aligned}$$

We know $|U| \leq \epsilon k$. By condition 1 we know p^i is active, hence $|C_{p^i} \cap S^*| > m^{1-\alpha}$. So by condition 3 we have

$$|X| \geq m^{1-\alpha}\delta^{\epsilon k} = m^{1-\alpha+\frac{4}{3}\epsilon \log \delta}.$$

For similar reasons we have the same lower bound on the size of each $Y_{(p,U')}$. Furthermore

$$r \leq 2^{\epsilon k} \cdot \epsilon k \cdot n^{\epsilon + H(\epsilon)} = \epsilon k \cdot n^{2\epsilon + H(\epsilon)} = \epsilon k \cdot m^{\frac{8}{3}\epsilon + \frac{4}{3}H(\epsilon)}.$$

To apply Corollary 1 we need to satisfy $|X| \geq rm^{1-\beta}$ and $|Y_{(p,U')}| \geq m^{1-\beta}$. Both conditions are implied by the inequality

$$\beta - \alpha > \tfrac{8}{3}\epsilon + \tfrac{4}{3}H(\epsilon) - \tfrac{4}{3}\epsilon \log \delta. \qquad (\dagger)$$

We can now fix values for the constants α and ϵ to satisfy the inequalities $(\star)$ and $(\dagger)$. Since $H(\epsilon)$ goes to zero as ϵ goes to zero, we can make the right hand sides of $(\star)$ and $(\dagger)$ arbitrary small by setting ϵ to be a small constant. We then set α appropriately.

Finally, it is straightforward to check that if the Prover forgets a bit for an index i, then the three conditions are preserved. □

Acknowledgements. Part of this work was done while Lauria was at the Institute of Mathematics of the Academy of Sciences of the Czech Republic, supported by the Eduard Čech Center. Lauria, Pudlák and Thapen did part of this research at the Isaac Newton Institute for the Mathematical Sciences, where Pudlák and Thapen were visiting fellows in the programme *Semantics and Syntax*. Pudlák and Thapen were also supported by grant IAA100190902 of GA AV ČR, and by Center of Excellence CE-ITI under grant P202/12/G061 of GA ČR and RVO: 67985840.

We would also like to thank the anonymous referees for their comments and corrections.

References

1. Ajtai, M., Komlós, J., Szemerédi, E.: A note on Ramsey numbers. Journal of Combinatorial Theory, Series A 29(3), 354–360 (1980)
2. Atserias, A., Dalmau, V.: A combinatorial characterization of resolution width. J. Comput. Syst. Sci. 74(3), 323–334 (2008)
3. Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. J. Artif. Intell. Res. (JAIR) 40, 353–373 (2011)
4. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow - resolution made simple. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, pp. 517–526 (1999)
5. Beyersdorff, O., Galesi, N., Lauria, M., Razborov, A.A.: Parameterized bounded-depth frege is not optimal. ACM Trans. Comput. Theory 4(3), 7:1–7:16 (2012)

6. Blake, A.: Canonical Expressions in Boolean Algebra. PhD thesis, University of Chicago (1938)
7. Bohman, T., Keevash, P.: The early evolution of the h-free process. Inventiones Mathematicae 181(2), 291–336 (2010)
8. Carlucci, L., Galesi, N., Lauria, M.: Paris-harrington tautologies. In: Proc. of IEEE 26th Conference on Computational Complexity, pp. 93–103 (2011)
9. Chung, F.R.K., Erdős, P., Graham, R.L.: Erdős on Graphs: His Legacy of Unsolved Problems, 1st edn. AK Peters, Ltd. (January 1998)
10. Conlon, D.: A new upper bound for diagonal ramsey numbers. Annals of Mathematics 170(2), 941–960 (2009)
11. Dantchev, S., Martin, B., Szeider, S.: Parameterized proof complexity. Computational Complexity 20, 51–85 (2011), doi:10.1007/s00037-010-0001-1
12. Erdös, P.: Some remarks on the theory of graphs. Bull. Amer. Math. Soc. 53, 292–294 (1947)
13. Erdős, P., Szekeres, G.: A combinatorial problem in geometry. In: Gessel, I., Rota, G.-C. (eds.) Classic Papers in Combinatorics. Modern Birkhäuser Classics, pp. 49–56. Birkhäuser, Boston (1987)
14. Kim, J.H.: The Ramsey number $r(3,t)$ has order of magnitude $t^2/\log(t)$. Random Structures and Algorithms 7(3), 173–208 (1995)
15. Krajicek, J.: Tautologies from pseudo-random generators. Bulletin of Symbolic Logic, 197–212 (2001)
16. Krajíček, J.: Lower bounds to the size of constant-depth propositional proofs. Journal of Symbolic Logic 59(1), 73–86 (1994)
17. Krajíček, J.: A note on propositional proof complexity of some Ramsey-type statements. Archive for Mathematical Logic 50, 245–255 (2011), doi:10.1007/s00153-010-0212-9
18. Krishnamurthy, B., Moll, R.N.: Examples of hard tautologies in the propositional calculus. In: STOC 1981, 13th ACM Symposium on Th. of Computing, pp. 28–37 (1981)
19. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning sat solvers as resolution engines. Artificial Intelligence 175(2), 512–525 (2011)
20. Prömel, H., Rödl, V.: Non-ramsey graphs are c log n-universal. Journal of Combinatorial Theory, Series A 88(2), 379–384 (1999)
21. Pudlák, P.: Ramsey's theorem in Bounded Arithmetic. In: Schönfeld, W., Börger, E., Kleine Büning, H., Richter, M.M. (eds.) CSL 1990. LNCS, vol. 533, pp. 308–317. Springer, Heidelberg (1991)
22. Pudlák, P.: A lower bound on the size of resolution proofs of the Ramsey theorem. Inf. Process. Lett. 112(14-15), 610–611 (2012)
23. Spencer, J.: Asymptotic lower bounds for Ramsey functions. Discrete Mathematics 20, 69–76 (1977)

An Improved Lower Bound for the Randomized Decision Tree Complexity of Recursive Majority*,**

Nikos Leonardos

National and Kapodistrian University of Athens
Department of Informatics and Telecommunications
nikos.leonardos@gmail.com

Abstract. We prove that the randomized decision tree complexity of the recursive majority-of-three is $\Omega(2.55^d)$, where d is the depth of the recursion. The proof is by a bottom up induction, which is same in spirit as the one in the proof of Saks and Wigderson in their 1986 paper on the complexity of evaluating game trees. Previous work includes an $\Omega\big((7/3)^d\big)$ lower bound, published in 2003 by Jayram, Kumar, and Sivakumar. Their proof used a top down induction and tools from information theory. In 2011, Magniez, Nayak, Santha, and Xiao, improved the lower bound to $\Omega\big((5/2)^d\big)$ and the upper bound to $O(2.64946^d)$.

Keywords: Boolean functions, randomized computation, decision tree complexity, query complexity, lower bounds, generalized costs.

1 Introduction

In this paper we will be working with the decision tree model. We prove a lower bound on the randomized decision tree complexity of the recursive majority-of-three function. Formally, $\mathrm{maj}_1(x_1, x_2, x_3)$ is 1 if and only if at least two of x_1, x_2, x_3 are 1. Letting $y_i = (x_{(i-1)3^d+1}, \ldots, x_{i3^d})$, for $i = 1, 2, 3$, define for $d > 0$,

$$\mathrm{maj}_{d+1}(x_1, \ldots, x_{3^{d+1}}) = \mathrm{maj}(\mathrm{maj}_{d-1}(y_1), \mathrm{maj}_{d-1}(y_2), \mathrm{maj}_{d-1}(y_3)).$$

We write maj for maj_1. The function can be represented by a uniform ternary tree. In particular, let U_d be a tree of depth d, such that every internal node has three children and all leaves are on the same level. The function computed by interpreting U_d as a circuit with internal nodes labeled by maj-gates is maj_d.

This function seems to have been given by Ravi Boppana (see Example 1.2 in [7]) as an example of a function that has deterministic complexity 3^d, while its randomized complexity is asymptotically smaller. Other functions with this property are known. A notable example is the function nand_d, first analyzed by Snir [10]. This is the function represented by a uniform binary tree of depth d,

* This research was partly supported by ERC project CODAMODA.
** Available online at ECCC: http://eccc.hpi-web.de/report/2012/099/.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 696–708, 2013.

with the internal nodes labeled by nand-gates. A simple randomized framework that can be used to compute both maj_d and nand_d is the following. Start at the root; as long as the output is not known, choose a child at random and evaluate it recursively. Algorithms of this type are called in [7] *directional*. For maj_d the directional algorithm computes the output in $(8/3)^d$ queries. It was noted in [7] that better algorithms exist for maj_d. Interestingly, Saks and Wigderson show that the directional algorithm is optimal for the nand_d function, and show that its zero-error randomized decision tree complexity is $\Theta\big((\frac{1+\sqrt{33}}{4})^d\big)$. Their proof uses a bottom up induction and generalized costs. Their method of generalized costs allows them to charge for a query according to the value of the variable. Furthermore, they conjecture that the maximum gap between deterministic and randomized complexity is achieved for this function.

Inspired by their technique we prove an $\Omega(2.55^d)$ lower bound on maj_d that also holds for algorithms with bounded-error. (The bound of [7] for nand_d was extended to bounded-error algorithms by Santha in [8].) In contrast to the exact asymptotic bounds we have for nand_d, there had been no progress on the randomized decision tree complexity of maj_d for several years. However, recent papers have narrowed the gap between the upper and lower bounds for recursive majority. An $\Omega\big((7/3)^d\big)$ lower bound was showed in [4]. Jayram, Kumar, and Sivakumar, proved their bound using tools from information theory and a top down induction. Furthermore, they presented a non-directional algorithm that improves the $O\big((8/3)^d\big)$ upper bound. Magniez, Nayak, Santha, and Xiao [6], significantly improved the lower bound to $\Omega\big((5/2)^d\big)$ and the upper bound to $O(2.64946^d)$. (Both of these lower bounds hold for the case that the randomized decision tree is allowed to err.)

Our proof of the lower bound is simpler than the aforementioned ones; it doesn't require a background in information theory and it only uses induction. Note that, Landau, Nachmias, Peres, and Vanniasegaram [5], showed how to remove the information theoretic notions from the proof in [4], keeping its underlying structure the same. Our proof can be even more simplified, if one requires the known $\Omega(2.5^d)$ lower bound. A simpler proof of this bound seems to have been already known to Jonah Serman [9] in 2007.

We note that both maj_d and nand_d, belong to the class of *read-once functions*. These are functions that can be computed by read-once Boolean formulae, that is, formulae such that each input variable appears exactly once. Heiman, Newman, and Wigderson [2] showed that read-once formulae with threshold gates have zero-error randomized complexity $\Omega(n/2^d)$ (here n is the number of variables and d the depth of a canonical tree-representation of the read-once function). Heiman and Wigderson [3] managed to show that for every read-once function f we have $R(f) \in \Omega(D(f)^{0.51})$, where $R(f)$ and $D(f)$ are the randomized and deterministic complexity of f respectively. Note that the conjecture of Saks and Wigderson states that for every function f we have $R(f) \in \Omega(D(f)^{0.753\ldots})$.

2 Definitions and Notation

In this section we introduce basic concepts related to decision tree complexity. The reader can find a more complete exposition in the survey of Buhrman and de Wolf [1].

2.1 Definitions Pertaining to Decision Trees

A *deterministic Boolean decision tree* Q over a set of variables $Z = \{z_i \mid i \in [n]\}$, where $[n] = \{1, 2, \ldots, n\}$, is a rooted and ordered binary tree. Each internal node is labeled by a variable $z_i \in Z$ and each leaf with a value from $\{0, 1\}$. An *assignment* to Z (or an *input* to Q) is a member of $\{0, 1\}^n$. The output $Q(\sigma)$ of Q on an input σ is defined recursively as follows. Start at the root and let its label be z_i. If $\sigma_i = 0$, we continue with the left child of the root; if $\sigma_i = 1$, we continue with the right child of the root. We continue recursively until we reach a leaf. We define $Q(\sigma)$ to be the label of that leaf. When we reach an internal node, we say that Q *queries* or *reads* the corresponding variable. We say that Q *computes* a Boolean function $f : \{0, 1\}^n \to \{0, 1\}$, if for all $\sigma \in \{0, 1\}^n$, $Q(\sigma) = f(\sigma)$. The *cost of* Q *on input* σ, $\mathrm{cost}(Q; \sigma)$, is the number of variables queried when the input is σ. The *cost of* Q, $\mathrm{cost}(Q)$, is its *depth*, the maximum distance of a leaf from the root. The *deterministic complexity*, $D(f)$, of a Boolean function f is the minimum cost over all Boolean decision trees that compute f.

A *randomized Boolean decision tree* Q_{R} is a distribution p over deterministic decision trees. On input σ, a deterministic decision tree is chosen according to p and evaluated. The *cost of* Q_{R} *on input* σ is $\mathrm{cost}(Q_{\mathrm{R}}; \sigma) = \sum_Q p(Q)\,\mathrm{cost}(Q; \sigma)$. The *cost* of Q_{R} is $\max_\sigma \mathrm{cost}(Q_{\mathrm{R}}; \sigma)$. A randomized decision tree Q_{R} *computes* a Boolean function f, if $p(Q) > 0$ only when Q computes f. A randomized decision tree Q_{R} *computes* a Boolean function f *with error* δ, if, for all inputs σ, $Q_{\mathrm{R}}(\sigma) = f(\sigma)$ with probability at least $1 - \delta$. The *randomized complexity*, $R(f)$, of a Boolean function f is the minimum cost of any randomized Boolean decision tree that computes f. The *δ-error randomized complexity*, $R_\delta(f)$, of a Boolean function f, is the minimum cost of any randomized Boolean decision tree that computes f with error δ.

We are going to take a distributional view on randomized algorithms. Let μ be a distribution over $\{0, 1\}^n$ and Q_{R} a randomized decision tree. The *expected cost of* Q_{R} *under* μ is

$$\mathrm{cost}_\mu(Q_{\mathrm{R}}) = \textstyle\sum_\sigma \mu(\sigma)\,\mathrm{cost}(Q_{\mathrm{R}}; \sigma).$$

The *δ-error expected complexity under* μ, $R^\mu_\delta(f)$, of a Boolean function f, is the minimum expected cost under μ of any randomized Boolean decision tree that computes f with error δ. Clearly, $R_\delta(f) \geq R^\mu_\delta(f)$, for any μ, and thus we can prove lower bounds on randomized complexity by providing lower bounds for the expected cost under any distribution.

2.2 Introducing Cost-Functions

We are going to utilize the method of generalized costs of Saks and Wigderson [7]. To that end, we define a *cost-function* relative to a variable set Z, to be a function $\phi : \{0,1\}^n \times Z \to \mathbb{R}$. We extend the previous cost-related definitions as follows. The cost of a decision tree Q under cost-function ϕ on input σ is

$$\mathrm{cost}(Q;\phi;\sigma) = \textstyle\sum_{z\in S} \phi(\sigma; z),$$

where $S = \{z \mid z \text{ is queried by } Q \text{ on input } \sigma\}$. The cost of a randomized decision tree Q_{R} on input σ under cost-function ϕ is

$$\mathrm{cost}(Q_{\mathrm{R}};\phi;\sigma) = \textstyle\sum_{Q} p(Q)\,\mathrm{cost}(Q;\phi;\sigma),$$

where p is the corresponding distribution over deterministic decision trees. Finally, the expected cost of a randomized decision tree Q_{R} under cost-function ϕ and distribution μ is

$$\mathrm{cost}_{\mu}(Q_{\mathrm{R}};\phi) = \textstyle\sum_{\sigma} \mu(\sigma)\,\mathrm{cost}(Q_{\mathrm{R}};\phi;\sigma).$$

Fact 1. *Let ϕ and ψ be two cost-functions relative to Z. For any decision tree Q over Z, any assignment σ to Z, and any $a, b \in \mathbb{R}$, we have*

$$a\,\mathrm{cost}(Q;\phi;\sigma) + b\,\mathrm{cost}(Q;\psi;\sigma) = \mathrm{cost}(Q; a\phi + b\psi; \sigma).$$

For $\phi, \psi : \{0,1\}^n \times Z \to \mathbb{R}$, we write $\phi \geq \psi$, if for all $(\sigma, z) \in \{0,1\}^n \times Z$, $\phi(\sigma, z) \geq \psi(\sigma, z)$.

Fact 2. *Let ϕ and ψ be two cost-functions relative to Z. For any decision tree Q over Z and any assignment σ to Z, if $\phi \geq \psi$, then* $\mathrm{cost}(Q;\phi;\sigma) \geq \mathrm{cost}(Q;\psi;\sigma)$.

2.3 Definitions Pertaining to Trees

For a rooted tree T, the *depth* of a leaf is the number of edges on the path to the root. The *depth* of the tree is the maximum depth of a leaf. We denote by L_T the set of its leaves and by V_T the set of its internal nodes. Define the set of *leaf-parents* of T, P_T, as the set of all nodes in V_T all of whose children are leaves. For $S \subseteq P_T$ let $L_T(S)$ be the set of the leaves of the nodes in S. We call a tree *uniform* if all the leaves are on the same level. A tree such that every node has exactly three children is called *ternary*. For a positive integer d, let U_d denote the uniform ternary tree of depth d.

In the following, let T denote a ternary tree with n leaves. We define a distribution μ_T over $\{0,1\}^n$ that is placing positive weight on inputs that we consider difficult; we call these inputs reluctant, in accordance with [7]. Let

$$M_0 = \{(0,0,1),(0,1,0),(1,0,0)\} \text{ and } M_1 = \{(0,1,1),(1,0,1),(1,1,0)\}.$$

In the following definition we view T as a circuit with every internal node labeled by a maj-gate. We denote the corresponding function by F_T.

Definition 3. *Call an input to a ternary tree* reluctant*, if it is such that the inputs to every gate belong to $M_0 \cup M_1$. Let μ_T, the* reluctant distribution for T*, be the uniform distribution over all reluctant inputs. We write $\mu_d \equiv \mu_{U_d}$, and $\mu_T(0)$ ($\mu_T(1)$) for μ_T conditioned on the output of F_T being 0 (1).*

Suppose the inputs to a gate, under an assignment σ, belong to M_0 (M_1). We call an input to this gate a *minority under* σ if it has the value 1 (0) and a *majority* otherwise.

3 Proof Outline and Preliminaries

Our goal is to prove a lower bound on the expected cost of any randomized decision tree Q_{R} that computes maj_d with bounded error δ. We now discuss the outline of our proof. We start with the tree $T \equiv U_d$ that represents maj_d, the natural cost-function ψ that charges 1 for any query, and the reluctant distribution $\mu \equiv \mu_T$. We define a process that shrinks tree T to a smaller tree T' and a corresponding randomized decision tree Q'_{R} that computes $F_{T'}$ with bounded error δ. The crucial part is to show that for a "more expensive" cost-function ψ', $\mathrm{cost}_{\mu}(Q_{\mathrm{R}}; \psi) \geq \mathrm{cost}_{\mu'}(Q'_{\mathrm{R}}; \psi')$, where $\mu' \equiv \mu_{T'}$. Our goal is to apply the shrinking process repeatedly to the leaves of U_d, until we obtain a recurrence of the form

$$R_{\delta}^{\mu_d}(\mathrm{maj}_d) \geq \lambda \cdot R_{\delta}^{\mu_{d-1}}(\mathrm{maj}_{d-1}),$$

for some constant λ. The quality of our lower bound (i.e. the constant λ, since the recurrence will lead to $R_{\delta}(\mathrm{maj}_d) \in \Omega(\lambda^d)$) will depend on how much more expensive ψ' is than ψ.

The main ingredient in this framework is the shrinking process. A natural choice would be to shrink T by removing three leaves u, v, w so that their parent s would become a leaf in T'. Then, if we had a good algorithm Q for F_T we could design an algorithm Q' for $F_{T'}$ as follows. On input σs, Q' would simulate Q on one of the inputs $\sigma 01s$, $\sigma 10s$, $\sigma 0s1$, $\sigma 1s0$, $\sigma s01$, $\sigma s10$, with equal probability. We will show in the next section that such a shrinking process can give an alternate—and simpler—proof of the $\Omega(2.5^d)$ lower bound of Magniez, Nayak, Santha, and Xiao [6].

To improve their bound we are going to shrink nine leaves to three at a time instead of three to one. This is made precise by the following definition.

Definition 4 ($\mathrm{shrink}(T; s)$)**.** *For a ternary tree T, let s be the parent of $u, v, w \in P_T$. Define* $\mathrm{shrink}(T; s)$ *as the tree with the children of u, v, w removed.*

After shrinking our initial tree T to $T' \equiv \mathrm{shrink}(T; s)$ (notice that $u, v, w \in L_{T'}$), we need to define a randomized decision tree Q'_{R} that will compute $F_{T'}$ with error at most δ. We do so by defining for each deterministic tree Q that Q_{R} may choose, a randomized tree Q'.

Definition 5. *Let Q be any deterministic decision tree for F_T. We define a randomized decision tree Q' for $F_{T'}$, where $T' \equiv \mathrm{shrink}(T; s)$. The algorithm Q' on input σuvw chooses σ_u, σ_v, σ_w independently and uniformly at random from*

$\{x01, x10, 0x1, 1x0, 01x, 10x\}$, *where* x *is* u, v, w *respectively. Then,* Q' *simulates* Q *on input* $\hat{\sigma} = \sigma\sigma_u\sigma_v\sigma_w$. *This induces a randomized algorithm* Q'_{R} *for* $F_{T'}$.

Fact 6. *If* Q_{R} *is a* δ*-error randomized decision tree for* F_T, *then* Q'_{R} *is a* δ*-error randomized decision tree for* $F_{T'}$.

It will be useful to express $\mathrm{cost}_{\mu'}(Q'; \psi')$, for some cost-function ψ', in terms of Q. We have the following proposition.

Proposition 7. *For a ternary tree* T *and* s *the parent of* $u, v, w \in P_T$, *let* $T' \equiv \mathrm{shrink}(T; s)$. *Let* ψ' *be a cost-function on* T', *such that* $\psi'(\sigma; z) = \lambda$ *for all* $\sigma \in \{0,1\}^{|L_{T'}|}$ *and* $z \in \{u, v, w\}$. *Then*

$$\mathrm{cost}_{\mu'}(Q'; \psi') = \mathrm{cost}_{\mu}(Q; \psi^*),$$

where (μ, μ') *is any of* $(\mu_T, \mu_{T'})$, $(\mu_T(0), \mu_{T'}(0))$, $(\mu_T(1), \mu_{T'}(1))$, *and*

$$\psi^*(\sigma; z) = \begin{cases} \psi'(\sigma; z), & \text{if } z \in L_T \setminus L_T(u, v, w); \\ 0.5 \cdot \lambda, & \text{if } z \in L_T(u, v, w) \text{ and } z \text{ is a majority under } \sigma; \\ 0, & \text{if } z \in L_T(u, v, w) \text{ and } z \text{ is a minority under } \sigma. \end{cases} \quad (1)$$

Proof. Observe that by the definition of Q', $\Pr[\hat{\sigma} = \sigma] = \mu(\sigma)$. Furthermore, each σ is encountered 2^3 times over the random choices of Q'. For $(i, j, k) \in [3]^3$ define cost-functions for T as follows.

$$\psi_{(i,j,k)}(\sigma; z) = \begin{cases} 0, & \text{if } z \in L_T(u, v, w) \setminus \{u_i, v_j, w_k\}; \\ \lambda, & \text{if } z \in \{u_i, v_j, w_k\}; \\ \psi'(\sigma; z), & \text{otherwise.} \end{cases}$$

The indices i, j, k are going to play the role of x in Definition 5. We have

$$\mathrm{cost}_{\mu'}(Q'; \psi') = \textstyle\sum_{\sigma} \mu(\sigma) \frac{1}{2^3} \sum_{(i,j,k) \in I_\sigma} \mathrm{cost}(Q; \psi_{(i,j,k)}; \sigma),$$

where $I_\sigma = \{(i, j, k) \in [3]^3 \mid u_i, v_j, w_k \text{ are majorities under } \sigma\}$. The proposition follows since for any σ in the support of μ and any z, we have $\psi^*(\sigma, z) = \sum_{(i,j,k) \in I_\sigma} \frac{1}{2^3} \psi_{(i,j,k)}(\sigma, z)$. □

4 The $\Omega(2.5^d)$ Lower Bound and a Toy Problem

We sketch a proof of the $\Omega(2.5^d)$ lower bound of Magniez, Nayak, Santha, and Xiao [6], by applying the proof outline discussed in the previous section coupled with a simple shrinking process that shrinks three leaves to one. In addition, we define and analyze a toy problem that will play a crucial role in obtaining the improved $\Omega(2.55^d)$ bound.

For both of these tasks it is useful to define a cost-function ϕ_η, where $\eta \in \mathbb{R}$, as follows.

$$\phi_\eta(\sigma; z) = \begin{cases} 1, \text{ if } z \text{ is a minority under } \sigma; \\ \eta, \text{ otherwise.} \end{cases} \quad (2)$$

4.1 Proof of the $\Omega(2.5^d)$ Lower Bound

Let T be any ternary tree and x, y, z three of its leaves with a common parent u. Let T' be the ternary tree obtained by removing x, y, z, and thus transforming u to a leaf. Let ψ be a cost-function for T with $\psi(\sigma, x) = \psi(\sigma, y) = \psi(\sigma, z) = \lambda$ for any σ. Let ψ' be a cost-function for T' with $\psi'(\sigma, u) = 2.5 \cdot \lambda$ and $\psi'(\sigma, v) = \psi(\sigma, v)$ for any other leaf v. For any algorithm Q for F_T consider the algorithm Q' for $F_{T'}$ that on input σu outputs one of $Q(\sigma 01u)$, $Q(\sigma 10u)$, $Q(\sigma 0u1)$, $Q(\sigma 1u0)$, $Q(\sigma u01)$, $Q(\sigma u10)$ with equal probability. If Q is a δ-error algorithm for F_T, then Q' is a δ-error algorithm for $F_{T'}$. We claim that, with $\mu = \mu_T$ and $\mu' = \mu_{T'}$,

$$\mathrm{cost}_{\mu}(Q; \psi) \geq \mathrm{cost}_{\mu'}(Q'; \psi').$$

Accepting this claim, we start with $T \equiv U_d$ and apply it repeatedly by shrinking each time three leaves at depth d that are siblings. We end up with U_{d-1} and a cost function that charges 2.5 for each query. We have shown $R_{\delta}^{\mu_d}(\mathrm{maj}_d) \geq 2.5 \cdot R_{\delta}^{\mu_{d-1}}(\mathrm{maj}_{d-1})$. Repeating this d times we obtain $R_{\delta}^{\mu_d}(\mathrm{maj}_d) \geq 2.5^d \cdot R_{\delta}^{\mu_0}(\mathrm{maj}_0)$. Finally, it is not hard to show that you have to read a bit with probability at least $1-2\delta$ to be able to guess it with error at most δ, thus $R_{\delta}^{\mu_0}(\mathrm{maj}_0) \geq (1-2\delta)$. Putting these together, $R_{\delta}^{\mu_d}(\mathrm{maj}_d) \geq (1-2\delta) \cdot 2.5^d$.

To prove the claim, we observe that as in Proposition 7, $\mathrm{cost}_{\mu'}(Q'; \psi') = \mathrm{cost}_{\mu}(Q; \psi'')$, where

$$\psi''(\sigma; z) = \begin{cases} \psi'(\sigma; z), & \text{if } z \in L_T \setminus L_T(u); \\ 1.25 \cdot \lambda, & \text{if } z \in L_T(u) \text{ and } z \text{ is a majority under } \sigma; \\ 0, & \text{if } z \in L_T(u) \text{ and } z \text{ is a minority under } \sigma. \end{cases}$$

By Fact 1 it suffices to show that $\mathrm{cost}_{\mu}(Q; \psi - \psi'') \geq 0$. Note now that $\psi - \psi''$ is equal to $\lambda\phi_{-0.25}$ on x, y, z, and zero everywhere else. Thus, we can focus on how these three leaves are queried by Q and ignore all the other leaves. To do this, observe that if we fix values on the rest of the leaves, we obtain from Q a decision tree on three variables.

In the table below we list the deterministic decision trees[1] Q for three variables that are relevant to our problem. We label the input variables x, y, z, in the order they are queried. We write "and* z" to denote a conditional read. That is, z is queried only if the value of $\mathrm{maj}(x, y, z)$ cannot be determined from the values of x and y. Decision trees that read z even if $x = y$ are of no interest, neither for maj_d, nor for the toy problem we will consider in the next section.

In the last column we calculate $\sum_{\sigma \in M_0} \mathrm{cost}(Q; \phi_{\eta}; \sigma)$. Because of the symmetries involved we can look up the costs for $\sigma \in M_1$ as well. For example, the cost of the decision tree in row (2a) when $\sigma \in M_1$, is the same as the cost of the decision tree in row (2b) when $\sigma \in M_0$.

[1] We abuse the term "decision tree" here, since we are actually listing algorithms that query bits but do not output anything.

	Decision tree	Cost
(1)	If $x = 0$, stop; if $x = 1$, stop.	$1 + 2\eta$
(2a)	If $x = 0$, stop; if $x = 1$, read y.	$1 + 3\eta$
(2b)	If $x = 0$, read y; if $x = 1$, stop.	$2 + 3\eta$
(3a)	If $x = 0$, stop; if $x = 1$, read y and* z.	$1 + 4\eta$
(3b)	If $x = 0$, read y and* z; if $x = 1$, stop.	$2 + 4\eta$
(4)	If $x = 0$, read y; if $x = 1$, read y.	$2 + 4\eta$
(5a)	If $x = 0$, read y; if $x = 1$, read y and* z.	$2 + 5\eta$
(5b)	If $x = 0$, read y and* z; if $x = 1$, read y.	$2 + 5\eta$
(6)	If $x = 0$, read y and* z; if $x = 1$, read y and* z.	$2 + 6\eta$

What we are going to use from this table is that for $\eta \in [-0.5, 0]$ the decision tree of row (3a) has the minimum cost when $(x, y, z) \in M_0$, and the tree of row (3b) when $(x, y, z) \in M_1$. Their cost is $1 + 4\eta$. One can now verify that there is no (deterministic) decision tree Q that can achieve $\text{cost}_\mu(Q; \phi_{-0.25}) < 0$. This completes the proof of the claim and the sketch of the $\Omega(2.5^d)$ lower bound.

Remark. Note the role of the value of u in the above argument. In particular, if $u = 0$, then the best decision tree is the one on row (3a), whereas if $u = 1$, it is the one on row (3b). We can do better, if we only want a bound for maj_1.

Proposition 8. $R_\delta^{\mu_1}(\text{maj}_1) \geq \frac{8}{3} \cdot R_\delta^{\mu_0}(\text{maj}_0)$.

Proof. Let ψ_1 be the cost-function for maj_1 defined by $\psi_1(\sigma; z) = 1$ for all σ and z. Let ψ_0 be the cost-function for maj_0 defined by $\psi_0(0; u) = \psi_0(1; u) = 8/3$. Then, as in the proof of Proposition 7, we can show that $\text{cost}_{\mu_1}(Q_1; \psi_1) - \text{cost}_{\mu_0}(Q_0; \psi_0) = \text{cost}_{\mu_1}(Q_1; \phi_{-1/3})$. Observe now—by examining the table—that for any deterministic algorithm Q, $\sum_{\sigma \in M_0 \cup M_1} \text{cost}(Q; \phi_{-1/3}; \sigma) \geq 0$. The zero is achieved by the tree on row (6) of the table. Thus, $\text{cost}_{\mu_1}(Q_1; \psi_1) \geq \text{cost}_{\mu_0}(Q_0; \psi_0)$ and the result follows. □

4.2 The Toy Problem and a Corollary

Recall that in order to improve the lower bound, we need a shrinking process that shrinks nine leaves to three. Since it would be rather tedious to analyze decision trees on nine variables, we introduce a toy problem that reduces our analysis on decision trees over $\{0, 1\}^6$. We present first the toy problem and following its analysis a corollary that reveals its usefulness.

Let μ be the uniform distribution over $\big\{(u, v) \;\big|\; \big(u \in M_0 \wedge v \in M_1\big) \vee \big(u \in M_1 \wedge v \in M_0\big)\big\}$. We seek the minimum real η for which $\text{cost}_\mu(Q; \phi_\eta) \geq 0$ for any decision tree Q. We show that we can have $\eta = -0.3$. Although it is not stated in the following lemma, it is easily observable from the proof that this value is best possible. Although the analysis of the toy problem is optimal, one could improve the constant 2.55 by analyzing directly the decision trees over $\{0, 1\}^9$.

Lemma 9. *For any decision tree Q over $\{0,1\}^6$, $\mathrm{cost}_\mu(Q;\phi_{-0.3}) \geq 0$.*

Proof. For the proof we are going to do some case analysis, taking advantage of the symmetries involved. Denote the input by (x,y,z,u,v,w), and call (x,y,z) the left side and (u,v,w) the right side. Assume, without loss of generality (due to the symmetry of μ and the fact that we are calculating expected cost), that the variables on the left side are queried in the order x,y,z and on the right side in the order u,v,w. Assume further, that x is the first variable queried by Q, and let Q_0 (Q_1) be the decision tree if $x=0$ ($x=1$). We only analyze Q_0, as the analysis of Q_1 would be the same with the roles of 0 and 1 exchanged. Thus, we assume $x=0$ and proceed with the analysis of Q_0.

In all of the following cases we calculate the cost scaled; in particular, we calculate $C \equiv \sum_{\sigma:x=0} \mathrm{cost}(Q;\phi;\sigma)$.

Case 1. Suppose that Q_0 is empty. Then $C = 3+6\eta > 0$.

Case 2. Suppose that Q_0 queries y. Then, either $x=y$ or $x \neq y$. In the first case, we may assume Q_0 does not query z, since such a query increases the cost by 1. In the second case, we may assume Q_0 queries z, since such a query decreases the cost by $-\eta$. Therefore, the optimal Q_0 first "finishes" with the left side and then proceeds to the right side, knowing whether $(u,v,w) \in M_0$ or $(u,v,w) \in M_1$. In the first case, the optimal Q_0 continues with the right side as in row (3a) of the table; in the second case, as in row (3b). The cost is $C = (3\cdot 2\eta + 1 + 4\eta) + 2\cdot(3\cdot(1+2\eta)+1+4\eta)$, which is 0 for $\eta = -0.3$.

Case 3. Suppose that Q_0 queries u.

(i) Suppose $x=u$. If Q_0 does not query anything else, then this case contributes to the cost $4\cdot(1+\eta)$. Otherwise let us assume (without loss of generality) that it reads y. Then, as in Case 2, we may assume that Q_0 "finishes" the left side before doing anything else. There are four inputs such that $x=u=0$. For two of the inputs the left side belongs to M_0 and for the other two to M_1. In the first case, the optimal Q_0 reads v and w (they are both majorities). In the second case, it does not read any of v,w (it costs an additional $1+2\eta > 0$ if it reads them). In total the cost of this case is then $(1+4\eta)+(2+4\eta)+2\cdot(1+3\eta) = 5+14\eta$.

(ii) Suppose $x \neq u$. If Q_0 does not query anything else, then this case contributes to the cost $2+8\eta$. Otherwise let us assume (without loss of generality) that it reads y. With similar considerations as in case 3(i), we find that the total cost of this case is then $(2+4\eta)+2\cdot 3\eta + 2\cdot(1+3\eta) = 4+16\eta$.

Summing up for case 3, we find that the best Q_0 can do is $C = 9+30\eta = 0$. The case analysis is complete. □

We prove a corollary of this lemma that connects the toy problem to our real goal, which is the analysis of the process of shrinking nine leaves to three.

Lemma 10. *Let $T \equiv U_2$ with root s and $T' \equiv \mathrm{shrink}(T;s)$. Let ψ and ψ' be cost-functions such that $\psi(\sigma;z) = \lambda \geq 0$ for all $\sigma \in \{0,1\}^9$ and all variables $z \in L_T$, and $\psi'(\sigma;z) = 2.55\cdot\lambda$ for all $\sigma \in \{0,1\}^3$ and all variables $z \in L_{T'}$. Then, for any deterministic decision tree Q over $\{0,1\}^9$,*

$$\mathrm{cost}_\mu(Q;\psi) \geq \mathrm{cost}_{\mu'}(Q';\psi'),$$

where (μ,μ') is any of $(\mu_T,\mu_{T'})$, $(\mu_T(0),\mu_{T'}(0))$, $(\mu_T(1),\mu_{T'}(1))$.

Proof. Recall the definition of ψ^* from page 701. We have

$$\begin{aligned}
&\mathrm{cost}_\mu(Q;\psi) - \mathrm{cost}_{\mu'}(Q';\psi') \\
&\quad = \mathrm{cost}_\mu(Q;\psi) - \mathrm{cost}_\mu(Q;\psi^*) && \text{by Proposition 7} \\
&\quad = \mathrm{cost}_\mu(Q;\psi-\psi^*) && \text{by Fact 1} \\
&\quad = \textstyle\sum_{\sigma:\mathrm{maj}_2(\sigma)=0} \mu(\sigma)\,\mathrm{cost}(Q;\psi-\psi^*;\sigma) \\
&\qquad\qquad + \textstyle\sum_{\sigma:\mathrm{maj}_2(\sigma)=1} \mu(\sigma)\,\mathrm{cost}(Q;\psi-\psi^*;\sigma).
\end{aligned}$$

According to whether we are interested in μ_T, $\mu_T(0)$, or $\mu_T(1)$, one of the sums might be empty. Without loss of generality, we assume the first sum is nonempty and show it is nonnegative. The other sum can be treated similarly. To that end, we define an intermediate cost-function ξ. In the following definition, σ is an assignment, z a variable, and u is the value of the parent of z under σ.

$$\xi(\sigma;z) = \begin{cases} \lambda, & \text{if } z \text{ is a minority under } \sigma; \\ -0.275\cdot\lambda, & \text{if } z \text{ is a majority under } \sigma \text{ and } u=0; \\ -0.3\cdot\lambda, & \text{if } z \text{ is a majority under } \sigma \text{ and } u=1. \end{cases}$$

Observe that $\psi-\psi^* \geq \xi$ (they agree on minorities and $\psi-\psi^*$ is $\lambda - 0.5\cdot 2.55\cdot\lambda = -0.275\cdot\lambda$ on all majorities) and thus it suffices to show that

$$\textstyle\sum_{\sigma:\mathrm{maj}_2(\sigma)=0} \mu(\sigma)\,\mathrm{cost}(Q;\xi;\sigma) \geq 0. \tag{3}$$

We are going to decompose the above sum into terms that correspond either to the toy problem and Lemma 9 applies or correspond to queries over 3 variables and the table of Section 4.1 can be used.

To that end, we decompose ξ into several cost-functions. Let u, v, and w be the children of s. Define a cost-function ξ_u by

$$\xi_u(\sigma;z) = \begin{cases} 0, & \text{if } z \in L_T(u); \\ -0.3\cdot\lambda, & \text{if } z \text{ is a majority under } \sigma \text{ and } z \in L_T(v,w); \\ \lambda, & \text{if } z \text{ is a minority under } \sigma \text{ and } z \in L_T(v,w). \end{cases}$$

Similarly define ξ_v and ξ_w. For $\alpha \in M_0$ define

$$C_u(\alpha) \equiv \textstyle\sum_{\beta\in M_0}\sum_{\gamma\in M_1} \mu(\alpha\beta\gamma)\,\mathrm{cost}(Q;\xi_u;\alpha\beta\gamma) + \mu(\alpha\gamma\beta)\,\mathrm{cost}(Q;\xi_u;\alpha\gamma\beta).$$

Similarly define C_v and C_w (assigning α to v and w respectively). These terms—as shown later—correspond to the toy problem. Define a cost-function ξ'_u by

$$\xi'_u(\sigma;z) = \begin{cases} 0, & \text{if } z \in L_T(v,w); \\ -0.25\cdot\lambda, & \text{if } z \text{ is a majority under } \sigma \text{ and } z \in L_T(u); \\ \lambda, & \text{if } z \text{ is a minority under } \sigma \text{ and } z \in L_T(u). \end{cases}$$

Similarly define ξ'_v and ξ'_w. For $(\alpha,\beta) \in M_0 \times M_1$ define

$$C'_u(\alpha,\beta) \equiv \textstyle\sum_{\gamma\in M_0} \mu(\gamma\alpha\beta)\,\mathrm{cost}(Q;\xi'_u;\gamma\alpha\beta) + \mu(\gamma\beta\alpha)\,\mathrm{cost}(Q;\xi'_u;\gamma\beta\alpha).$$

Similarly define C'_v and C'_w. These terms will be analyzed using the table.

We now argue that we have the following decomposition

$$\sum_{\sigma:\mathrm{maj}_2(\sigma)=0} \mu(\sigma)\,\mathrm{cost}(Q;\xi;\sigma) = \frac{1}{2}\Bigg[\sum_{\alpha\in M_0}\Big(C_u(\alpha)+C_v(\alpha)+C_w(\alpha)\Big) + \sum_{\alpha\in M_0}\sum_{\beta\in M_1}\Big(C'_u(\alpha,\beta)+C'_v(\alpha,\beta)+C'_w(\alpha,\beta)\Big)\Bigg]. \quad (4)$$

To prove this, we fix a $\sigma = xyz$ on the left-hand side and see if each bit—assuming it is queried by Q—is charged the same in both sides of the equation. Without loss of generality, let us assume σ is such that $\mathrm{maj}(x) = \mathrm{maj}(y) = 0$ and $\mathrm{maj}(z) = 1$. A minority (under σ) is charged λ on the left side. A minority below u is charged λ once in $C_v(y)$ and once in $C'_u(y,z)$ on the right, for a total of $0.5\cdot(\lambda+\lambda)$. Similarly for a minority below v. A minority below w will be charged λ in $C_u(x)$ and $C_v(y)$, which corresponds to the amount charged on the left side. A majority below u will be charged $-0.3\cdot\lambda$ in $C_v(y)$ and $-0.25\cdot\lambda$ in $C'_u(y,z)$, for a total of $0.5\cdot(-0.3-0.25)\cdot\lambda$; this is how much is charged in the left side as well. Similarly for a majority below v. Finally, a majority below w is charged $-0.3\cdot\lambda$ in $C_u(x)$ and $C_v(y)$, equal to the amount charged on the left side.

We now finish the proof by showing that the right-hand side of (4) is nonnegative. We argue that, for any $\alpha\in M_0$, $C_u(\alpha)\geq 0$. Each fixed $\alpha\in M_0$ induces a decision tree Q_α over $\{0,1\}^6$ such that $Q_\alpha(\beta\gamma) = Q(\alpha\beta\gamma)$. Observe that ξ_u agrees on $L_T(v,w)$ with $\lambda\phi_{-0.3}$ (where $\phi_{-0.3}$ is defined in Equation 2). Thus, $C_u(\gamma) = \lambda\,\mathrm{cost}_\mu(Q_\gamma;\phi_{-0.3})$ and Lemma 9 shows that $C_u(\alpha)\geq 0$. Similarly, for any $\alpha\in M_0$, $C_v(\gamma)$, $C_w(\gamma)\geq 0$. Along similar lines we can show that, for any $(\alpha,\beta)\in M_0\times M_1$, $C'_u(\alpha,\beta)$, $C'_v(\alpha,\beta)$, $C'_w(\alpha,\beta)\geq 0$. (Lemma 9 is not needed in this case; inspection of the table in Section 4.1 suffices.) □

5 Proof of the Lower Bound

We prove a lemma that carries out the inductive proof sketched in Section 3.

Lemma 11 (Shrinking Lemma). *For a ternary tree T and s the parent of $u,v,w\in P_T$, let T' denote* $\mathrm{shrink}(T;s)$. *Let ψ and ψ' be cost-functions on T and T' such that $\psi(\sigma;z)=\lambda$ for all $\sigma\in\{0,1\}^{|L_T|}$ and $z\in L_T(u,v,w)$ and*

$$\psi'(\sigma;t) = \begin{cases} 2.55\cdot\lambda, & \text{if } t\in\{u,v,w\};\\ \psi(\sigma;t), & \text{otherwise.}\end{cases}$$

Then, for any randomized decision tree Q_{R}, $\mu\equiv\mu_T$ and $\mu'\equiv\mu_{T'}$,

$$\mathrm{cost}_\mu(Q_{\mathrm{R}};\psi)\geq\mathrm{cost}_{\mu'}(Q'_{\mathrm{R}};\psi').$$

Proof. Let n denote the number of leaves in T. Fix a partial assignment $\pi\in\{0,1\}^{n-9}$ for the leaves in $L_T\setminus L_T(u,v,w)$ and let $\rho\in\{0,1\}^9$. We write $\pi\rho$ for

the assignment that equals ρ on the variables $L_T(u,v,w)$ and π everywhere else. For any deterministic tree Q we have (recalling Proposition 7 and Fact 1)

$$\begin{aligned}\Delta(Q) &\equiv \mathrm{cost}_\mu(Q;\psi) - \mathrm{cost}_{\mu'}(Q';\psi') = \mathrm{cost}_\mu(Q;\psi) - \mathrm{cost}_\mu(Q;\psi^*)\\ &= \mathrm{cost}_\mu(Q;\psi-\psi^*) = \textstyle\sum_{\pi\rho}\mu(\pi\rho)\,\mathrm{cost}(Q;\psi-\psi^*;\pi\rho).\end{aligned}$$

Now, ψ and ψ^* are equal over $L_T \setminus L_T(u,v,w)$. Furthermore, having fixed π, we can define a deterministic tree Q_π over $\{0,1\}^9$ so that on input $\rho \in \{0,1\}^9$ we have $Q_\pi(\rho) = Q(\pi\rho)$. Thus $\Delta(Q) = \sum_{\pi\rho}\mu(\pi\rho)\,\mathrm{cost}(Q_\pi;\psi-\psi^*;\rho)$. Finally, recalling the definition of ψ^* (page 701), we see that $\psi - \psi^* \geq \lambda\phi_{-0.3}$ on $L_T(u,v,w)$ and by Fact 2, $\Delta(Q) \geq \sum_{\pi\rho}\mu(\pi\rho)\,\mathrm{cost}(Q_\pi;\lambda\phi_{-0.3};\rho)$. Thus, we may apply Lemma 10, which implies that, for each fixed π, each summand is greater or equal to zero. It follows that, for any Q, $\Delta(Q) \geq 0$. We have $\mathrm{cost}_\mu(Q_{\mathrm{R}};\psi) - \mathrm{cost}_{\mu'}(Q'_{\mathrm{R}};\psi') = \sum_Q p(Q)\Delta(Q) \geq 0$. □

Theorem 12. $R_\delta^{\mu_d}(\mathrm{maj}_d) \geq \frac{8}{3}\cdot(1-2\delta)\cdot 2.55^{d-1}$.

Proof. We start with $T \equiv U_d$ and apply the Shrinking Lemma repeatedly by shrinking each time nine leaves at depth d that have a common ancestor at depth $d-2$. We end up with U_{d-1} and a cost function that charges 2.55 for each query, obtaining $R_\delta^{\mu_d}(\mathrm{maj}_d) \geq 2.55\cdot R_\delta^{\mu_{d-1}}(\mathrm{maj}_{d-1})$. Repeating this $d-1$ times we get $R_\delta^{\mu_d}(\mathrm{maj}_d) \geq 2.55^{d-1}\cdot R_\delta^{\mu_1}(\mathrm{maj}_1)$. By Proposition 8, $R_\delta^{\mu_1}(\mathrm{maj}_1) \geq \frac{8}{3}\cdot R_\delta^{\mu_0}(\mathrm{maj}_0)$. A δ-error decision tree for maj_0 should guess a random bit with error at most δ; thus, $R_\delta^{\mu_0}(\mathrm{maj}_0) \geq 1-2\delta$. □

Acknowledgements. I thank Mike Saks for useful discussions and corrections in an earlier draft of this paper and Jeff Steif for pointing out a serious error in an earlier version. I also thank Carola Winzen and other reviewers for useful comments.

References

[1] Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: a survey. Theor. Comput. Sci. 288(1), 21–43 (2002)
[2] Heiman, R., Newman, I., Wigderson, A.: On read-once threshold formulae and their randomized decision tree complexity. Theor. Comput. Sci. 107(1), 63–76 (1993)
[3] Heiman, R., Wigderson, A.: Randomized vs. deterministic decision tree complexity for read-once boolean functions. Computational Complexity 1, 311–329 (1991)
[4] Jayram, T.S., Kumar, R., Sivakumar, D.: Two applications of information complexity. In: STOC, pp. 673–682. ACM (2003)
[5] Landau, I., Nachmias, A., Peres, Y., Vanniasegaram, S.: The lower bound for evaluating a recursive ternary majority function and an entropy-free proof. Undergraduate Research Reports, Department of Statistics, University of California, Berkeley (2006)

[6] Magniez, F., Nayak, A., Santha, M., Xiao, D.: Improved bounds for the randomized decision tree complexity of recursive majority. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 317–329. Springer, Heidelberg (2011)
[7] Saks, M.E., Wigderson, A.: Probabilistic boolean decision trees and the complexity of evaluating game trees. In: FOCS, pp. 29–38. IEEE Computer Society (1986)
[8] Santha, M.: On the monte carlo boolean decision tree complexity of read-once formulae. Random Struct. Algorithms 6(1), 75–88 (1995)
[9] Sherman, J.: Communicated to the author by Ryan O'Donnell in 12 September 2012 (unpublished, 2007)
[10] Snir, M.: Lower bounds on probabilistic linear decision trees. Theor. Comput. Sci. 38, 69–82 (1985)

A Quasi-Polynomial Time Partition Oracle for Graphs with an Excluded Minor*

Reut Levi[1,**] and Dana Ron[2,***]

[1] School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
reuti.levi@gmail.com
[2] School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel
danar@eng.tau.ac.il

Abstract. Motivated by the problem of testing planarity and related properties, we study the problem of designing efficient *partition oracles*. A *partition oracle* is a procedure that, given access to the incidence lists representation of a bounded-degree graph $G = (V, E)$ and a parameter ϵ, when queried on a vertex $v \in V$, returns the part (subset of vertices) which v belongs to in a partition of all graph vertices. The partition should be such that all parts are small, each part is connected, and if the graph has certain properties, the total number of edges between parts is at most $\epsilon|V|$. In this work we give a partition oracle for graphs with excluded minors whose query complexity is quasi-polynomial in $1/\epsilon$, thus improving on the result of Hassidim et al. (*Proceedings of FOCS 2009*) who gave a partition oracle with query complexity exponential in $1/\epsilon$. This improvement implies corresponding improvements in the complexity of testing planarity and other properties that are characterized by excluded minors as well as sublinear-time approximation algorithms that work under the promise that the graph has an excluded minor.

1 Introduction

An important and well studied family of graphs is the family of *Planar Graphs*. A natural problem is that of deciding whether a given graph $G = (V, E)$ is planar. Indeed, there is variety of linear-time algorithms for deciding planarity (e.g. [12,3]). However, what if one is willing to relax the decision task while requiring that the algorithm be much more efficient, and run in *sub-linear* time? Namely, here we refer to the notion of *Property Testing* where the goal is to decide (with high success probability) whether a graph has the property (planarity) or is far from having the property (in the sense that relatively many edges-modifications are required in order to obtain the property). Such a task should be performed by accessing only small portions of the input graph.

* A full version of this paper appears in [14].

** Research supported by the Israel Science Foundation grant nos. 1147/09 and 246/08.

*** Research supported by the Israel Science Foundation grant number 246/08.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 709–720, 2013.

Another type of problem related to planar graphs is that of solving a certain decision, search, or optimization problem, under the *promise* that the input graph is planar, where the problem may be hard in general. In some cases the problem remains hard even under the promise (e.g., Minimum Vertex-Cover [10]), while in other cases the promise can be exploited to give more efficient algorithms than are known for general graphs (e.g., Graph Isomorphism [13]). Here too we may seek even more efficient, sublinear-time, algorithms, which are allowed to output approximate solutions.

The problem of testing planarity, and, more generally, testing any minor-closed property of graphs[1] was first studied by Benjamini, Schramm and Shapira [2]. They gave a testing algorithm whose query complexity and running time are *independent* of $|V|$.[2] This result was later improved (in terms of the dependence on the distance parameter, ϵ) by Hassidim et al. [11], who also considered sublinear-time approximation algorithms that work under the promise that the graph has an excluded (constant size) minor (or more generally, for hyperfinite graphs as we explain subsequently). They show how to approximate the size of the minimum vertex cover, the minimum dominating set and the maximum independent set of such graphs, to within an additive term of $\epsilon|V|$ in time that depends only on ϵ and the degree bound, d, but not on $|V|$.

The main tool introduced by Hassidim et al. [11] for performing these tasks is *Partition Oracles*. Given query access to the incidence-lists representation of a graph, a partition oracle provides access to a partition of the vertices into small connected components. A partition oracle is defined with respect to a class of graphs, $\mathcal{C}$, and may be randomized. If the input graph belongs to $\mathcal{C}$, then with high probability the partition determined by the oracle is such that the number of edges between vertices in different parts of the partition is relatively small (i.e., at most $\epsilon|V|$). Such a bound on the number of edges between parts together with the bound on the size of each part lends itself to designing efficient testing algorithms and other sublinear approximation algorithms.

Hassidim et al. [11] provide a partition oracle for hyperfinite [8] classes of graphs that makes $2^{d^{\mathrm{poly}(1/\epsilon)}}$ queries to the graph, where d is and upper bound on the degree. A graph $G = (V, E)$ is (ϵ, k)-*hyperfinite* if it is possible to remove at most $\epsilon|V|$ edges of the graph so that the remaining graph has connected components of size at most k. A graph G is ρ-*hyperfinite* for $\rho : \mathbb{R}_+ \to \mathbb{R}_+$ if for every $\epsilon \in (0, 1]$, G is $(\epsilon, \rho(\epsilon))$-hyperfinite. For graphs with an excluded minor, (a special case of hyperfinite graphs), they provide a partition oracle with query complexity $d^{\mathrm{poly}(1/\epsilon)}$ (as detailed in [18, Sec. 2]). Hassidim et al. [11]

[1] For a fixed graph H, H is a *minor* of G if H is isomorphic to a graph that can be obtained by zero or more edge contractions on a subgraph of G. We say that a graph G is H-*minor free* (or *excludes H as a minor*) if H is not a minor of G. A property $\mathcal{P}$ (class of graphs) is *minor-closed* if every minor of a graph in $\mathcal{P}$ is also in $\mathcal{P}$. Any minor-closed property can be characterized by a finite family of excluded minors [22].

[2] Their algorithm has two-sided error. If one-sided error is desired, then for any fixed H that contains a simple cycle, the query complexity of one-sided error testing of H-minor freeness is $\Omega(\sqrt{|V|})$ [4].

leave as an open problem whether it is possible to design a partition oracle for graphs with an excluded minor that has query complexity polynomial in $1/\epsilon$. In particular, this would imply an algorithm for testing planarity whose complexity is polynomial in $1/\epsilon$.

1.1 Our Contribution

In this work we present a partition oracle for graphs with an excluded minor whose query complexity and running time are $(d/\epsilon)^{O(\log(1/\epsilon))} = d^{O(\log^2(1/\epsilon))}$, that is, quasi-polynomial in $1/\epsilon$.

Implications. Hassidim et al. [11] show how it is possible to reduce the problem of testing H-minor freeness (for a fixed graph H) to the problem of designing a partition oracle for H-minor free graphs. Using this reduction they obtain a testing algorithm for H-minor freeness (and more generally, for any minor-closed property) whose query complexity and running time are $2^{\mathrm{poly}(1/\epsilon)}$. As noted previously, this improves on the testing algorithm of Benjamini et al. [2] for minor-closed properties, whose complexity is $2^{2^{2^{\mathrm{poly}(1/\epsilon)}}}$. Using our partition oracle (and the reduction in [11]) we get a testing algorithm whose complexity is $2^{O(\log^2(1/\epsilon))}$.

Other applications of a partition oracle for a class of graphs $\mathcal{C}$ are constant time algorithms that work under the promise that the input graph belongs to $\mathcal{C}$, where in our case $\mathcal{C}$ is any class of graphs with an excluded minor. Under this promise, Hassidim et al. [11] provide constant time $\epsilon|V|$-additive-approximation algorithms for the size of a minimum vertex cover, minimum dominating set and maximum independent set. They also obtain an ϵ-additive-approximation algorithm for the distance from not having any graph in $\mathcal{H}$ as an induced subgraph where $\mathcal{H}$ is a fixed subset of graphs. Combined with our partition oracle, the query complexity of these algorithms drops from $d^{\mathrm{poly}(1/\epsilon)}$ to $(d/\epsilon)^{O(\log(1/\epsilon))} = d^{O(\log^2(1/\epsilon))}$.

Techniques. As in [11], our partition oracle runs a local emulation of a global partitioning algorithm. Hence, we first give a high-level idea of the global partitioning algorithm, and then discuss the local emulation. Our global partitioning algorithm is based on the global partitioning algorithm of [11] for graphs with an excluded minor, as described in [18, Sec. 2], which in turn builds on a clustering method of Czygrinow, Hańćkowiak, and Wawrzyniak [6]. The algorithm is also similar to the "Binary Borůvka" algorithm [20] for finding a minimum-weight spanning tree. The global algorithm works iteratively, coarsening the partition in each iteration. Initially each vertex is in its own part of the partition, and in each iteration some subsets of parts are merged into larger (connected) parts. The decisions regarding these merges are based on the numbers of edges between parts, as well as on certain random choices. Applying the analysis in [18] it is possible to show that with high constant probability, after $O(\log(1/\epsilon))$ iterations, the number of edges between parts is at most $\epsilon|V|$, as required. Since the sizes of the parts obtained after the last merging step may be much larger than desired,

as a final step it is possible to refine the partition without increasing the number of edges crossing between parts by too much by applying an algorithm of Alon Seymour and Thomas [1].

When turning to the local emulation of the global algorithm by the partition oracle, the query complexity and running time of the partition oracle depend on the sizes of the parts in the *intermediate* stages of the algorithm. These sizes are bounded as a function of $1/\epsilon$ and d, but can still be quite large. As an end-result, the partition oracle described in [18] has complexity that grows exponentially with $\text{poly}(1/\epsilon)$. To reduce this complexity, we modify the global partition algorithm as follows: If (following a merging stage) the size of a part goes above a certain threshold, we 'break' it into smaller parts. To this end we apply to each large part (in each iteration) the abovementioned algorithm of Alon Seymour and Thomas [1]. This algorithm finds (in graphs with an excluded minor) a relatively small vertex separator whose removal creates small connected components. Each such refinement of the partition increases the number of edges crossing between parts. However, we set the parameters for the algorithm in [1] so that the decrease in the number of edges between parts due to the merging steps dominates the increase due to the 'breaking' steps. One could have hoped that since the sizes of the parts are now always bounded by some polynomial in $1/\epsilon$ (and d), the complexity of the partition oracle will by $\text{poly}(d/\epsilon)$ as well. However, this is not the case, since in order to determine the part that a vertex, v, belongs to after a certain iteration, it is necessary to determine the parts that other vertices in the local neighborhood of v belong to in previous iterations. This leads to a recursion formula whose solution is quasi-polynomial in $1/\epsilon$.

1.2 Other Related Work

Yoshida and Ito [23] were the first to provide a testing algorithm for a minor-closed property whose complexity is polynomial in $1/\epsilon$ and d. They give a testing algorithm for the property of being outerplanar. Their result was generalized by Edelman et al. [7] who design a partition oracle with complexity $\text{poly}(d/\epsilon)$ for the class of bounded treewidth graphs. Known families of graphs with bounded treewidth include cactus graphs, outerplanar graphs and series-parallel graphs. However, many graphs with an excluded minor do not have bounded treewidth. For example, planar graphs are known to have treewidth of $\Omega(\sqrt{n})$.

Building on the partition oracle of [11], Newman and Sohler [17] design an algorithm for testing any property of graphs under the promise that the input graph is taken from $\mathcal{C}$, for any $\mathcal{C}$ that is a ρ-hyperfinite family of graphs. The number of queries their algorithm makes to the graph is independent of $|V|$ but is at least exponential in $1/\epsilon$. In a recent work, Onak [19] proves that there exists a property such that testing this property requires performing $2^{\Omega(1/\epsilon)}$ queries even under the promise that the input graph is taken from a hyperfinite family of graphs. This family of graphs $\mathcal{T}$ consists of graphs that are unions of bounded degree trees. Onak [19] defines a subclass of $\mathcal{T}$ and shows that every algorithm for testing the property of membership in this subclass must perform $2^{\Omega(1/\epsilon)}$ queries to the graph.

Czumaj, Shapira, and Sohler [5] investigated another promise problem. They proved that any hereditary property, namely a property that is closed under vertex removal, can be tested in time independent of the input size if the input graph belongs to a hereditary and non-expanding family of graphs.

As for approximation under a promise, Elek [9] proved that under the promise that the input graph, $G = (V, E)$, has sub-exponential growth and bounded degree, the size of the minimum vertex cover, the minimum dominating set and the maximum independent set, can be approximated up to an $\epsilon|V|$-additive error with time complexity that is independent of the graph size. Newman and Sohler [17] showed how to obtain an $\epsilon|V|$-additive approximation for a large class of graph parameters.

2 Preliminaries

In this section we introduce several definitions and some known results that will be used in the following sections. Unless stated explicitly otherwise, we consider simple graphs, that is, with no self-loops and no parallel edges. The graphs we consider have a known degree bound d, and we assume we have query access to their incidence-lists representation. Namely, for any vertex v and index $1 \leq i \leq d$ it is possible to obtain the i^{th} neighbor of v (where if v has less than i neighbors, then a special symbol is returned). If the graph is edge-weighted, then the weight of the edge is returned as well.

For a graph $G = (V, E)$ and two sets of vertices $V_1, V_2 \subseteq V$, we let $E(V_1, V_2)$ denote the set of edges in G with one endpoint in V_1 and one endpoint in V_2. That is $E(V_1, V_2) \stackrel{\text{def}}{=} \{(v_1, v_2) \in E : \ v_1 \in V_1, v_2 \in V_2\}$.

Definition 1 *Let $G = (V, E, w)$ be an edge-weighted graph and let $\mathcal{P} = (V_1, \ldots, V_t)$ be a partition of the vertices of G such that for every $1 \leq i \leq t$, the subgraph induced by V_i is connected. Define the* contraction *$G/\mathcal{P}$ of G with respect to the partition $\mathcal{P}$ to be the edge-weighted graph $G' = (V', E', w')$ where:*

1. *$V' = \{V_1, \ldots, V_t\}$ (that is, there is a vertex in V' for each subset of the partition $\mathcal{P}$);*
2. *$(V_i, V_j) \in E'$ if and only if $i \neq j$ and $E(V_i, V_j) \neq \emptyset$;*
3. *$w'((V_i, V_j)) = \sum_{(u,v) \in E(V_i, V_j)} w((u, v))$.*

As a special case of Definition 1 we get the standard notion of a single-edge contraction.

Definition 2 *Let $G = (V, E, w)$ be an edge-weighted graph on n vertices $v_1, \ldots, v_n$, and let (v_i, v_j) be an edge of G. The graph obtained from G by* contracting *the edge (v_i, v_j) is $G/\mathcal{P}$ where $\mathcal{P}$ is the partition of V into $\{v_i, v_j\}$ and singletons $\{v_k\}$ for every $k \neq i, j$.*

Definition 3 *For $\epsilon \in (0, 1]$, $k \geq 1$ and a graph $G = (V, E)$, we say that a partition $\mathcal{P} = (V_1, \ldots, V_t)$ of V is an* (ϵ, k)-partition *(with respect to G), if the following conditions hold:*

1. *For every* $1 \leq i \leq t$ *it holds that* $|V_i| \leq k$*;*
2. *For every* $1 \leq i \leq t$ *the subgraph induced by* V_i *in* G *is connected;*
3. *The total number of edges whose endpoints are in different parts of the partition is at most* $\epsilon|V|$ *(that is,* $|\{(v_i, v_j) \in E : v_i \in V_j, v_j \in V_j, i \neq j\}| \leq \epsilon|V|$*).*

Let $G = (V, E)$ be a graph and let $\mathcal{P}$ be a partition of V. We denote by $g_{\mathcal{P}}$ the function from $v \in V$ to 2^V (the set of all subsets of V), that on input $v \in V$, returns the subset $V_\ell \in \mathcal{P}$ such that $v \in V_\ell$.

Definition 4 ([11]) *An oracle* $\mathcal{O}$ *is a* partition oracle *if, given query access to the incidence-lists representation of a graph* $G = (V, E)$*, the oracle* $\mathcal{O}$ *provides query access to a partition* $\mathcal{P} = (V_1, \ldots, V_t)$ *of* V*, where* $\mathcal{P}$ *is determined by* G *and the internal randomness of the oracle. Namely, on input* $v \in V$*, the oracle returns* $g_{\mathcal{P}}(v)$ *and for any sequence of queries,* $\mathcal{O}$ *answers consistently with the same* $\mathcal{P}$*. An oracle* $\mathcal{O}$ *is an* (ϵ, k)-partition oracle with respect to a class of graphs $\mathcal{C}$ *if the partition* $\mathcal{P}$ *it answers according to has the following properties.*

1. *For every* $V_\ell \in \mathcal{P}$ *,* $|V_\ell| \leq k$ *and the subgraph induced by* V_ℓ *in* G *is connected.*
2. *If* G *belongs to* $\mathcal{C}$*, then* $|\{(u, v) \in E : g_{\mathcal{P}}(v) \neq g_{\mathcal{P}}(u)\}| \leq \epsilon|V|$ *with high constant probability, where the probability is taken over the internal coin flips of* $\mathcal{O}$*.*

By the above definition, if $G \in \mathcal{C}$, then with high constant probability the partition $\mathcal{P}$ is an (ϵ, k)-partition, while if $G \notin \mathcal{C}$ then it is only required that each part of the partition is connected and has size at most k. We are interested in partition oracles that have small query complexity, namely, that perform few queries to the graph (for each vertex they are queried on).

Recall that a graph H is called a *minor* of a graph G if H is isomorphic to a graph that can be obtained by zero or more edge contractions on a subgraph of G. A graph G is *H-minor free* if H is not a minor of G. Mader [15] proved that a sufficiently large average degree guarantees a K_t-minor. With combination with the theorem of Nash-Williams' [16] the follwing is obtained.

Fact 1 *Let* H *be a fixed graph. There is a constant* $c_1(|H|)$*, such that in every* H*-minor free graph,* $G = (V, E)$*, it holds that* $|E| \leq c_1(|H|) \cdot |V|$ *and that* E *can be partitioned into at most* $c_1(H)$ *forests.*

The following corollary will play a central role in this work. It follows from [1, Proposition 4.1] (see proof in the full version of this paper [14]):

Corollary 1 *Let* H *be a fixed graph. There is a constant* $c_2(H) > 1$ *such that for every* $\gamma \in (0, 1]$*, every* H*-minor free graph* $G = (V, E)$ *with degree bounded by* d *is* $(\gamma, c_2(H)d^2/\gamma^2)$*-hyperfinite. Furthermore, a* $(\gamma, c_2(H)d^2/\gamma^2)$ *partition of* V *can be found in time* $O(|V|^{3/2})$*.*

3 A Global Partitioning Algorithm

Our partition oracle is local, in the sense that its output is determined by the local neighborhood of the vertex it is queried on. However, as in previous work,

the oracle is based on a global partitioning algorithm, which accesses the whole graph, and the oracle emulates this algorithm locally. In this section we describe this global partition algorithm. As noted in the introduction, our algorithm and its analysis are based on [18] (which in turn builds on a clustering method of Czygrinow, Hańćkowiak, and Wawrzyniak [6], and is also similar to the "Binary Borůvka" algorithm [20] for finding a minimum-weight spanning tree).

The algorithm proceeds in iterations, where in iteration i it considers a graph G^{i-1}, where G^{i-1} is edge-weighted. The vertices of G^{i-1} correspond to (disjoint) subsets of vertices that induce connected subgraphs in G, and the weight of an edge between two vertices in G^{i-1} is the number of edges in G between the two corresponding subsets of vertices. Initially, the underlying graph G^0 is G and all edges have weight 1. In each iteration the algorithm contracts a subset of the edges so that each vertex in G^i corresponds to a subset that is the union of subsets of vertices that correspond to vertices in G^{i-1}. When the algorithm terminates it outputs the partition into subsets that correspond to the vertices of the final graph.

Each iteration of the algorithm consists of two phases. In the first phase of iteration i, a subset of the edges of G^{i-1} are contracted, resulting in a graph $\widetilde{G}^i$. In the second phase, some of the subsets that correspond to vertices in $\widetilde{G}^i$ remain as is, and some are 'broken' into smaller subsets. The vertices of G^i correspond to these subsets (both 'broken' and 'unbroken'). Observe that if the graph G is H-minor free for some fixed graph H, then every G^i and $\widetilde{G}^i$ are H-minor free as well.

In the first phase of iteration i, the contracted edges are selected randomly as follows. Each vertex in G^{i-1} selects an incident edge with maximum weight and tosses a fair coin to be 'Heads' or 'Tails'. Each selected edge is contracted if and only if it is selected by a 'Heads' vertex and its other endpoint is a 'Tails' vertex. This way, in each iteration, the contracted edges form stars (depth-1 trees). Therefore, a vertex in the graph $\widetilde{G}^i$ that results from the contraction of edges in G^{i-1} as described above, corresponds to a subset of vertices in V that induces a connected subgraph in G, and $\widetilde{G}^i = G/\widetilde{\mathcal{P}}^i$ (recall Definition 1) where $\widetilde{\mathcal{P}}^i$ is this partition into subsets. Since each vertex in each $\widetilde{G}^i$ corresponds to a connected subgraph in G, we shall refer to the vertices of $\widetilde{G}^i$ as *connected components* (to be precise, they are connected components in the graph resulting from removing all edges in G the correspond to (weighted) edges in $\widetilde{G}^i$). In each iteration, following the contraction of edges, if the size of a component goes above a certain threshold, $k = \text{poly}(d/\epsilon)$, then the component is 'broken' into smaller connected components, each of size at most k. This is done using the algorithm referred to in Corollary 1, and G^i is the (edge-weighted) graph whose vertices correspond to the new components.

Theorem 2 *Let H be a fixed graph. If the input graph G is H-minor free and has degree bounded by d, then for any given $\epsilon \in (0, 1]$, Algorithm 3 outputs an $(\epsilon, O(d^2/\epsilon^2))$-partition of G with high constant probability.*

Algorithm 1. A global $(\epsilon, c_2(H)d^2/\epsilon^2)$-partition algorithm for an H-minor free graph $G = (V, E)$

1. Set $G^0 := G$
2. For $i = 1$ to $\ell = \Theta(\log 1/\epsilon)$:
   (a) Toss a fair coin for every vertex in G^{i-1}.
   (b) For each vertex u let (u, v) be an edge with maximum weight that is incident to u (where ties are broken arbitrarily). If u's coin toss is 'Heads' and v's coin toss is 'Tails', then contract (u, v).
   (c) Let $\widetilde{G}^i = (\widetilde{V}^i, \widetilde{E}^i, \widetilde{w}^i)$ denote the graph resulting from the contraction of the edges as determined in the previous step. Hence, each vertex $\widetilde{v}^i_j \in \widetilde{V}^i$ corresponds to a subset of vertices in G, which we denote by $\widetilde{C}^i_j$.
   (d) Let $\gamma = \epsilon/(3\ell)$. For each $\widetilde{C}^i_j$ such that $|\widetilde{C}^i_j| > c_2(H)/\gamma^2$, partition the vertices in $\widetilde{C}^i_j$ into connected subsets of size at most $k = c_2(H)d^2/\gamma^2$ each by running the algorithm referred to in Corollary 1 on the subgraph induced by $\widetilde{C}^i_j$ in G.
   (e) Set $G^i := G/\mathcal{P}^i$, where $\mathcal{P}^i$ is the partition resulting from the previous step.
3. For each subset C^ℓ_j in $\mathcal{P}^\ell$ such that $|C^\ell_j| > c_2(H)d^2/\epsilon^2$, partition the vertices in C^ℓ_j into connected subsets each of size at most $3c_2(H)d^2/\epsilon^2$ by running the algorithm referred to in Corollary 1, and output the resulting partition.

Proof: We first claim that in each iteration, after Step 2b, the total weight of the edges in the graph is decreased by a factor of $\left(1 - \frac{1}{8c_1(H)}\right)$ with probability at least $\frac{1}{8c_1(H)-1}$, where the probability is taken over the coin tosses of the algorithm. Fixing an iteration i, by Fact 1 we know that the edges of G^{i-1} can be partitioned into at most $c_1(H)$ forests. It follows that one of these forests contains edges with total weight at least $\frac{w(G^{i-1})}{c_1(H)}$ where $w(G^{i-1})$ denotes the total weight of the edges in G^{i-1}. Suppose we orient the edges of the forest from roots to leaves, so that each vertex in the forest has in-degree at most 1. Recall that for each vertex v, the edge selected by v in Step 2b is the heaviest among its incident edges. It follows that the expected total weight of edges contracted in Step 2b is at least $\frac{w(G^{i-1})}{c_1(H)}$ (recall that an edge (v, u) selected by v is contracted if the coin flip of v is 'Heads' and that of u is 'Tails', an event that occurs with probability 1/4). Thus, the expected total weight of edges that are *not* contracted is at most $w(G^{i-1}) - \frac{w(G^{i-1})}{4c_1(H)} = \left(1 - \frac{1}{4c_1(H)}\right) w(G^{i-1})$. By Markov's inequality the probability that the total weight of edges that are not contracted is at least $\left(1 - \frac{1}{8c_1(H)}\right) w(G^{i-1})$ is at most $1 - \frac{1}{8c_1(H)-1}$. We say that an iteration i is *successful* if $w(\widetilde{G}^i) \leq \left(1 - \frac{1}{8c_1(H)}\right) w(G^{i-1})$. Using martingale analysis it can be shown that with probability at least 9/10, the number of successful iterations is at least $\frac{\ell}{16c_1(H)-2}$ (the full analysis appears in [14]).

Our second claim is that for every i, after Step 2d, it holds that $w(G^i) \leq w(\widetilde{G}^i) + \frac{\epsilon n}{3\ell}$ (where $n = |V|$). This follows from Corollary 1: For each component $\widetilde{C}^i_j$ that we break, we increase the total weight of edges between components by

an additive term of at most $\gamma|\widetilde{C}_j^i| = \frac{\epsilon|\widetilde{C}_j^i|}{3\ell}$. Thus, after Step 2d of the ℓ^{th} iteration, with probability at least 9/10, the weight of the edges in G^ℓ is upper bounded by $2\epsilon n/3$. Since we add at most $\epsilon n/3$ weight in Step 3 (when breaking the subsets corresponding to vertices in G^ℓ into subsets of size at most $3c_2(H)d^2/\epsilon^2$), we obtain the desired result. ■

4 The Partition Oracle

In this section we describe how, given query access to the incidence-lists representation of a graph $G = (V, E)$ and a vertex $v \in V$, it is possible to emulate Algorithm 3 locally and determine the part that v belongs to in the partition $\mathcal{P}$ that the algorithm outputs. Namely, we prove the following theorem.

Theorem 3 *For any fixed graph H there exists an $(\epsilon, O(d^2/\epsilon^2))$-partition-oracle for H-minor free graphs that makes $(d/\epsilon)^{O(\log(1/\epsilon))}$ queries to the graph for each query to the oracle. The total time complexity of a sequence of q queries to the oracle is $q \log q \cdot (d/\epsilon)^{O(\log(1/\epsilon))}$.*

Proof: Recall that the partition $\mathcal{P}$ is determined randomly based on the 'Heads'/'Tails' coin-flips of the vertices in each iteration. Since we want the oracle to be efficient, the oracle will flip coins "on the fly" as required for determining $g_{\mathcal{P}}(v)$. Since the oracle has to be consistent with the same $\mathcal{P}$ for any sequence of queries it gets, it will keep in its memory all the outcomes of the coin flips it has made. For the sake of simplicity, whenever an outcome of a coin is required, we shall say that a coin is flipped, without explicitly stating that first the oracle checks whether the outcome of this coin flip has already been determined. We shall also explain subsequently how to break ties (deterministically) in the choice of a heaviest incident edge in each iteration of the algorithm.

Recall that the algorithm constructs a sequence of graphs $G^0 = G$, $\widetilde{G}^1$, G^1, $\ldots, \widetilde{G}^\ell, G^\ell$, and that for each $0 \leq i \leq \ell$, the vertices in G^i correspond to connected subgraphs of G (which we refer to as components). For a vertex $v \in V$ let $C^i(v)$ denote the vertex/component that v it belongs to in G^i, and define $\widetilde{C}^i(v)$ analogously with respect to $\widetilde{G}^i$. Indeed, we shall refer to vertices in G^i ($\widetilde{G}^i$) and to the components that correspond to them, interchangeably. When the algorithm flips a coin for a vertex C in G^i, we may think of the coin flip as being associated with the vertex having the largest id (according to some arbitrary ordering) in the corresponding component in G. When the algorithm selects a heaviest edge incident to C and there are several edges $(C, C_1), \ldots, (C, C_r)$ with the same maximum weight, it breaks ties by selecting the edge (C, C_j) for which C_j contains the vertex with the largest id (according to the same abovementioned arbitrary ordering). We can then refer to (C, C_j) as *the* heaviest edge incident to C. In particular, since in G^0 all edges have the same weight, the heaviest edge incident to a vertex u in G^0 is the edge (u, y) for which y is maximized.

Let $Q^i(v)$ denote the number of queries to G that are performed in order to determine $C^i(v)$, and let Q^i denote an upper bound on $Q^i(v)$ that holds for

any vertex v. We first observe that $Q^1 \leq d^2$. In order to determine $C^1(v)$, the oracle first flips a coin for v. If the outcome is 'Tails' then the oracle queries the neighbors of v. For each neighbor u of v it determines whether (u, v) is the heaviest edge incident to u (by querying all of u's neighbors). If so, it flips a coin for u, and if the outcome is 'Heads', then the edge is contracted (implying that $u \in \widetilde{C}^1(v)$). If v is a 'Heads' vertex, then it finds its heaviest incident edge, (v, u) by querying all of v's neighbors. If u is a 'Tails' vertex (so that (v, u) is contracted), then the oracle queries all of u neighbors, and for each neighbor it queries all of its neighbors. By doing so (and flipping all necessary coins) it can determine which additional edges (u, y) incident to u are contracted (implying for each that $y \in \widetilde{C}^1(v) = \widetilde{C}^1(u)$). In both cases (of the outcome of v's coin flip), the number of queries performed to G is at most d^2. Recall that a component as constructed above is 'broken' if it contains more than $k = \tilde{O}(d^2/\epsilon^2)$ vertices. Since $|\widetilde{C}^1(v)| \leq d + 1$ for every v, we have that $C^1(v) = \widetilde{C}^1(v)$.

For general $i > 1$, to determine the connected component that a vertex v belongs to after iteration i, we do the following. First we determine the component it belongs to after iteration $i - 1$, namely $C^{i-1}(v)$, at a cost of at most Q^{i-1} queries. Note that by the definition of the algorithm, $|C^{i-1}(v)| \leq k$. We now have two cases:

Case 1: $C^{i-1}(v)$ is a 'Tails' vertex for iteration i. In this case we query all edges incident to vertices in $C^{i-1}(v)$, which amounts to at most $d \cdot k$ edges. For each endpoint u of such an edge we find $C^{i-1}(u)$. For each $C^{i-1}(u)$ that is 'Heads' we determine whether its heaviest incident edge connects to $C^{i-1}(v)$ and if so the edge is contracted (so that $C^{i-1}(u) \subset \widetilde{C}^i(v)$). To do so, we need, again, to query all the edges incident to vertices in $C^{i-1}(u)$, and for each endpoint y of such an edge we need to find $C^{i-1}(y)$. The weight of each edge $(C^{i-1}(u), C^{i-1}(y))$ is $|E(C^{i-1}(u), C^{i-1}(y))|$ (and since all edges incident to vertices in $C^{i-1}(y)$ have been queried, this weight is determined). The total number of vertices x for which we need to find $C^{i-1}(x)$ is upper bounded by d^2k^2, and this is also an upper bound on the number of queries performed in order to determine the identity of these vertices.

Case 2: $C^{i-1}(v)$ is a 'Heads' vertex in iteration i. In this case we find its heaviest incident edge in G^{i-1}, as previously described for $C^{i-1}(u)$. Let C' denote the other endpoint in G^{i-1}. If C' is a 'Tails' vertex then we apply the same procedure to C' as described in Case 1 for $C^{i-1}(v)$ (that is, in the case that $C^{i-1}(v)$ is a 'Tails' vertex in G^{i-1}). The bound on the number of queries performed is also as in Case 1. In either of the two cases we might need to 'break' $\widetilde{C}^i(v)$ (in case $|\widetilde{C}^i(v)| > k$) so as to obtain $C^i(v)$. However, this does not require performing any additional queries to G since all edges between vertices in $\widetilde{C}^i(v)$ are known, and this step only contributes to the running time of the partition oracle. We thus get the following recurrence relation for Q^i: $Q^i = d^2 \cdot k^2 + d^2 \cdot k^2 \cdot Q^{i-1}$. Since $k = \text{poly}(d/\epsilon)$ we get that $Q^\ell \leq (d \cdot \text{poly}(d/\epsilon))^{2\ell} = (d/\epsilon)^{O(\log(1/\epsilon))}$, as claimed.

Finally, we turn to the running time. Let $T^i(v)$ denote the running time for determining $C^i(v)$. By the same reasoning as above we have that $T^i \leq O(d^2 \cdot k^2) \cdot T^{i-1} + B$ where B is an upper bound on the running time of breaking a

connected component at each iteration. From Corollary 1 we obtain that $B \leq (d \cdot k^2)^{3/2}$. Thus, the running time of the oracle is $(d/\epsilon)^{O(\log(1/\epsilon))}$ for a single query. As explained above, for the sake of consistency, the oracle stores its previous coin-flips. By using a balanced search tree to store the coin flips we obtain that the total running time of the oracle for a sequence of q queries is $q \log q \cdot (d/\epsilon)^{O(\log(1/\epsilon))}$, as claimed. ■

5 Applications

In this section we state the improved complexity for the applications, of the partition oracle, which are presented in [11]. We obtain an improvement either in the query complexity or in the time complexity for all their applications excluding the application of approximating the distance to hereditary properties in which case the improvement we obtain is not asymptotic.

- Hassidim et al. [11] show that for any fixed graph H there is a testing algorithm for the property of being H-minor free in the bounded-degree model that performs $O(1/\epsilon^2)$ queries to $\mathcal{O}$, where $\mathcal{O}$ is an $(\epsilon d/4, k)$-partitioning oracle for the class of H-minor free graphs with degree bounded by d, and has $O(dk/\epsilon + k^3/\epsilon^6)$ time complexity. By using the partition oracle from Theorem 3 we obtain that the query and time complexity of testing H-minor freeness (in the bounded-degree model) is improved from $d^{\text{poly}(1/\epsilon)}$ to $(d/\epsilon)^{O(\log 1/\epsilon)}$.
- Let $\mathcal{P}$ be a minor-closed property. According to [21], $\mathcal{P}$ can be characterized as a finite set of excluded minors. Let S denote this set. By taking the proximity parameter to be $\epsilon/|S|$ and applying the testing algorithm for minor-freeness on every minor in S we obtain that the query and time complexity of testing a minor-closed property in the bounded degree model is improved from $2^{\text{poly}(|S|/\epsilon)}$ to $(|S|/\epsilon)^{O(\log |S|/\epsilon)}$. In particular this implies a testing algorithm for planarity with complexity $(1/\epsilon)^{O(\log(1/\epsilon))}$.

The next approximation algorithms work under the promise that the input graph is a graph with an excluded minor (of constant size). Under this promise we obtain the following improvements in the query complexity while the time complexity remains unchanged (the former time complexity dominates the improvement in the time complexity of the partition oracle):

- Hassidim et al. [11] provide a constant time $\epsilon|V|$-additive-approximation algorithm for minimum vertex cover size, maximum independent set size, and the minimum dominating set size for any family of graphs with an efficient partition oracle. The algorithms makes $O(1/\epsilon^2)$ queries to the partition oracle. By using the partition oracle from Theorem 3, the query complexity of the approximation algorithms is improved from $d^{\text{poly}(1/\epsilon)}$ to $(d/\epsilon)^{O(\log 1/\epsilon)}$
- By Lemma 11 in [11], for any finite set of connected graphs $\mathcal{H}$, there is an ϵ-additive-approximation algorithm for the distance to the property of not having any graph in $\mathcal{H}$ as an induced subgraph, which makes $O(1/\epsilon^2)$ queries to the partition oracle. Hence, the query complexity of the algorithm is improved from $d^{\text{poly}(1/\epsilon)}$ to $(d/\epsilon)^{O(\log 1/\epsilon)}$.

References

1. Alon, N., Seymour, P.D., Thomas, R.: A separator theorem for graphs with an excluded minor and its applications. In: Proceedings of STOC, pp. 293–299 (1990)
2. Benjamini, I., Schramm, O., Shapira, A.: Every minor-closed property of sparse graphs is testable. In: Proceedings of STOC, pp. 393–402 (2008)
3. Boyer, J.M., Myrvold, W.H.: On the cutting edge: simplified $O(n)$ planarity by edge addition. JGAA 8(3), 241–273 (2004)
4. Czumaj, A., Goldreich, O., Ron, D., Seshahadri, C., Shapira, A., Sholer, C.: Finding cycles and trees in sublinear time. To Appear in RSA (2012), `http://arxiv.org/abs/1007.4230`
5. Czumaj, A., Shapira, A., Sohler, C.: Testing hereditary properties of nonexpanding bounded-degree graphs. SICOMP 38(6), 2499–2510 (2009)
6. Czygrinow, A., Hańćkowiak, M., Wawrzyniak, W.: Fast distributed approximations in planar graphs. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 78–92. Springer, Heidelberg (2008)
7. Edelman, A., Hassidim, A., Nguyen, H.N., Onak, K.: An efficient partitioning oracle for bounded-treewidth graphs. In: Goldberg, L.A., Jansen, K., Ravi, R., Rolim, J.D.P. (eds.) APPROX/RANDOM 2011. LNCS, vol. 6845, pp. 530–541. Springer, Heidelberg (2011)
8. Elek, G.: The combinatorial cost. Technical Report math/0608474, ArXiv (2006)
9. Elek, G.: Parameter testing in bounded degree graphs of subexponential growth. RSA 37(2), 248–270 (2010)
10. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete graphs problems. TCS 1(3), 237–267 (1976)
11. Hassidim, A., Kelner, J.A., Nguyen, H.N., Onak, K.: Local graph partitions for approximation and testing. In: Proceedings of FOCS, pp. 22–31 (2009)
12. Hopcroft, J.E., Tarjan, R.E.: Efficient planarity testing. JACM 21(4), 549–568 (1974)
13. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs (preliminary report). In: Proceedings of STOC, pp. 172–184 (1974)
14. Levi, R., Ron, D.: A quasi-polynomial time partition oracle for graphs with an excluded minor. CoRR, abs/1302.3417 (2013)
15. Mader, W.: Homomorphieeigenschaften und mittlere kantendichte von graphen. Mathematische Annalen 174(4), 265–268 (1967)
16. Nash-Williams, C.St.J.A.: Decomposition of finite graphs into forests. Journal of the London Mathematical Society s1-39(1), 12 (1964)
17. Newman, I., Sohler, C.: Every property of hyperfinite graphs is testable. In: Proceedings of STOC, pp. 675–684 (2011)
18. Onak, K.: New Sublinear Methods in the Struggle Against Classical Problems. PhD thesis, MIT (2010)
19. Onak, K.: On the complexity of learning and testing hyperfinite graphs (2012) (available from the author's website)
20. Pettie, S., Ramachandran, V.: Randomized minimum spanning tree algorithms using exponentially fewer random bits. TALG 4(1) (2008)
21. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Comb. Theory Ser. B 63(1), 65–110 (1995)
22. Robertson, N., Seymour, P.D.: Graph minors. XX. Wagner's conjecture. J. Comb. Theory Ser. B 92(1), 325–357 (2004)
23. Yoshida, Y., Ito, H.: Testing outerplanarity of bounded degree graphs. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010. LNCS, vol. 6302, pp. 642–655. Springer, Heidelberg (2010)

Fixed-Parameter Algorithms for Minimum Cost Edge-Connectivity Augmentation*

Dániel Marx[1] and László A. Végh[2]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI) Budapest, Hungary
dmarx@cs.bme.hu
[2] Department of Management, London School of Economics, London, UK
l.vegh@lse.ac.uk

Abstract. We consider connectivity-augmentation problems in a setting where each potential new edge has a nonnegative cost associated with it, and the task is to achieve a certain connectivity target with at most p new edges of minimum total cost. The main result is that the minimum cost augmentation of edge-connectivity from $k-1$ to k with at most p new edges is fixed-parameter tractable parameterized by p and admits a polynomial kernel. We also prove the fixed-parameter tractability of increasing edge-connectivity from 0 to 2, and increasing node-connectivity from 1 to 2.

1 Introduction

Designing networks satisfying certain connectivity requirements has been a rich source of computational problems since the earliest days of algorithmic graph theory: for example, the original motivation of Borůvka's work on finding minimum cost spanning trees was designing efficient electricity network in Moravia [22]. In many applications, we have stronger requirements than simply achieving connectivity: one may want to have connections between (certain pairs of) nodes even after a certain number of node or link failures. Survivable network design problems deal with such more general requirements.

In the simplest scenario, the task is to achieve k-edge-connectivity or k-node-connectivity by adding the minimum number of new edges to a given directed or undirected graph G. This setting already leads to a surprisingly complex theory and, somewhat unexpectedly, there are exact polynomial-time algorithms for many of these questions. For example, there is a polynomial-time algorithm for achieving k-edge-connectivity in an undirected graph by adding the minimum number of edges (Watanabe and Nakamura [24], see also Frank [7]). For k-node-connectivity, a polynomial-time algorithm is known only for the special case when the graph is already $(k-1)$-node-connected; the general case is still open [23]. We refer the reader to the recent book by Frank [8] on more results

* Full version available on Arxiv:1304.6593. The first author was supported by the European Research Council (ERC) grant *"PARAMTIGHT: Parameterized complexity and the search for tight complexity results,"* reference 280152.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 721–732, 2013.

of similar flavour. One can observe that increasing connectivity by one already poses significant challenges and in general the node-connectivity versions of these problems seem to be more difficult than their edge-connectivity counterparts.

For most applications, minimizing the number of new edges is a very simplified objective: for example, it might not be possible to realize direct connections between nodes that are very far from each other. A slightly more realistic setting is to assume that the input specifies a list of potential new edges ("links") and the task is to achieve the required connectivity by using the minimum number of links from this list. Unfortunately, almost all problems of this form turn out to be NP-hard: deciding if the empty graph on n nodes can be augmented to be 2-edge-connected with n new edges from a given list is equivalent to finding a Hamiltonian cycle (similar simple arguments can show the NP-hardness of augmenting to k-edge-connectivity also for larger k). Even though these problems are already hard, this setting is still unrealistic: it is difficult to imagine any application where all the potential new links have the same cost. Therefore, one typically tries to solve a minimum cost version of the problem, where for every pair u, v of nodes, a (finite or infinite) cost $c(u, v)$ of connecting u and v is given. When the goal is to achieve k-edge connectivity, we call this problem *Minimum Cost Edge-Connectivity Augmentation to k* (see Section 2 for a more formal definition). In the special case when the input graph is assumed to be $(k-1)$-edge-connected (as in e.g. [16,13,18,23]), we call the problem *Minimum Cost Edge-Connectivity Augmentation by One*. Alternatively, one can think of this problem with the edge-connectivity target being the minimum cut value of the input graph plus one. The same terminology will be used for the node-connectivity versions and the minimum cardinality variants (where every cost is either 1 or infinite).

Due to the hardness of the more general minimum cost problems, research over the last two decades has focused mostly on the approximability of the problem. This field is also known as survivable network design, e.g. [1,11,15,3,17,2]; for a survey, see [18]. In this paper, we approach these problems from the viewpoint of parameterized complexity. We say that a problem with parameter p is *fixed-parameter tractable (FPT)* if it can be solved in time $f(p) \cdot n^{O(1)}$, where $f(p)$ is an arbitrary computable function depending only on p and n is the size of the input [5,6]. The tool box of fixed-parameter tractability includes many techniques such as bounded search trees, color coding, bidimensionality, etc. The method that received most attention in recent years is the technique of kernelization [19,20]. A *polynomial kernelization* is a polynomial-time algorithm that produces an equivalent instance of size $p^{O(1)}$, i.e., polynomial in the parameter, but not depending on the size of the instance. Clearly, polynomial kernelization implies fixed-parameter tractability, as kernelization in time $n^{O(1)}$ followed by any brute force algorithm on the $p^{O(1)}$-size kernel yields a $f(p) \cdot n^{O(1)}$ time algorithm. The conceptual message of polynomial kernelization is that the hard problem can be solved by first applying a preprocessing to extract a "hard core" and then solving this small hard instance by whatever method available. An interesting example of fixed-parameter tractability in the context of connectivity

augmentation is the result by Jackson and Jordán [14], showing that for the problem of making a graph k-node-connected by adding a minimum number of arbitrary new edges admits a $2^{O(k)} \cdot n^{O(1)}$ time algorithm (it is still open whether there is a polynomial-time algorithm for this problem).

As observed above, if the link between arbitrary pair of nodes is not always available (or if they have different costs for different pairs), then the problem for augmenting a $(k-1)$-edge-connected graph to a k-edge-connected one is NP-hard for any fixed $k \geq 2$. Thus for these problems we cannot expect fixed-parameter tractability when parameterizing by k. In this paper, we consider a different parameterization: we assume that the input contains an integer p, which is a upper bound on the number of new edges that can be added. Assuming that the number p of new links is much smaller than the size of the graph, exponential dependence on p is still acceptable, as long as the running time depends only polynomially on the size of the graph. It follows from Nagamochi [21, Lemma 7] that *Minimum Cardinality Edge-Connectivity Augmentation from 1 to 2* is fixed-parameter tractable parameterized by this upper bound p. Guo and Uhlmann [12] showed that this problem, as well as its node-connectivity counterpart, admits a kernel of $O(p^2)$ nodes and $O(p^2)$ links. Neither of these algorithms seem to work for the more general minimum cost version of the problem, as the algorithms rely on discarding links that can be replaced by more useful ones. Arguments of this form cannot be generalized to the case when the links have different costs, as the more useful links can have higher costs. Our results go beyond the results of [21,12] by considering higher order edge-connectivity and by allowing arbitrary costs on the links.

We present a kernelization algorithm for the problem *Minimum Cost Edge-Connectivity Augmentation by One* for arbitrary k. The algorithm starts by doing the opposite of the obvious: instead of decreasing the size of the instance by discarding provably unnecessary links, we add new links to ensure that the instance has a certain closure property; we call instances satisfying this property *metric instances*. We argue that these changes do not affect the value of the optimum solution. The natural machinery for this approach is to work with a more general problem. Besides the costs, every link is equipped with a positive integer weight. Our task is to find a minimum cost set of links of total weight at most p whose addition makes the graph k-edge-connected. Our main result addresses the corresponding problem, *Weighted Minimum Cost Edge-Connectivity Augmentation*.

Theorem 1.1. *Weighted Minimum Cost Edge-Connectivity Augmentation by One admits a kernel of $O(p)$ nodes, $O(p)$ edges, $O(p^3)$ links, with all costs integers of $O(p^6 \log p)$ bits.*

The original problem is the special case when all links have weight one. Strictly speaking, Theorem 1.1 does not give a kernel for the original problem, as the kernel may contain links of higher weight even if all links in the input had weight one. Our next theorem, which can be derived from the previous one, shows that we may obtain a kernel that is an unweighted instance. However, there is a trade-off in the bound on the kernel size.

Theorem 1.2. *Minimum Cost Edge-Connectivity Augmentation by One admits a kernel of $O(p^4)$ nodes, $O(p^4)$ edges and $O(p^4)$ links, with all costs integers of $O(p^8 \log p)$ bits.*

Let us now outline the main ideas of the proof of Theorem 1.1. We first show that every input can be efficiently reduced to a metric instance, one with the closure property. We first describe our algorithm in the special case of increasing edge-connectivity from 1 to 2, where connectivity augmentation can be interpreted as covering a tree by paths. The closure property of the instance allows us to prove that there is an optimum solution where every new link is incident only to "corner nodes" (leaves and branch nodes). Either the problem is infeasible, or we can bound the number of corner nodes by $O(p)$. Hence we can also bound the number of potential links in the resulting small instance.

Augmenting edge connectivity from 2 to 3 is similar to augmenting from 1 to 2, but this time the graph we need to work on is no longer a tree, but a cactus graph. Thus the arguments are slightly more complicated, but generally go along the same lines. Finally, in the general case of increasing edge-connectivity from $k-1$ to k, we use the uncrossing properties of minimum cuts and a classical result of Dinits, Karzanov, and Lomonosov [4] to show that we can assume that (depending on the parity of k) the problem can be always reduced to the case $k = 2$ or $k = 3$.

In kernels for the weighted problem, a further technical issue has to be overcome: each finite cost in the produced instance has to be a rational number represented by $p^{O(1)}$ bits. As we have no assumption on the sizes of the numbers appearing in the input, this is a nontrivial requirement. It turns out that a technique of Frank and Tardos [10] (used earlier in the design of strongly polynomial-time algorithms) can be straightforwardly applied here: the costs in the input can be preprocessed in a way that the each number is an integer of $O(p^6 \log p)$ bits long and the relative costs of the feasible solutions do not change. We believe that this observation is of independent interest, as this technique seems to be an essential tool for kernelization of problems involving costs.

To prove Theorem 1.2 (see the full version), we first obtain a kernel by applying our weighted result to our unweighted instance; this kernel will however contain links of weight higher than one. Still, every link f in the (weighted) kernel can be replaced by a sequence of $w(f)$ original unweighted edges. This replaces the $O(p^2)$ links by $O(p^4)$ original ones.

We try to extend our results in two directions. The results described next are proved only in the full version of the paper. First, we show that in the case of increasing connectivity from 1 to 2, the node-connectivity version can be directly reduced to the edge-connectivity version.

Theorem 1.3. *Weighted Minimum Cost Node-Connectivity Augmentation from 1 to 2 admits a a kernel of $O(p)$ nodes, $O(p)$ edges, $O(p^3)$ links, with all costs integers of $O(p^6 \log p)$ bits.*

For higher connectivities, we do not expect such a clean reduction to work. Polynomial-time exact and approximation algorithms for node-connectivity are typically much more involved than for edge-connectivity (compare e.g. [24] and [7] to [9] and [23]), and it is reasonable to expect that the situation is similar in the case of fixed-parameter tractability.

A natural goal for future work is trying to remove the assumption of Theorems 1.1 and 1.2 that the input graph is $(k-1)$-connected. In the case of 2-edge-connectivity, we show that the problem is fixed-parameter tractable even if the input graph is not connected. However, the algorithm uses nontrivial branching and it does not provide a polynomial kernel.

Theorem 1.4. *Minimum Cost Edge-Connectivity Augmentation to* 2 *can be solved in time* $2^{O(p \log p)} \cdot n^{O(1)}$.

The additional branching arguments needed in Theorem 1.4 can show a glimpse of the difficulties one can encounter when trying to solve the problem larger k, especially with respect to kernelization. For augmentation by one, the following notion of shadows was crucial to define the metric closure of the instances: f is a shadow of link e if the weight of e is at most that of f, and e covers every k-cut covered by f — in other words, link f can be automatically substituted by link e. When the input graph is not assumed to be connected, we cannot extend the shadow relation to links connecting different components, only in special, restricted situations. Therefore, we cannot prove the existence of an optimal solution with all links incident to corner nodes only. Instead, we prove that there is an optimal solution such that all leaves are adjacent to either corner nodes or certain other special nodes; this enables the branching in the FPT algorithm. A further difficulty arises if we want to avoid using two copies of the same link. This was automatically excluded for augmentation by one, whereas now further efforts are needed to enforce this.

2 Preliminaries

For a set V, let $\binom{V}{2}$ denote the edge set of the complete graph on V. Let $n = |V|$ denote the number of nodes. For a set $X \subseteq V$ and $F \subseteq \binom{V}{2}$, let $d_F(X)$ denote the number of edges $e = uv \in F$ with $u \in X$, $v \in V \setminus X$. When we are given a graph $G = (V, E)$ and it is clear from the context, $d(X)$ will denote $d_E(X)$. A set $\emptyset \neq X \subsetneq V$ will be called a *cut*, and *minimum cut* if $d(X)$ takes the minimum value. For a function $z : V \to \mathbb{R}$, and a set $X \subseteq V$, let $z(X) = \sum_{v \in X} z(v)$ (we use the same notation with functions on edges as well). For $u, v \in V$, a set $X \subseteq V$ is called an $u\bar{v}$-set if $u \in X$, $v \in V \setminus X$.

Let us be given an undirected graph $G = (V, E)$ (possibly containing parallel edges), a connectivity target $k \in \mathbb{Z}_+$, and a cost function $c : \binom{V}{2} \to \mathbb{R}_+ \cup \{\infty\}$. For a given nonnegative integer p, our aim is to find a minimum cost set of edges $F \subseteq \binom{V}{2}$ of cardinality at most p such that $(V, E \cup F)$ is k-edge-connected.

We will work with a more general version of this problem. Let E^* denote an edge set on V, possibly containing parallel edges. We call the elements of E *edges*

and all edges in E^* *links*. Besides the cost function $c : E^* \to \mathbb{R}_+ \cup \{\infty\}$, we are also given a positive integer weight function $w : E^* \to \mathbb{Z}_+$. We restrict the total weight of the augmenting edge set to be at most p instead of its cardinality. Let us define our main problem.

Weighted Minimum Cost Edge Connectivity Augmentation	
Input:	Graph $G = (V, E)$, set of links E^*, integers $k, p > 0$, weight function $w : E^* \to \mathbb{Z}_+$, cost function $c : E^* \to \mathbb{R}_+ \cup \{\infty\}$.
Find:	minimum cost link set $F \subseteq E^*$ such that $w(F) \leq p$ and $(V, E \cup F)$ is k-edge-connected.

A problem instance is thus given by (V, E, E^*, c, w, k, p). An $F \subseteq E^*$ for which $(V, E \cup F)$ is k-edge-connected is called an *augmenting link set*. If all weights are equal to one, we simply refer to the problem as *Minimum Cost Edge Connectivity Augmentation*.

An edge between $x, y \in V$ will be denoted as xy. For a link f, we use $f = (x, y)$ if it is a link between x and y; note that there might be several links between the same nodes with different weights. We may ignore all links of weight $> p$. If for a pair of nodes $u, v \in V$, there are two links e and f between u and v such that $c(e) \leq c(f)$ and $w(e) \leq w(f)$, then we may also ignore the link f. It is convenient to assume that for every value $1 \leq t \leq p$ and every two nodes $u, v \in V$, there is exactly one link e between u and v with $w(e) = t$ (if there is no such link in the input E^*, we can add one of cost ∞). This e will be referred to as the *t-link between u and v*. With this convention, we will assume that E^* consists of exactly p copies of $\binom{V}{2}$: a t-link between any two nodes $u, v \in V$ for every $1 \leq t \leq p$. However, in the input links of infinite cost should not be listed.

For a set $S \subseteq V$, by G/S we mean the contraction of S to a single node s. That is, the node set of the contracted graph is $(V - S) \cup \{s\}$, and every edge uv with $u \notin S$, $v \in S$ is replaced by an edge us (possibly creating parallel edges); edges inside S are removed. Note that S is not assumed to be connected. We also contract the links to E^*/S accordingly. If multiple t-links are created between s and another node, we keep only one with minimum cost.

We say that two nodes x and y are *k-inseparable* if there is no $x\bar{y}$-set X with $d(X) < k$. By Menger's theorem, this is equivalent to the existence of k edge-disjoint paths between x and y; this property can be tested in polynomial time by a max flow-min cut computation. Let us say that the node set $S \subseteq V$ is *k-inseparable* if any two nodes $x, y \in S$ are k-inseparable. It is easy to verify that being k-inseparable is an equivalence relation. The maximal k-inseparable sets hence give a partition of the node set V. The following proposition provides us with a preprocessing step that can be used to simplify the instance:

Proposition 2.1. *For a problem instance (V, E, E^*, c, w, k, p), let $S \subseteq V$ be a k-inseparable set of nodes. Let us consider the instance obtained by the contraction of S. Assume $\bar{F} \subseteq E^*/S$ is an optimal solution to the contracted problem. Then the pre-image of $\bar{F}$ in E^* is an optimal solution to the original problem.*

Note that contracting a k-inseparable set S does not affect whether $x, y \notin S$ are k-inseparable. Thus by Proposition 2.1, we can simplify the instance by contracting each class of the partition given by the k-inseparable relation. Observe that after such a contraction, there are no longer any k-inseparable pair of nodes. Thus we may assume in our algorithms that every pair of nodes can be separated by a cut of size smaller than k.

3 Augmenting Edge Connectivity by One

3.1 Metric Instances

The following notions will be used for augmenting edge-connectivity from 1 to 2 and from 2 to 3. We formulate them here in a generic way. Assume the input graph is $(k-1)$-edge-connected. Let $\mathcal{D}$ denote the set of all minimum cuts, represented by the node sets. That is, $X \in \mathcal{D}$ if and only if $d(X) = k-1$. Note that, by the minimality of the cut, both X and $V \setminus X$ induce connected graphs if $X \in \mathcal{D}$. For a link $e = (u, v) \in E^*$, let us define $\mathcal{D}(e) \subseteq \mathcal{D}$ as the subset of minimum cuts *covered* by e. That is, $X \in \mathcal{D}$ is in $\mathcal{D}(e)$ if and only if X is an $u\bar{v}$-set or a $v\bar{u}$-set. Clearly, augmenting edge-connectivity by one is equivalent to covering all the minimum cuts of the graph.

Proposition 3.1. *Assume (V, E) is $(k-1)$-edge-connected. Then $(V, E \cup F)$ is k-edge-connected if and only if $\cup_{e \in F} \mathcal{D}(e) = \mathcal{D}$.*

The following definition identifies the class of metric instances that plays a key role in our algorithm.

Definition 3.2. *We say that the link f is a* shadow *of link e, if $w(f) \geq w(e)$ and $\mathcal{D}(f) \subseteq \mathcal{D}(e)$. The instance (V, E, E^*, c, w, k, p) is* metric*, if*

(i) $c(f) \leq c(e)$ holds whenever the link f is a shadow of link e.
(ii) Consider three links $e = (u, v)$, $f = (v, z)$ and $h = (u, z)$ with $w(h) \geq w(e) + w(f)$. Then $c(h) \leq c(e) + c(f)$.

Whereas the input instance may not be metric, we can create its metric completion with the following simple subroutine. Let us call the inequalities in *(i) shadow inequalities* and those in *(ii) triangle inequalities*. Let us define the *rank* of the inequality $c(f) \leq c(e)$ to be $w(f)$, and the rank of $c(h) \leq c(e) + c(f)$ to be $w(h)$. By *fixing* the triangle inequality $c(h) > c(e) + c(f)$, we mean decreasing the value of $c(h)$ to $c(e) + c(f)$.

The subroutine METRIC-COMPLETION(c) consists of p iterations, one for each $t = 1, 2, \ldots, p$. In the t'th iteration, first all triangle inequalities of rank t are taken in an arbitrary order, and the violated ones are fixed. That is, $c(h)$ is set to $\min\{c(h), c(e) + c(f)\}$. Then for every t-link f, we decrease $c(f)$ to the $\min\{c(e) : f$ is a shadow of $e\}$. Note that we perform these steps one after the other for every violated inequality: in each step, we decrease the cost of a single link f only (this will be important in the analysis of the algorithm). The first part of iteration 1 is void as there are no rank 1 triangle inequalities. The subroutine

can be implemented in polynomial time: the number of triangle inequalities is $O(p^3n^3)$, and they can be efficiently listed; further, every link is the shadow of $O(pn^2)$ other ones.

Lemma 3.3. *Consider a problem instance (V, E, E^*, c, w, k, p) with the graph (V, E) being $(k-1)$-edge-connected.* Metric-Completion(c) *returns a metric cost function $\bar{c}$ with $\bar{c}(e) \leq c(e)$ for every link $e \in E^*$. Moreover, if for a link set $\bar{F} \subseteq E^*$, graph $(V, E \cup \bar{F})$ is k-edge-connected, then there exists an $F \subseteq E^*$ such that $(V, E \cup F)$ is k-edge-connected, $c(F) \leq \bar{c}(\bar{F})$, and $w(F) \leq w(\bar{F})$. Consequently, an optimal solution for $\bar{c}$ provides an optimal solution for c.*

The proof (see full version) proceeds by showing that after iteration t, all rank t inequalities are satisfied and they remain satisfied later on. The proof also provides an efficient way for transforming an augmenting link set $\bar{F}$ to another F as in the lemma. For this, in every step of Metric-Completion(c) we have to keep track of the inequalities responsible for cost reductions.

By Lemma 3.3, we may restrict our attention to metric instances. In what follows, we show how to construct a kernel for metric instances for cases $k = 2$ and $k = 3$. (The case $k = 2$ could be easily reduced to $k = 3$, but we treat it separately as it is somewhat simpler.) Section 3.4 then shows how the case of general k can be reduced to either of these cases depending on the parity of k.

3.2 Augmentation from 1 to 2

In this section, we assume that the input graph (V, E) is connected. By Proposition 2.1, we may assume that it is a tree: after contracting all the 2-inseparable sets, there are no two nodes with two edge-disjoint paths between them, implying that there is no cycle in the graph. The minimum cuts are given by the edges of the tree, that is, $\mathcal{D}$ is in one-to-one correspondence with E.

Based on Lemma 3.3, it suffices to solve the problem assuming that the instance $(V, E, E^*, c, w, 2, p)$ is metric. The main observation is that in a metric instance we only need to use links that connect certain special nodes, whose number we can bound by a function of p.

Let us refer to the leaves and nodes of degree at least 3 as *corner nodes*; let $R \subseteq V$ denote their set. Every leaf in the tree (V, E) requires at least one incident edge in F. If the number of leaves is greater than $2p$, we may conclude that the problem is infeasible. (Formally, in this case we may return the following kernel: a single edge as the input graph with an empty link set.) If there are at most $2p$ leaves, then $|R| \leq 4p - 2$, due to the following simple fact.

Proposition 3.4. *The number of nodes of degree at least 3 in a tree is at most the number of leaves minus 2.*

Based on the following theorem, we can obtain a kernel on at most $4p - 2$ nodes by contracting each path of degree-2 nodes to a single edge. The number of links in the kernel will be $O(p^3)$.

Theorem 3.5. *For a metric instance $(V, E, E^*, c, w, 2, p)$, there exists an optimal solution F such that every edge in F is only incident to corner nodes.*

The proof (see full version) analyses an optimal solution with the total number of links minimal, and subject to this, the total length of the paths in the tree between the endpoints of the links minimal. Such an optimal solution may contain no links incident to degree 2 nodes.

3.3 Augmentation from 2 to 3

In this section we assume that the input graph is 2-edge-connected but not 3-edge-connected. Let us call a 2-edge-connected graph $G = (V, E)$ a *cactus*, if every edge belongs to exactly one circuit. This is equivalent to saying that every block (maximal induced 2-node-connected subgraph) is a circuit (possibly of length 2, using two parallel edges). Figure 1 gives an example of a cactus.

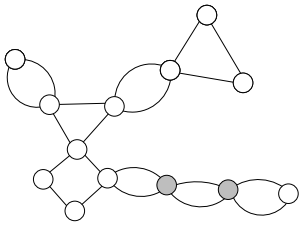


Fig. 1. A cactus graph. The shaded nodes are in the set T.

By Proposition 2.1, we may assume that every 3-inseparable set in G is a singleton, that is, there are no two nodes in the graph connected by 3 edge-disjoint paths.

Proposition 3.6. *Assume that $G = (V, E)$ is a 2-edge-connected graph such that every 3-inseparable set is a singleton. Then G is a cactus.*

In the rest of the section, we assume $G = (V, E)$ is a cactus. The set of minimum cuts $\mathcal{D}$ corresponds to arbitrary pairs of 2 edges on the same circuit.

Again by Lemma 3.3, we may restrict our attention to metric instances. Let us call a circuit of length 2 a *2-circuit* (that is, a set of two parallel edges between two nodes). Let R_1 denote the set of nodes of degree 2, or equivalently, the set of nodes incident to exactly one circuit. Let R_2 denote the set of nodes incident to at least 3 circuits, or at least two circuits not both 2-circuits. Let $R = R_1 \cup R_2$ and let $T = V \setminus R$ denote the set of remaining nodes, that is, the set of nodes that are incident to precisely two circuits, both 2-circuits (see Figure 1). The elements of R will be again called *corner nodes*. We can give the following simple bound:

Proposition 3.7. $|R_2| \leq 4|R_1| - 8$.

Observe that every node in R_1 forms a singleton minimum cut. Hence if $|R_1| > 2p$, we may conclude infeasibility. Otherwise, Proposition 3.7 gives $|R| \leq 10p - 8$.

We prove the analogue of Theorem 3.5: we show that it is sufficient to consider only links incident to R. It follows that we can obtain a kernel on at most $10p-8$ nodes by replacing every path consisting of 2-circuits by a single 2-circuit. The number of links in the kernel will be $O(p^3)$.

3.4 Augmenting Edge-Connectivity for Higher Values

In this section, we assume that the input graph $G = (V, E)$ is already $(k-1)$-connected, where k is the connectivity target. We show that for even or odd k, the problem can be reduced to the $k = 2$ or the $k = 3$ case, respectively.

Assume first k is even. We use the following simple structure theorem, which is based on the observation that if the minimum cut value in a graph is odd, then the family of minimum cuts is cross-free.

Theorem 3.8 ([8, Thm 7.1.2]). *Assume the minimum cut value $k-1$ in the graph $G = (V, E)$ is odd. Then there exists a tree $H = (U, L)$ along with a map $\varphi : V \to U$ such that the min-cuts of G and the edges of H are in one-to-one correspondence: for every edge $e \in L$, the pre-images of the two components of $H - e$ are the sides of the corresponding min-cut, and every minimum cut can be obtained this way.*

For odd k, the following theorem shows that the minimum cuts can be represented by a cactus.

Theorem 3.9 (Dinits, Karzanov, Lomonosov [4], [8, Thm 7.1.8]). *Consider a loopless graph $G = (V, E)$ with minimum cut value $k-1$. Then there exists a cactus $H = (U, L)$ along with a map $\varphi : V \to U$ such that the min-cuts of G and the edges of H are in one-to-one correspondence. That is, for every minimum cut $X \subseteq U$ of H, $\varphi^{-1}(X)$ is a minimum cut in G, and every minimum cut in G can be obtained in this form.*

Observe that if G does not contain k-inseparable pairs (e.g., it was obtained by contracting all the maximal k-inseparable sets), then φ in Theorems 3.8 and 3.9 is one-to-one: $\varphi(x) = \varphi(y)$ would mean that there is no minimum cut separating x and y. Therefore, in this case Theorems 3.8 and 3.9 imply that we can replace the graph with a tree or cactus graph H in a way that the minimum cuts are preserved. Note that the *value* of the minimum cut does change: it becomes 1 (if H is a tree) or 2 (if H is a cactus), but $X \subseteq V$ is a minimum cut in G if and only if it is a minimum cut in H.

Lemma 3.10. *Let $G = (V, E)$ be a $(k-1)$-edge-connected graph containing no k-inseparable pairs. Then in polynomial time, one can construct a graph $H = (V, L)$ on the same node set having exactly the same set of minimum cuts such that*

1. *if k is even, then H is a tree (hence the minimum cuts are of size 1);*
2. *if k is odd, then H is a cactus (hence the minimum cuts are of size 2);*

Now we are ready to show that if G is $(k-1)$-edge-connected, then a kernel containing $O(p)$ nodes, $O(p)$ edges, and $O(p^3)$ links is possible for every k. First,

we contract every maximal k-inseparable set; if multiple links are created between two nodes with the same weight, let us only keep one with minimum cost. By Proposition 2.1, this does not change the problem. Then we can apply Lemma 3.10 to obtain an equivalent problem on graph H having a specific structure. If k is even, then covering the $(k-1)$-cuts of G is equivalent to covering the 1-cuts of the tree H, that is, augmenting the connectivity of G to k is equivalent to augmenting the connectivity of H to 2. Therefore, we can use the algorithm described in Section 3.2 to obtain a kernel. If k is odd, then covering the $(k-1)$-cuts of G is equivalent to covering the 2-cuts of the cactus H, that is, augmenting the connectivity of G to k is equivalent to augmenting the connectivity of H to 3. In this case, Section 3.3 gives a kernel.

3.5 Decreasing the Size of the Cost

We have shown that for arbitrary instance (V, E, E^*, c, w, k, p), if (V, E) is $(k-1)$-edge-connected then there exists a kernel on $O(p)$ nodes and $O(p^3)$ links. However, the costs of the links in this kernel can be arbitrary rational numbers (assuming the input contained rational entries).

We show that the technique of Frank and Tardos [10] is applicable to replace the cost by integers whose size is polynomial in p and the instance remains equivalent to the original one.

Theorem 3.11 ([10]). *Let us be given a rational vector $c = (c_1, \ldots, c_n)$ and an integer N. Then there exists an integral vector $\bar{c} = (\bar{c}_1, \ldots, \bar{c}_n)$ such that $||\bar{c}||_\infty \leq 2^{4n^3} N^{n(n+2)}$ and $sign(c \cdot b) = sign(\bar{c} \cdot b)$, where b is an arbitrary integer vector with $||b||_1 \leq N-1$. Such a vector $\bar{c}$ can be constructed in polynomial time.*

In our setting, $n = O(p^3)$ is the length of the vector. What we need to guarantee is that for c and $\bar{c}$, $c(F) < c(F')$ if and only if $\bar{c}(F) < \bar{c}(F')$ for arbitrary two sets of links F, F' with $|F|, |F'| \leq p$. Hence we need to guarantee the property for vectors b with $||b||_1 \leq 2p$, giving $N = 2p+1$. Therefore the theorem provides a guarantee $||\bar{c}||_\infty \leq 2^{O(p^6)}(2p+1)^{O(p^6)}$, meaning that the entries of $\bar{c}$ can be described by $O(p^6 \log p)$ bits. An optimal solution for the cost vector $\bar{c}$ will be optimal for the original cost c. This completes the proof of Theorem 1.1.

References

1. Agrawal, A., Klein, P., Ravi, R.: When trees collide: An approximation algorithm for the generalized steiner problem on networks. SIAM Journal on Computing 24(3), 440–456 (1995)
2. Cheriyan, J., Végh, L.A.: Approximating minimum-cost k-node connected subgraphs via independence-free graphs. arXiv preprint arXiv:1212.3981 (2012)
3. Cheriyan, J., Vempala, S., Vetta, A.: An approximation algorithm for the minimum-cost k-vertex connected subgraph. SIAM J. Comput. 32(4), 1050–1055 (2003)

4. Dinits, E., Karzanov, A., Lomonosov, M.: On the structure of a family of minimal weighted cuts in graphs. In: Fridman, A. (ed.) Studies in Discrete Mathematics, Nauka, Moscow, pp. 290–306 (1976) (in Russian)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin (2006)
7. Frank, A.: Augmenting graphs to meet edge-connectivity requirements. SIAM J. Discret. Math. 5(1), 25–53 (1992)
8. Frank, A.: Connections in combinatorial optimization. Oxford lecture series in mathematics and its applications, vol. 38. Oxford Univ. Pr. (2011)
9. Frank, A., Jordán, T.: Minimal edge-coverings of pairs of sets. Journal of Combinatorial Theory, Series B 65(1), 73–110 (1995)
10. Frank, A., Tardos, É.: An application of simultaneous Diophantine approximation in combinatorial optimization. Combinatorica 7(1), 49–65 (1987)
11. Goemans, M., Williamson, D.: A general approximation technique for constrained forest problems. SIAM Journal on Computing 24(2), 296–317 (1995)
12. Guo, J., Uhlmann, J.: Kernelization and complexity results for connectivity augmentation problems. Networks 56(2), 131–142 (2010)
13. Hsu, T.: On four-connecting a triconnected graph. Journal of Algorithms 35(2), 202–234 (2000)
14. Jackson, B., Jordán, T.: Independence free graphs and vertex connectivity augmentation. Journal of Combinatorial Theory, Series B 94(1), 31–77 (2005)
15. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21(1), 39–60 (2001)
16. Jordán, T.: On the optimal vertex-connectivity augmentation. Journal of Combinatorial Theory, Series B 63(1), 8–20 (1995)
17. Kortsarz, G., Nutov, Z.: Approximating node connectivity problems via set covers. Algorithmica 37(2), 75–92 (2003)
18. Kortsarz, G., Nutov, Z.: Approximating minimum cost connectivity problems. In: Gonzalez, T. (ed.) Handbook on Approximation Algorithms and Metaheuristics. Chapman & Hall/CRC, London (2007)
19. Lokshtanov, D., Misra, N., Saurabh, S.: Kernelization – preprocessing with a guarantee. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) Fellows Festschrift 2012. LNCS, vol. 7370, pp. 129–161. Springer, Heidelberg (2012)
20. Misra, N., Raman, V., Saurabh, S.: Lower bounds on kernelization. Discrete Optimization 8(1), 110–128 (2011)
21. Nagamochi, H.: An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree. Discrete Applied Mathematics 126(1), 83–113 (2003)
22. Nesetril, J., Milková, E., Nesetrilová, H.: Otakar Boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. Discrete Mathematics 233(1-3), 3–36 (2001)
23. Végh, L.A.: Augmenting undirected node-connectivity by one. SIAM Journal on Discrete Mathematics 25(2), 695–718 (2011)
24. Watanabe, T., Nakamura, A.: Edge-connectivity augmentation problems. J. Comput. Syst. Sci. 35(1), 96–144 (1987)

Graph Reconstruction via Distance Oracles*

Claire Mathieu** and Hang Zhou***

Département d'Informatique UMR CNRS 8548,
École Normale Supérieure, Paris, France
{cmathieu,hangzhou}@di.ens.fr

Abstract. We study the problem of reconstructing a hidden graph given access to a distance oracle. We design randomized algorithms for the following problems: reconstruction of a degree bounded graph with query complexity $\tilde{O}(n^{3/2})$; reconstruction of a degree bounded outerplanar graph with query complexity $\tilde{O}(n)$; and near-optimal approximate reconstruction of a general graph.

1 Introduction

Decentralized networks (such as the Internet or sensor networks) raise algorithmic problems different from static, centrally planned networks. A challenge is the lack of accurate maps for the topology of these networks, due to their dynamical structure and to the lack of centralized control. How can we achieve an accurate picture of the topology with minimal overhead? This problem has recently received attention (see e.g., [4,8,10,12]).

For Internet networks, the topology can be investigated at the router and autonomous system (AS) level, where the set of routers (ASs) and their physical connections (peering relations) are the vertices and edges of a graph, respectively. Traditionally, inference of routing topology has relied on tools such as traceroute and mtrace to generate path information. However, these tools require cooperation of intermediate nodes or routers to generate messages. Increasingly, routers block traceroute requests due to privacy and security concerns, so inference of topology increasingly relies on delay information rather than on the route itself. At this level of generality, many problems are provably intractable [2], thus suggesting the need to study related but simpler questions. In this paper, for simplicity we assume that we have access to every vertex in the graph, and only the edges are unknown.

The Problem. Consider the shortest path metric $\delta(\cdot,\cdot)$ of a connected, unweighted graph $G = (V, E)$, where $|V| = n$. In our computational model, we are

* Full version available at http://arxiv.org/abs/1304.6588

** CNRS. Research supported in part by the French ANR Blanc program under contract ANR-12-BS02-005 (RDAM project) and by the NSF medium AF grant 0964037.

*** Research supported in part by the French ANR Blanc program under contract ANR-12-BS02-005 (RDAM project).

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 733–744, 2013.

given the vertex set V, and we have access to δ via a *query oracle* $\textsc{Query}(\cdot,\cdot)$ which, upon receiving a query $(u,v) \in V^2$, returns $\delta(u,v)$. The *metric reconstruction problem* is to find the metric δ on V. The efficiency of a reconstruction algorithm is measured by its *query complexity*, i.e., the number of queries to the oracle. (We focus on query complexity, but our algorithms can also easily be implemented in polynomial time and space).

Note that finding δ is equivalent to finding every edge in E, thus this problem is also called the *graph reconstruction problem*.

Related Work. Reyzin and Srivastava [18] showed an $\Omega(n^2)$ lower bound for the graph reconstruction problem on general graphs. We extend their result to get a lower bound for the graph approximate reconstruction problem.

To reconstruct graphs of bounded degree, we apply some algorithmic ideas previously developed for compact routing [21] and ideas for Voronoi cells [15].

A closely related model in network discovery and verification provides queries which, upon receiving a node q, returns the distances from q to all other nodes in the graph [12], instead of the distance between a pair of nodes in our model. The problem of minimizing the number of queries is **NP**-hard and admits an $O(\log n)$-approximation algorithm (see [12]). In another model, a query at a node q returns all edges on all shortest paths from q to any other node [4]. Network tomography also proposes statistical models [6,20].

Our Results. In Section 2, we consider the reconstruction problem on graphs of bounded degree. We provide a randomized algorithm to reconstruct such a graph with query complexity $\tilde{O}(n^{3/2})$. Our algorithm selects a set of nodes (called *centers*) of expected size $\tilde{O}(\sqrt{n})$, so that they separate the graph into $\tilde{O}(\sqrt{n})$ slightly overlapped subgraphs, each of size $O(\sqrt{n})$. We show that the graph reconstruction problem is reduced to reconstructing every subgraph, which can be done in $O(n)$ queries by exhaustive search inside this subgraph.

In Section 3, we consider *outerplanar* graphs of bounded degree. An *outerplanar* graph is a graph which can be embedded in the plane with all vertices on the exterior face. Chartrand and Harary [7] first introduced outerplanar graphs and proved that a graph is outerplanar if and only if it contains no subgraph homeomorphic from K_4 or $K_{2,3}$. Outerplanar graphs have received much attention in the literature because of their simplicity and numerous applications. In this paper, we show how to reconstruct degree bounded outerplanar graphs with expected query complexity $\tilde{O}(n)$. The idea is to find the node x which appears most often among all shortest paths (between every pair of nodes), and then partition the graph into components with respect x. We will show that such partition is β-*balanced* for some constant $\beta < 1$, i.e., each resulting component is at most β fraction of the graph. Such partitioning allows us to reconstruct the graph recursively with $O(\log n)$ levels of recursion. However, it takes too many queries to compute all shortest paths in order to get x. Instead, we consider an approximate version of x by computing a sampling of shortest paths to get the node which is most often visited among all sampling shortest paths. We will show that the node obtained in this way is able to provide a β-balanced partition with

high probability. Our algorithm for outerplanar graphs gives an $O(\Delta \cdot n \log^3 n)$ bound which, for a tree (a special case of an outerplanar graph), is only slightly worse than the optimal algorithm for trees with query complexity $O(\Delta \cdot n \log n)$ (see [14]). On the other hand, the tree model typically restricts queries to pairs of tree leaves, but we allow queries of any pair of vertices, not just leaves.

In Section 4, we consider an approximate version of the metric reconstruction problem for general graphs. The metric $\widehat{\delta}$ is an f-approximation of the metric δ if for every pair of nodes (u, v), $\widehat{\delta}(u, v) \leq \delta(u, v) \leq f \cdot \widehat{\delta}(u, v)$, where f is any sublinear function of n. We give a simple algorithm to compute an f-approximation of the metric with expected query complexity $O(n^2(\log n)/f)$. We show that our algorithm is near-optimal by providing an $\Omega(n^2/f)$ query lower bound.

An open question is whether the $\tilde{O}(n^{3/2})$ bound in Theorem 1 is tight.

Other Models. The problem of reconstructing an unknown graph by queries that reveal partial information has been studied extensively in many different contexts, independently stemming from a number of applications.

In evolutionary biology, the goal is to reconstruct evolutionary trees, thus the hidden graph has a tree structure. One may query a pair of species and get in return the distance between them in the (unknown) tree [22]. See for example [14,16,19]. In this paper, we assume that our graph is not necessarily a tree, but may have an arbitrary connected topology.

Another graph reconstruction problem is motivated by DNA shotgun sequencing and linkage discovery problem of artificial intelligence [5]. In this model we have access to an oracle which receives a subset of vertices and returns the number of edges whose endpoints are both in this subset. This model has been much studied (e.g., [3,9,13,18]) and an optimal algorithm has been found in [17]. Our model is different since there is no counting.

Geometric reconstruction deals with, for example, reconstructing a curve from a sampling of points [1,11] or reconstructing a road network from a given collection of path traces [8]. In contrast, our problem contains no geometry, so results are incomparable.

2 Degree Bounded Graphs

Theorem 1. *Assume that the graph G has bounded degree Δ. Then we have a randomized algorithm for the metric reconstruction problem, with query complexity $O(\Delta^4 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$, which is $\tilde{O}(n^{3/2})$ when Δ is constant.*

Our reconstruction proceeds in two phases.

In the first phase, we follow the notation from Thorup and Zwick [21]: Let $A \subset V$ be a subset of vertices called *centers*. For $v \in V$, let $\delta(A, v) = \min\{\delta(u, v) \mid u \in A\}$ denote the distance from v to the closest node in A. For every $w \in V$, let the *cluster* of w with respect to the set A be defined by $C_w^A = \{v \in V \mid \delta(w, v) < \delta(A, v)\}$. Thus for $w \notin A$, C_w^A is the set of the vertices whose closest neighbor in $A \cup \{w\}$ is w. Algorithm MODIFIED-CENTER(V, s), which is randomized, takes as input the vertex set V and a parameter $s \in [1, n]$, and returns a subset $A \subset V$

of vertices such that all clusters C_w^A (for all $w \in V$) are of size at most $6n/s$. A has expected size at most $2s \log n$, thus the expected number of queries is $O(s \cdot n \cdot \log^2 n \cdot \log\log n)$. This algorithm applies, in a different context, ideas from [21], except that we use sampling to compute an estimate of $|C_w^A|$.

In the second phase, Algorithm Local-Reconstruction(V, A) takes as input the vertex set V and the set A computed by Modified-Center(V, s), and returns the edge set of G. It partitions the graph into slightly overlapped components according to the centers in A, and proceeds by exhaustive search within each component. Inspired by the Voronoi diagram partitioning in [15], we show that these components together cover every edge of the graph. The expected query complexity in this phase is $O(s \log n(n + \Delta^4(n/s)^2))$.

Letting $s = \sqrt{n}$, the expected total number of queries in the two phases is $O(\Delta^4 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$.

We use the notation Query(A, v) to mean Query(a, v) for every $a \in A$, and the notation Query(A, B) to mean Query(a, b) for every $a \in A$ and $b \in B$.

```
Modified-Center(V, s)
 1  A ← ∅, W ← V
 2  T ← K · log n · log log n  (K = O(1) to be defined later)
 3  while W ≠ ∅
 4     do A' ← Random subset of W s.t. every node has prob. s/|W|
 5        Query(A', V)
 6        A ← A ∪ A'
 7        for w ∈ W
 8           do X ← Random multi-subset of V with s · T elements
 9              Query(X, w)
10              Let Ĉ_w^A ← |X ∩ C_w^A| · n/|X|
11        W ← {w ∈ W : Ĉ_w^A ≥ 5n/s}
12  return A
```

```
Local-Reconstruction(V, A)
 1  E ← ∅
 2  for a ∈ A
 3     do B_a ← {v ∈ V | δ(v, a) ≤ 2}
 4        Query(B_a, V)
 5        D_a ← B_a
 6        for b ∈ B_a
 7           do D_a ← D_a ∪ {v ∈ V | δ(b, v) < δ(A, v)}
 8        Query(D_a, D_a)
 9        E ← E ∪ {(d_1, d_2) ∈ D_a × D_a : δ(d_1, d_2) = 1}
10  return E
```

Figure 1 gives an illustration of Algorithm Local-Reconstruction(V, A). Vertices $a_1, \ldots, a_5$ are centers in A and define subsets $D_{a_1}, \ldots, D_{a_5}$ which overlap slightly. We will show in Lemma 3 that the subsets D_a (for all $a \in A$) together

cover every edge in E. Thus the local reconstruction over every D_a (for $a \in A$) is sufficient to reconstruct the graph.

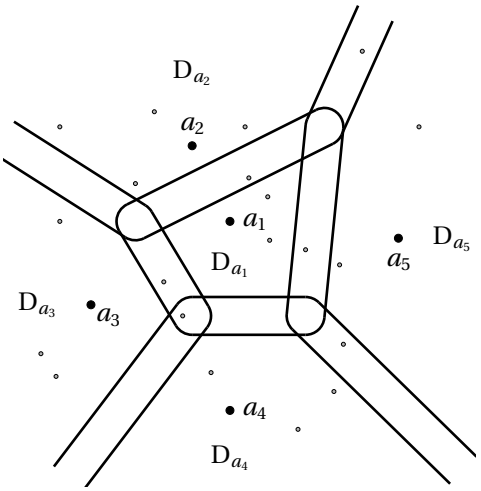


Fig. 1. Partition by centers

Theorem 1 follows from Lemma 2 and 3.

Lemma 2. *With probability at least* $1/(4e)$*, the* Modified-Center(V, s) *algorithm takes* $O(s \cdot n \cdot \log^2 n \cdot \log\log n)$ *queries and returns a set* A *of size at most* $4s \log n$ *such that* $|C_w^A| \leq 6n/s$ *for every* $w \in V$*.*

Remark. *The difference between our algorithm and algorithm* Center(G, s) *in [21] is that,* Center(G, s) *eliminates* $w \in W$ *when* $|C_w^A| < 4n/s$*, by calculating* $|C_w^A|$ *exactly, which needs* n *queries in our model; while our algorithm gives an estimation of* $|C_w^A|$ *using* $O(s \cdot \log n \cdot \log\log n)$ *queries, so that with high probability, it eliminates* $w \in W$ *when* $C_w^A < 4n/s$ *and it does not eliminate* $w \in W$ *when* $C_w^A > 6n/s$*.*

Proof. Fix A and w and let $Y_w = |X \cap C_w^A| = |\{x \in X \mid \delta(x, w) < \delta(x, A)\}|$. The expected value of Y_w is $|C_w^A| \cdot |X|/n$. Since X is random, by standard Chernoff bounds there is a constant K such that, for any node w,

$$\begin{cases} \mathbb{P}\left[Y_w > 5T\right] > 1 - 1/(4n\log n), & \text{if } C_w^A > 6n/s \text{ (and thus } E[Y_w] > 6T) \\ \mathbb{P}\left[Y_w < 5T\right] > 1 - 1/(4n\log n), & \text{if } C_w^A < 4n/s \text{ (and thus } E[Y_w] < 4T). \end{cases}$$

Let $\widehat{C_w^A} = Y_w \cdot n/|X|$, where $|X| = s \cdot T$. When the number of nodes w in estimation is at most $4n \log n$, with probability at least $(1-1/(4n\log n))^{4n\log n} \sim 1/e$, we have:

$$\begin{cases} \widehat{C_w^A} > 5n/s, & \text{if } C_w^A > 6n/s \\ \widehat{C_w^A} < 5n/s, & \text{if } C_w^A < 4n/s \end{cases}, \text{ for every } w \text{ in estimation.} \qquad (1)$$

We assume that n is large enough that this probability is at least $1/(2e)$.

Using the same proof as that of Theorem 3.1 in [21], we can prove that under condition (1), algorithm MODIFIED-CENTER(V, s) executes an expected number of at most $2 \log n$ iterations of the while loop and returns a set A of expected size at most $2s \log n$ such that $|C_w^A| \leq 6n/s$ for every $w \in V$. Thus with probability at least $1/2$, the algorithm executes at most $4 \log n$ iterations of the while loop and the set A is of size at most $4s \log n$. The number of queries is $O(s \cdot n \cdot \log^2 n \cdot \log \log n)$ in this case, since every iteration takes $O(s \cdot n \cdot \log n \cdot \log \log n)$ queries. So the lemma follows. □

Lemma 3. *Under the conditions that $|A| \leq 4s \log n$ and $|C_w^A| \leq 6n/s$ for every $w \in V$, Algorithm* LOCAL-RECONSTRUCTION(V, A) *finds all edges in the graph using $O(s \log n(n + \Delta^4(n/s)^2))$ queries.*

Proof. Let $D_a = B_a \bigcup_{b \in B_a} C_b^A$. We will prove that for every edge (u, v) in E, there is some $a \in A$, such that u and v are both in D_a. Thus the algorithm is correct: it finds all edges in E.

Consider $(u, v) \in E$. Without loss of generality, we assume $\delta(A, u) \leq \delta(A, v)$. Let $a \in A$ be such that $\delta(a, u) = \delta(A, u)$. We will show that u and v are both in D_a. When $\delta(a, u) \leq 1$, u and v are both in $B_a \subseteq D_a$. So we consider only $\delta(a, u) \geq 2$. Take b to be the node, in any of the shortest paths from a to u, such that $\delta(a, b) = 2$. Then $\delta(b, u) = \delta(a, u) - 2$ and $\delta(b, v) \leq \delta(b, u) + \delta(u, v) = \delta(a, u) - 1$ by the triangle inequality. Using $\delta(a, u) = \delta(A, u) \leq \delta(A, v)$, we have $\delta(b, u) < \delta(A, u)$ and $\delta(b, v) < \delta(A, v)$. So u and v are both in C_b^A, which is a subset of D_a since $b \in B_a$.

Because every D_a (for $a \in A$) has size at most $\Delta^2 \cdot 6n/s$, the total query complexity is $O(s \log n(n + \Delta^4(n/s)^2))$. □

3 Degree Bounded Outerplanar Graphs

In this section, we consider the connected graph $G = (V, E)$ to be *outerplanar* [7] and of bounded degree Δ. We show how to reconstruct such a graph with expected query complexity $\tilde{O}(n)$. Generally speaking, we partition the graph into balanced-sized subgraphs and recursively reconstruct these subgraphs.

3.1 Self-contained Subsets, Polygons and Partitions

Before giving details of the algorithm, we first need some new notions.

Definition 4. *The subset $U \subseteq V$ is said to be* self-contained*, if for every $(x, y) \in U \times U$, any shortest path in G between x and y contains nodes only in U.*

For every subset $U \subseteq V$, note $G[U]$ to be the subgraph *induced* by U, i.e., $G[U]$ has exactly the edges over U in the graph. It is easy to see that for every self-contained subset U, $G[U]$ is outerplanar and connected; and that the intersection of several self-contained subsets is again self-contained.

Definition 5. *We say that the k-tuple* $(x_1, \ldots, x_k) \in V^k$ *(where* $k \geq 3$*) forms a* polygon *if* $G[\{x_1, \ldots, x_k\}]$ *has exactly k edges:* $(x_1, x_2), (x_2, x_3), \ldots, (x_k, x_1)$.

Definition 6. *Let U be a self-contained subset of V and let* $U_1, \ldots, U_\eta$ *be subsets of U. We say that* $\{U_1, \ldots, U_\eta\}$ *is a* partition *of U if every* U_i *is self-contained, and for every edge* (x, y) *in* $G[U]$*, there exists some* U_i *(*$1 \leq i \leq \eta$*) such that x and y are both in* U_i*. Let* $\beta < 1$ *be some constant. The partition* $\{U_1, \ldots, U_\eta\}$ *of U is said to be* β-balanced *if every* U_i *is of size at most* $\beta|U|$.

Given any partition of U, the reconstruction problem over U can be reduced to the independent reconstruction over every U_i $(1 \leq i \leq \eta)$.

Let U be a self-contained subset of V. For every vertex $v \in U$, its removal would separate U into n_v $(n_v \geq 1)$ connected components. For every $i \in [1, n_v]$, let $S^*_{v,i}$ be the set of nodes in the i^{th} component and let $S_{v,i} = S^*_{v,i} \cup \{v\}$. We say that $\{S_{v,1}, \ldots, S_{v,n_v}\}$ is the *partition of U by the node v*.

3.2 Balanced-Partition Algorithm

Let us now introduce the main algorithm Balanced-Partition(U), which takes as input a self-contained subset $U \subseteq V$ with $|U| \geq 10$ and returns a β-balanced partition of U, for some constant $\beta \in (0.7, 1)$. The algorithm takes a sampling of 2ω nodes $(a_1, \ldots, a_\omega, b_1, \ldots, b_\omega)$, where $\omega = C \cdot \log|U|$ for some constant $C > 1$, and tries to find a β-balanced partition of U under this sampling. It stops if it finds such a partition, and repeatedly tries another sampling otherwise. Below is the general framework of our algorithm. The details of the algorithmic implementation are given in the full version of the paper, where we give the constants C and β.

1. Take a sampling of 2ω nodes $(a_1, \ldots, a_\omega, b_1, \ldots, b_\omega)$. For every $i \in [1, \omega]$, compute a shortest path between a_i and b_i. Let x be some node with the most occurrences in the ω paths above.
2. Partition U into $S_{x,1}, \ldots, S_{x,n_x}$ by the node x. If all these sets have size at most $\beta|U|$, return $\{S_{x,1}, \ldots, S_{x,n_x}\}$; otherwise let $D = S_{x,k}$ be the largest set among them and let $V_0 = U \backslash S^*_{x,k}$.
3. In the set D, compute the neighbors of x in order: $y_1, \ldots, y_\lambda$, where $\lambda \leq \Delta$. If $\lambda = 1$, go to Step 1.
4. For every $i \in [1, \lambda]$, partition U into $S_{y_i,1}, \ldots, S_{y_i,n_{y_i}}$ by y_i. Let S_{y_i,k_i} be the subset containing x and let $V_i = U \backslash S^*_{y_i,k_i}$ (see Figure 2). If $|V_i| > \beta|U|$, go to Step 1.
5. Let $T = D \cap S_{y_1,k_1} \cap \cdots \cap S_{y_\lambda,k_\lambda}$. Separate T into subsets $T_1, \ldots, T_{\lambda-1}$ as in Figure 2. If every T_i has at most $\beta|U|$ nodes, return $\{T_1, \ldots, T_{\lambda-1}, V_0, \ldots, V_\lambda\}$.
6. Let T_j be the set with more than $\beta|U|$ nodes. Find the unique polygon $(q_1, \ldots, q_l)$ in T_j that goes by nodes x, y_j and y_{j+1}.
7. For every $i \in [1, l]$, partition U into $S_{q_i,1}, \ldots, S_{q_i,n_{q_i}}$ by q_i. Let S_{q_i,m_i} be the subset containing the polygon above and let $W_i = U \backslash S^*_{q_i,m_i}$ (see Figure 3). If $|W_i| > \beta|U|$, go to Step 1.

8. Let $R = S_{q_1,m_1} \cap \cdots \cap S_{q_l,m_l}$. Separate R into subsets $R_1, \ldots, R_l$ as in Figure 3. If some R_i has more than $\beta|U|$ nodes, go to Step 1; else return $\{R_1, \ldots, R_l, W_1, \ldots, W_l\}$.

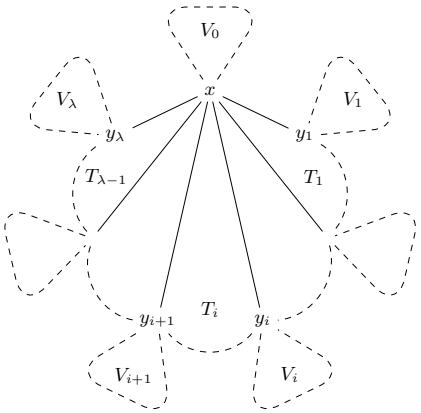


Fig. 2. Partition by neighbors

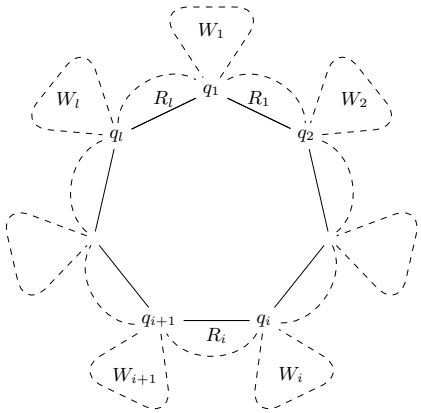


Fig. 3. Partition by polygon

In the full version of the paper, we give formal definitions and algorithms for subproblems: shortest path between two nodes; partition U by a given node; obtain the neighbors of x in order; partitions U with respect to an edge; and find the unique polygon that goes by nodes x, y_j and y_{j+1}. Finally, we give an improved implementation of partitioning U by a polygon (Steps 7 - 8). All these algorithms use $O(\Delta \cdot |U| \log^2 |U|)$ queries. It is easy to see that the algorithm Balanced-Partition(U) always stops with a β-balanced partition of U.

3.3 From Balanced Partitioning to Graph Reconstruction

Let us show how to reconstruct the graph using Balanced-Partition(U) assuming the following proposition, which will be proved in Section 3.4.

Proposition 7. *For any self-contained subset $U \subseteq V$ with $|U| \geq 10$, the randomized algorithm* Balanced-Partition(U) *returns a β-balanced partition of U with query complexity $O(\Delta \cdot |U| \log^2 |U|)$.*

Based on the algorithm Balanced-Partition(U), we reconstruct the graph recursively: we partition the vertex set V into self-contained subsets $V_1, \ldots, V_k$ such that every V_i has size $\leq \beta n$; for every V_i, if $|V_i| < 10$, we reconstruct $G[V_i]$ using at most 9^2 queries; otherwise we partition V_i into self-contained subsets of size at most $\beta|V_i| \leq \beta^2 n$, and continue with these subsets, etc. Thus the number of levels L of the recursion is $O(\log n)$.

Every time Balanced-Partition(U) returns a partition $\{U_1, \ldots, U_k\}$, we always have $|U_1| + \cdots + |U_k| \leq |U| + 2(k-1)$. For every $1 \leq i \leq L$, let $U_{i,1}, \ldots, U_{i,M_i}$

be all sets on the i^{th} level of the recursion. We then have $|U_{i,1}|+\cdots+|U_{i,M_i}| \leq 3n$. Thus the total query complexity on every level is $O(\Delta \cdot n \log^2 n)$ by Proposition 7. So we have the following theorem.

Theorem 8. *Assume that the outerplanar graph G has bounded degree Δ. We have a randomized algorithm for the metric reconstruction problem with query complexity $O(\Delta \cdot n \log^3 n)$, which is $\tilde{O}(n)$ when Δ is constant.*

3.4 Complexity Analysis of the Balanced-Partition Algorithm

Now let us prove Proposition 7. Since the query complexity to try every sampling is $O(\Delta \cdot |U| \log^2 |U|)$, we only need to prove, as in the following proposition, that for every sampling, the algorithm finds a β-balanced partition with high probability. This guarantees that the average number of samplings is a constant, which gives the $O(\Delta \cdot |U| \log^2 |U|)$ query complexity in Proposition 7.

Proposition 9. *In the algorithm* Balanced-Partition(U)*, every sampling of $(a_1, \ldots, a_\omega, b_1, \ldots, b_\omega)$ gives a β-balanced partition with probability at least $2/3$.*

To prove Proposition 9, we need Lemmas 10, 11 and 12, whose proofs are in the full version of the paper.

Lemma 10. *Let $(a_1, \ldots, a_\omega, b_1, \ldots, b_\omega)$ be any sampling during the algorithm* Balanced-Partition(U)*. Let x be the node computed from this sampling in Step 1. We say that a set S is a β-*bad set*, if it is a self-contained subset of U such that $x \notin S$ and $|S| \geq \beta|U|$ for some constant β. Then x does not lead to a β-balanced partition of U only when there exists some β-bad set.*

For any node $u \in U$, define p_u to be be the probability that u is in at least one of the shortest paths between two nodes a and b, where a and b are chosen uniformly and independently at random from U.

Lemma 11. *There exists some constant $\alpha \in (0, 1)$, s.t. in every outerplanar graph of bounded degree, there is a node z with $p_z \geq \alpha$.*

Lemma 12. *Let $\omega = C \cdot \log |U|$ (for some constant C to be chosen in the proof). Take a sample of 2ω nodes uniformly and independently at random from U. Let them be $a_1, \ldots, a_\omega, b_1, \ldots, b_\omega$. For every $v \in U$, let $\widehat{p}_v$ be the percentage of pairs $(a_i, b_i)_{1 \leq i \leq \omega}$ such that v is in some shortest path between a_i and b_i. Let x be some node in U with the largest $\widehat{p}_x$. Then with probability at least $2/3$, we have $p_x > \alpha/2$, where $\alpha > 0$ is the constant in Lemma 11.*

Now we will prove Proposition 9. By Lemma 10, we only need to bound the probability of existence of β-bad set. Let C be the constant chosen in Lemma 12. Let x be the node computed from the sampling $(a_1, \ldots, a_\omega, b_1, \ldots, b_\omega)$ in Step 1 of Algorithm Balanced-Partition(U). Take $\beta = \sqrt{1 - \alpha/2}$, where the constant $\alpha \in (0, 1)$ is provided by Lemma 11. Then $\beta \in (0.7, 1)$. Suppose there exists a β-bad set S. For every $(a, b) \in S \times S$, any shortest path between a and b cannot go by x, since S is self-contained. So $p_x \leq 1 - (|S|/|U|)^2 \leq 1 - \beta^2 = \alpha/2$. By Lemma 12, the probability that $p_x \leq \alpha/2$ is at most $1/3$. So the probability of existence of β-bad set is at most $1/3$. Thus we complete the proof.

4 Approximate Reconstruction on General Graphs

In this section, we study the approximate version of the metric reconstruction problem. We first give an algorithm for the approximate reconstruction, and then show that this algorithm is near-optimal by providing a query lower bound which coincides with its query complexity up to a logarithmic factor.

Definition 13. *Let f be any sublinear function of n. An* f-approximation $\widehat{\delta}$ *of the metric δ is such that, for every $(u,v) \in V^2$, $\widehat{\delta}(u,v) \leq \delta(u,v) \leq f \cdot \widehat{\delta}(u,v)$.*

The following algorithm Approx-Reconstruction(V) receives the vertex set V and samples an expected number of $O(n(\log n)/f)$ nodes. For every sampled node u, it makes all queries related to u and provides an estimate $\widehat{\delta}(v,w)$ for every v within distance $f/2$ from u and every $w \in V \backslash \{v\}$.

Approx-Reconstruction(V)

```
1  while δ̂ is not defined on every pair of nodes
2    do u ← a node chosen from V uniformly at random
3       for every v ∈ V
4          do Query(u, v) and let δ̂(u, v) ← δ(u, v).
5       S_u ← {v : δ(u, v) < f/2}
6       for v ∈ S_u \ {u}
7          do for w ∈ S_u\{v}
8                do δ̂(v, w) ← 1
9             for w ∉ S_u
10               do δ̂(v, w) ← δ(u, w) − δ(u, v)
11 return δ̂
```

Theorem 14. *The randomized algorithm* Approx-Reconstruction(V) *computes an f-approximation $\widehat{\delta}$ of the metric δ using $O(n^2(\log n)/f)$ queries.*

Proof. First we prove that for every (v,w), we have $\widehat{\delta}(v,w) \leq \delta(v,w) \leq f \cdot \widehat{\delta}(v,w)$. There are two cases:

Case 1: $w \in S_u \backslash \{v\}$ (line 7). Then $\widehat{\delta}(v,w) = 1 \leq \delta(v,w) \leq \delta(u,v) + \delta(u,w) < (f/2) + (f/2) = f = f \cdot \widehat{\delta}(v,w)$, because v and w are in S_u.

Case 2: $w \notin S_u$ (line 9). On the one hand, by the triangular inequality, $\delta(v,w) \geq \delta(w,u) - \delta(v,u) = \widehat{\delta}(v,w)$. On the other hand, by the triangular inequality, $\delta(w,v) \leq (\delta(u,w) - \delta(u,v)) + 2\delta(u,v)$. The first tem is $\widehat{\delta}(v,w)$. The second term, by definition of S_u, is at most $f-1$. Since $v \in S_u$ and $w \notin S_u$, we have $\delta(u,w) - \delta(v,w) \geq 1$, so the second term can be bounded by $f - 1 \leq (f-1) \cdot \widehat{\delta}(v,w)$. Adding completes the proof of the upper bound.

Next, we analyze the query complexity of the algorithm. Since G is connected, for every node v there are at least $f/2$ points u such that $v \in S_u$. Let X denote all samples during the algorithm. The number of queries is $n|X|$, and its expectation

is $n \sum_t \Pr[|X| > t]$. Let X_t denote the first t samples chosen. We have:

$$\Pr[|X| > t] = \Pr[\exists v, \forall u \in X_t, v \notin S_u] \leq \begin{cases} 1 & \text{if } t < 2n(\ln n)/f \\ \sum_v \Pr[\forall u \in X_t, v \notin S_u] & \text{otherwise.} \end{cases}$$

By independence, $\Pr[\forall u \in X_t, v \notin S_u] \leq (1 - (f/2)/n)^t \leq e^{-tf/(2n)}$. Thus

$$E[\#\text{queries}] \leq n\frac{2n \ln n}{f} + n^2 \frac{(1/n)}{f/(4n)} = O(n^2(\log n)/f). \qquad \square$$

On the lower bound side, Reyzin and Srivastava proved a tight $\Omega(n^2)$ bound for the exact reconstruction problem, as in the following proposition.

Proposition 15. *[18] Any deterministic or randomized algorithm for the exact graph reconstruction problem requires $\Omega(n^2)$ queries.*

We extend the proof of Proposition 15 to get a lower bound for approximate reconstruction as in Theorem 16, whose proof is in the full version of the paper.

Theorem 16. *Any deterministic or randomized approximation algorithm requires $\Omega(n^2/f)$ queries to compute an f-approximation of the graph metric.*

Acknowledgments. We would like to thank Fabrice Ben Hamouda for his helpful comments.

References

1. Amenta, N., Bern, M., Eppstein, D.: The crust and the beta-skeleton: Combinatorial curve reconstruction. In: Graphical Models and Image Processing, pp. 125–135 (1998)
2. Anandkumar, A., Hassidim, A., Kelner, J.A.: Topology discovery of sparse random graphs with few participants. In: SIGMETRICS, pp. 293–304. ACM (2011)
3. Angluin, D., Chen, J.: Learning a hidden graph using O(log n) queries per edge. In: Shawe-Taylor, J., Singer, Y. (eds.) COLT 2004. LNCS (LNAI), vol. 3120, pp. 210–223. Springer, Heidelberg (2004)
4. Beerliova, Z., Eberhard, F., Erlebach, T., Hall, A., Hoffmann, M., Mihaľák, M., Ram, L.S.: Network discovery and verification. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 127–138. Springer, Heidelberg (2005)
5. Bouvel, M., Grebinski, V., Kucherov, G.: Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 16–27. Springer, Heidelberg (2005)
6. Castro, R., Coates, M., Liang, G., Nowak, R., Yu, B.: Network tomography: recent developments. Statistical Science 19, 499–517 (2004)
7. Chartrand, G., Harary, F.: Planar permutation graphs. Annales de l'institut Henri Poincaré (B) Probabilités et Statistiques 3(4), 433–438 (1967)
8. Chen, D., Guibas, L.J., Hershberger, J., Sun, J.: Road network reconstruction for organizing paths. In: SODA, pp. 1309–1320 (2010)
9. Choi, S.-S., Kim, J.H.: Optimal query complexity bounds for finding graphs. In: STOC, pp. 749–758. ACM (2008)

10. Dall'Asta, L., Alvarez-Hamelin, I., Barrat, A., Vázquez, A., Vespignani, A.: Exploring networks with traceroute-like probes: Theory and simulations. Theoretical Computer Science 355(1), 6–24 (2006)
11. Dey, T.K., Wenger, R.: Reconstructing curves with sharp corners. Comput. Geom. Theory and Appl. 19, 89–99 (2000)
12. Erlebach, T., Hall, A., Hoffmann, M., Mihaľák, M.: Network discovery and verification with distance queries. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 69–80. Springer, Heidelberg (2006)
13. Grebinski, V., Kucherov, G.: Optimal reconstruction of graphs under the additive model. Algorithmica 28(1), 104–124 (2000)
14. Hein, J.J.: An optimal algorithm to reconstruct trees from additive distance data. Bulletin of Mathematical Biology 51(5), 597–603 (1989)
15. Honiden, S., Houle, M.E., Sommer, C.: Balancing graph voronoi diagrams. In: ISVD, pp. 183–191. IEEE (2009)
16. King, V., Zhang, L., Zhou, Y.: On the complexity of distance-based evolutionary tree reconstruction. In: SODA, pp. 444–453. SIAM (2003)
17. Mazzawi, H.: Optimally reconstructing weighted graphs using queries. In: SODA, pp. 608–615. SIAM (2010)
18. Reyzin, L., Srivastava, N.: Learning and verifying graphs using queries with a focus on edge counting. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 285–297. Springer, Heidelberg (2007)
19. Reyzin, L., Srivastava, N.: On the longest path algorithm for reconstructing trees from distance matrices. Information Processing Letters 101(3), 98–100 (2007)
20. Tarissan, F., Latapy, M., Prieur, C.: Efficient measurement of complex networks using link queries. In: INFOCOM Workshops, pp. 254–259. IEEE (2009)
21. Thorup, M., Zwick, U.: Compact routing schemes. In: SPAA, pp. 1–10. ACM (2001)
22. Waterman, M.S., Smith, T.F., Singh, M., Beyer, W.: Additive evolutionary trees. Journal of Theoretical Biology 64(2), 199–213 (1977)

Dual Techniques for Scheduling on a Machine with Varying Speed*

Nicole Megow[1] and José Verschae[2]

[1] Department of Mathematics, Technische Universität Berlin, Germany
nmegow@math.tu-berlin.de
[2] Departamento de Ingeniería Industrial and Centro de Modelamiento Matemático, Universidad de Chile, Santiago, Chile
jverscha@ing.uchile.cl

Abstract. We study scheduling problems on a machine of varying speed. Assuming a known speed function (given through an oracle) we ask for a cost-efficient scheduling solution. Our main result is a PTAS for minimizing the total weighted completion time on a machine of varying speed. This implies also a PTAS for the closely related problem of scheduling to minimize generalized global cost functions. The key to our results is a re-interpretation of the problem within the well-known *two-dimensional Gantt chart*: instead of the standard approach of scheduling in the *time-dimension*, we construct scheduling solutions in the *weight-dimension*.

We also consider a dynamic problem variant in which deciding upon the speed is part of the scheduling problem and we are interested in the tradeoff between scheduling cost and speed-scaling cost, which is typically the energy consumption. We obtain two insightful results: (1) the optimal scheduling order is independent of the energy consumption and (2) the problem can be reduced to the setting where the speed of the machine is fixed, and thus admits a PTAS.

1 Introduction

In several computation and production environments we face scheduling problems in which the speed of resources may vary. We distinguish mainly two types of varying speed scenarios: one, in which the speed is a *given* function of time, and another *dynamic* setting in which deciding upon the processor speed is part of the scheduling problem. The first setting occurs, e.g., in production environments where the speed of a resource may change due to overloading, aging, or in an extreme case it may be completely unavailable due to maintenance or failure. The dynamic setting finds application particularly in modern computer architectures, where speed-scaling is an important tool for power-management. Here we are interested in the tradeoff between the power consumption and the quality-of-service. Both research directions—scheduling on a machine with given speed

* Supported by the German Science Foundation (DFG) under contract ME 3825/1, by FONDECYT grant 3130407, and by Nucleo Milenio Información y Coordinación en Redes ICM/FIC P10-024F.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 745–756, 2013.

fluctuation as well as scheduling including speed-scaling—have been pursued quite extensively, but seemingly separately from each other.

The main focus of our work and the main technical contribution lie in the setting with a given speed function. We present a PTAS for scheduling to minimize the sum of weighted completion times $\sum_j w_j C_j$, which is best possible unless P=NP. In addition, we draw an interesting connection between the given speed and dynamic models which allows us to utilize the results for the given speed setting also for the dynamic problem. Very useful in our arguments is the well-known geometric view of the min-sum scheduling problem in a *two-dimensional Gantt chart*, an interpretation originally introduced in [11]. Crucial to our results is the deviation from the standard view of scheduling in the *time dimension* and switching to scheduling in the *weight dimension*. This dual view allows us to cope with the highly sensitive speed changes in the time dimension which prohibit standard rounding, guessing, and approximation techniques.

Previous Work

Research on scheduling on a machine of given varying speed has been mainly focused on the special case of scheduling with non-availability periods, see e.g. [18]. Despite a history of more than 30 years, only recently the first constant approximation for min $\sum w_j C_j$ was derived in [12]. In fact, their $(4+\varepsilon)$-approximation computes a universal sequence which has the same guarantee for any (unknown) speed function. For the setting with release dates, they give an approximation algorithm with the same guarantee for any given speed function. If the speed is only increasing, there is an efficient PTAS [20], if all release dates are equal. In this case the complexity remains an open question, whereas for general speed functions the problem is strongly NP-hard, even when for each job the weight and processing time are equal [22].

The problem of scheduling on a machine of varying speed is equivalent to scheduling on an ideal machine (of constant speed) but minimizing a more general global cost function $\sum w_j f(C_j)$, where f is a nondecreasing function. In this identification, $f(C)$ denotes the time that the varying speed machine needs to process a work volume of C [14]. Also, the special case of only nondecreasing (nonincreasing) speed functions corresponds to concave (convex) global cost functions. Recently, in [14] tight guarantees for the Smith rule for all convex and all concave functions f were given. They also show that the problem for increasing piecewise linear cost function is strongly NP-hard even with only two slopes, and so is our problem when the speed function takes only two distinct values.

Even more general min-sum cost functions have been studied, where each job may have its individual nondecreasing cost function. A $(2+\varepsilon)$-approximation was recently derived in [10]. For the more complex setting with release dates a randomized $\mathcal{O}(\log\log(n \max_j p_j))$-approximation is known [5]. Clearly, these results translate also to the setting with varying machine speed.

Scheduling with dynamic speed-scaling was initiated in [23] and became a very active research field in the past fifteen years. Most work focuses on scheduling problems where jobs have deadlines by which they must finish. We refer

to [2,15] for an overview. Closer to our setting is the work initiated by Pruhs et al. [19] where they obtain a polynomial algorithm for minimizing the total flow time given an energy budget if all jobs have the same work volume. This work is later continued by many other; see, e. g., [3, 6, 8] and the references therein. Most of this literature is concerned with online algorithms to minimize total (or weighted) flow time plus energy. The minimization of the weighted sum of completion times plus energy has been considered recently in [4, 7]. These works derive constant approximations for general non-preemptive models with unrelated machines and release dates [4] and additional precedence constraints [7]. For our general objective of speed-scaling with an energy budget, [4] also give a randomized $(2+\varepsilon)$-approximation for unrelated machines with release dates.

Our Results

We give several best possible algorithms for problem variants that involve scheduling to minimize the total weighted completion time on a single machine that may vary its speed.

Our main result is an efficient PTAS (Section 3) for scheduling to minimize $\sum w_j C_j$ on a machine of varying speed (given by an oracle). This is best possible since the problem is strongly NP-hard, even when the machine speed takes only two distinct values [14]. We also provide an FPTAS (Section 5.3) for the case that there is a constant number of time intervals with different uniform speeds (and the max ratio of speeds is bounded). Our results generalize recent previous results such as a PTAS on a machine with only *increasing* speeds [20] and FPTASes for only *one* non-availability period [16, 17].

Our results cannot be obtained with standard scheduling techniques which heavily rely on rounding processing requirements or completion times. Such approaches typically fail on machines that may change their speed since the slightest error introduced by rounding might provoke an unbounded increase in the solution cost. Similarly, adding any amount of idle time to the machine might be fatal. Our techniques completely avoid this difficulty by a change of paradigm. To explain our ideas it is helpful to use a 2D-Gantt chart interpretation [11]; see Section 2. As observed before, e.g., in [13], we obtain a *dual* scheduling problem by looking at the y-axis in a 2D-Gantt chart and switching the roles of the processing times and weights. In other words, a dual solution describes a schedule by specifying the remaining weight of the system at the moment a job completes. This simple idea avoids the difficulties on the time-axis and allows to combine old with new techniques for scheduling on the weight-axis.

In case that an algorithm can set the machine at arbitrary speeds, we show in Section 4 that the optimal scheduling sequence is independent of the available energy. This follows by analyzing a convex program that models the optimal energy assignment for a given job permutation. A similar observation was made independently by Vásquez [21] in a game-theoretic setting. We show that computing this universal optimal sequence corresponds to the problem of scheduling with a particular concave global cost function, which can be solved with our PTAS mentioned above, or with a PTAS for non-decreasing speed [20].

Interestingly, this reduction relies again on a problem transformation from time-space to weight-space in the 2D-Gantt chart. For a given scheduling sequence, we give an explicit formula for computing the optimal energy (speed) assignment. Thus, we have a PTAS for speed-scaling and scheduling for a given energy budget. We remark that the complexity of this problem is open.

In many applications, including most modern computer architectures, machines are only capable of using a given number of discrete power (speed) states. We also provide in Section 5 an efficient PTAS for this complex scenario. This algorithm is again based on our techniques relying on dual schedules. Furthermore, we obtain a $(1+\varepsilon)$-approximation of the Pareto frontier for the energy-cost bicriteria problem. On the other hand, we show that this problem is NP-hard even when there are only two speed states. We complement this result by giving an FPTAS for a constant number of available speeds.

We also notice that in the speed-scaling setting, our (F)PTAS results can be utilized to obtain a $(2+\varepsilon)$-approximation for the more general problem of preemptively scheduling jobs with non-trivial release dates on identical parallel machines. Here, we apply our previous results to solve a *fast single machine relaxation* [9] combined with a trick to control the actual job execution times. Then, we keep the energy assignments computed in the relaxation and apply *preemptive list scheduling* on parallel machines respecting release dates.

We finally remark that all our results for the setting with given speed translate directly to a corresponding result for the equivalent problem $1|\,|\sum_j w_j f(C_j)$ (with f non-decreasing).

2 Model, Definitions, and Preliminaries

Problem Definition. We consider two types of scheduling problems. In both cases we are given a set of jobs $J = \{1, \ldots, n\}$ with work volume (processing time at speed 1) $v_j \geq 0$ and weights $w_j \geq 1$. We seek a schedule on a single machine (permutation of jobs) that minimizes the sum of weighted completion times. The speed of the machine may vary—this is where the problems distinguish.

In the problem *scheduling on a machine of given varying speed* we assume that the speed function is given indirectly by an oracle. Given a value v, the oracle returns the first point in time when the machine can finish v units of work. Thus, for a given order of jobs, we can compute the execution time of each job and then the total cost of the solution (assuming that there is no idle time).

In the problem *scheduling with speed-scaling* an algorithm determines not only a schedule for the jobs but will also decide at which speed $s > 0$ the machine will run at any time. Running a machine at certain speed requires a certain amount of power. Power is typically modeled as a monomial (convex) function of speed, $P(s) = s^\alpha$ with a small constant $\alpha > 1$. Given an energy budget E, we ask for the optimal power (and thus speed) distribution and corresponding schedule that minimizes $\sum_j w_j C_j$. More generally, we are interested in quantifying the tradeoff between the scheduling objective $\sum_{j\in J} w_j C_j$ and the total energy consumption, that is, we aim for computing the Pareto curve for the bicriteria minimization problem. We consider two variants of speed-scaling: If the

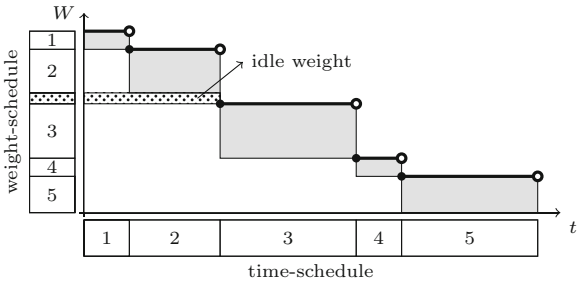


Fig. 1. 2D-Gantt chart. The x-axis shows a schedule, while the y-axis corresponds to $W(t) = \sum_{C_j > t} w_j$ plus the idle weight in the corresponding weight-schedule.

machine can run at an arbitrary power level $p \in \mathbb{R}_+$, we say that we are in the *continuous-speed* setting. On the other hand, if that machine can only choose among a finite set of speeds $\{s_1, \ldots, s_\kappa\}$ we are in an *discrete-speed* environment.

From Time-Space to Weight-Space. For a schedule $\mathcal{S}$, we let $C_j(\mathcal{S})$ denote the completion time of j and we let $W^{\mathcal{S}}(t)$ denote the total weight of jobs completed (strictly) after t. Whenever $\mathcal{S}$ is clear from the context we omit it. It is not hard to see that

$$\sum_{j \in J} w_j C_j(\mathcal{S}) = \int_0^\infty W^{\mathcal{S}}(t) dt. \tag{1}$$

Our main idea is to describe our schedule in terms of the remaining weight function W. That is, instead of determining C_j for each job j, we will implicitly describe the completion time of j by the value of W at the time that j completes. We call this value the *starting weight* of the job j, and denote it by S_j^w. Similarly, we define the *completion weight* of j as $C_j^w := S_j^w + w_j$. This has a natural interpretation in the two axes of the 2D-Gantt chart (see Figure 1): A typical schedule determines completion times for jobs in *time-space* (x-axis), which is highly sensitive when the speed of the machine may vary. We call such a solution a *time-schedule*. Describing a scheduling solution in terms of remaining weight can be seen as scheduling in the *weight-space* (y-axis), yielding a *weight-schedule*.

In weight-space the weights play the role of processing times. All notions that are usually considered in schedules apply in weight-space. For example, we say that a weight-schedule is feasible if there are no two jobs overlapping, and that the machine is idle at weight value w if $w \notin [S_j^w, C_j^w]$ for all j. In this case we say that w is *idle weight*. A weight-schedule immediately defines a non-preemptive time-schedule by ordering the jobs by decreasing completion weights.

Consider a weight-schedule $\mathcal{S}$ with completion weights $C_1^w \geq \ldots \geq C_n^w$, and corresponding completion times $C_1 \leq \ldots \leq C_n$. To simplify notation let $C_0 = C_{n+1}^w = 0$. Then we define the cost of $\mathcal{S}$ as $\sum_{j=1}^n (C_j^w - C_{j+1}^w) C_j$. It is easy to check, even from the 2D-Gantt chart, that this value equals $\sum_{j=1}^n x_j^{\mathcal{S}} C_j^w$, where $x_j^{\mathcal{S}}$ is the execution time of job j (in time-space). Moreover, the last expression is equivalent to Equation (1) if and only if the weight-schedule does

not have any idle weight. In general, the cost of the weight-schedule can only overestimate the cost of the corresponding schedule in time space, given by (1).

On a machine of varying speed, the weight-schedule has a number of technical advantages. For instance, while creating idle *time* can increase the cost arbitrarily, we can create idle *weight* without provoking an unbounded increase in the cost. This gives us flexibility in weight-space and implicitly a way to delay one or more jobs in the time-schedule without increasing the cost. More precisely, we have the following observation that can be easily seen in the 2D-Gantt chart.

Observation 1. *Consider a weight-schedule $\mathcal{S}$ with enough idle weight so that decreasing the completion weight of some job j, while leaving the rest untouched, yields a feasible weight-schedule. If the order of the jobs is changed, then the corresponding time-schedule is also modified. However, since the order of jobs in the time-schedule is reversed, job j gets delayed but the completion* time *of each job $j' \neq j$ is not increased. Thus, this operation does not increase the cost of $\mathcal{S}$.*

3 A PTAS for Scheduling on a Machine with Given Speeds

In what follows we give a PTAS for minimizing $\sum_j w_j C_j$ on a machine with a given speed function. In order to gain structure, we start by applying several modifications to the instance and optimal solution. First we round the weights of the jobs to the next integer power of $1+\varepsilon$, which can only increase the objective function by a factor $1+\varepsilon$. Additionally, we discretize the weight-space in intervals that increase exponentially. That is, we consider intervals $I_u = [(1+\varepsilon)^{u-1}, (1+\varepsilon)^u)$ for $u \in \{1, \ldots, \nu\}$ where $\nu := \left\lceil \log_{1+\varepsilon} \sum_{j \in J} w_j \right\rceil$. We denote the length of each interval I_u as $|I_u| := \varepsilon(1+\varepsilon)^{u-1}$. We will apply two important procedures to modify weight-schedules. They are used to create idle weight so to apply Observation 1, and they only increase the total cost by a factor $1+\mathcal{O}(\varepsilon)$. Similar techniques, applied in time-space, were used by Afrati et al. [1].

Weight Stretch: We multiply by $1+\varepsilon$ the completion weight of each job. This creates an idle weight interval of length εw_j before the starting weight of job j.

Stretch Intervals: We delay the completion weight of each job j with $C_j^w \in I_u$ by $|I_u|$, so that C_j^w belongs to I_{u+1}. Then $|I_{u+1}| - |I_u| = \varepsilon^2(1+\varepsilon)^{u-1} = \varepsilon |I_{u+1}|/(1+\varepsilon)$ units of weight are left idle in I_{u+1} after the transformation, unless there was only one job completely covering I_{u+1}. By moving jobs within I_u, we can assume that this idle weight is consecutive.

3.1 Dynamic Program

We now show our dynamic programming (DP) approach to obtain a PTAS. We first describe a DP table with exponentially many entries and then discuss how to reduce its size. Consider a subset of jobs $S \subseteq J$ and a partial schedule of S in the weight-space. In our dynamic program, S will correspond to the set of jobs at the beginning of the weight-schedule, i. e., if $j \in S$ and $k \in J \setminus S$

then $C_j^w < C_k^w$. A partial weight-schedule $\mathcal{S}$ of jobs in S implies a schedule in time-space with the following interpretation. Note that the makespan of the time-schedule is completely defined by the total work volume $\sum_j v_j$. We impose that the last job of the schedule, which corresponds to the first job in $\mathcal{S}$, finishes at the makespan. This uniquely determines a value of C_j for each $j \in S$, and thus also its execution time $x_j^{\mathcal{S}}$. The total cost of this partial schedule is $\sum_{j \in S} x_j^{\mathcal{S}} C_j^w$ (which has a simple interpretation in the 2D-Gantt chart).

Consider $\mathcal{F}_u := \{S \subseteq J : w(S) \leq (1+\varepsilon)^u\}$. That is, a set $S \in \mathcal{F}_u$ is a potential set to be scheduled in I_u or before. For a given interval I_u and set $S \in \mathcal{F}_u$, we construct a table entry $T(u, S)$ with a $(1 + \mathcal{O}(\varepsilon))$-approximation to the optimal cost of a weight-schedule of S subject to $C_j^w \leq (1+\varepsilon)^u$ for all $j \in S$.

Consider now $S \in \mathcal{F}_u$ and $S' \in \mathcal{F}_{u-1}$ with $S' \subseteq S$. Let $\mathcal{S}$ be a partial schedule of S where the set of jobs with completion weight in I_u is exactly $S \setminus S'$. We define $\mathrm{APX}_u(S', S) = (1+\varepsilon)^u \sum_{j \in S \setminus S'} x_j^{\mathcal{S}}$, which is a $(1+\varepsilon)$-approximation to $\sum_{j \in S \setminus S'} x_j^{\mathcal{S}} C_j^w$, the partial cost associated to $S \setminus S'$. We remark that the values $\sum_{j \in S \setminus S'} x_j^{\mathcal{S}}$ and $\mathrm{APX}_u(S', S)$ do not depend on the whole schedule $\mathcal{S}$, but only on the total work volume of jobs in S'. We can compute $T(u, S)$ with the following formula, $T(u, S) = \min\{T(u-1, S') + \mathrm{APX}_u(S', S) : S' \in \mathcal{F}_{u-1}, S' \subseteq S\}$.

The set $\mathcal{F}_u$ can be of exponential, and thus also this DP table. In the following we show that there is a polynomial size set $\tilde{\mathcal{F}}_u$ that yields $(1+\varepsilon)$-approximate solutions. We remark that the set $\tilde{\mathcal{F}}_u$ will not depend on the speed of the machine. Thus, the same set can be used in the speed-scaling scenario.

3.2 Light Jobs

We structure an instance by classifying jobs by their size in weight-space.

Definition 1. *In a given schedule, a job j is said to be* light *if $w_j \leq \varepsilon |I_u|$, where u is such that $S_j^w \in I_u$. A job that is not light is* heavy.

Given a weight-schedule for heavy jobs, we can greedily find a $(1 + \mathcal{O}(\varepsilon))$-approximate solution for the complete instance. To show this, consider any weight-schedule $\mathcal{S}$. First, remove all light jobs. Then we move jobs within each interval I_u, such that the idle weight inside each interval is consecutive. Clearly, this can only increase the cost of the solution by a $1+\varepsilon$ factor. After, we apply the following preemptive greedy algorithm to assign light jobs, which we call *Algorithm Smith in Weight-Space*: For $u = 1, \ldots, \nu$ and each idle weight $w \in I_u$, process a job j maximizing v_j/w_j among all available jobs with $w_j \leq \varepsilon |I_u|$.

To remove preemptions, we apply the Stretch Interval subroutine[1] twice, creating an idle weight interval in I_u of length at least $2\varepsilon|I_u|/(1+\varepsilon) \geq \varepsilon|I_u|$ (for $\varepsilon \leq 1$). This gives enough space in each interval I_u to completely process the (unique) preempted light job with starting weight in I_u. Then, Observation 1 implies that we can remove preemptions, obtaining a new schedule $\mathcal{S}'$. We now show that the cost of $\mathcal{S}'$ is at most a factor of $1 + \mathcal{O}(\varepsilon)$ larger than the cost of $\mathcal{S}$.

[1] The Stretch Interval procedure also applies to preemptive settings by interpreting each piece of a job as an independent job.

To do so we need a few definitions. For any weight-schedule $\mathcal{S}$, let us define the *remaining volume function* as $V^{\mathcal{S}}(w) := \sum_{j:C_j^w \geq w} v_j$. For a given w, let $I_j(w)$ be equal 1 if the weight-schedule processes j at weight w, and 0 otherwise. Then, $f_j(w) := (1/w_j) \int_w^\infty I_j(w')dw'$ corresponds to the fraction of job j processed after w. With this we define the *fractional remaining volume function*, which is similar to the remaining volume function but treats light jobs as "liquid":

$$V_f^{\mathcal{S}}(w) := \sum_{j:j \text{ is light}} f_j(w) \cdot v_j + \sum_{j:j \text{ is heavy}, C_j^w \geq w} v_j \qquad \text{for all } w \geq 0.$$

We notice that $V_f^{\mathcal{S}}(w) \leq V^{\mathcal{S}}(w)$ for all $w \geq 0$.

Consider now the function $f(v)$ corresponding to the earliest point in time in which the machine can process a work volume of v. Notice that this is the same function used when transforming our problem to $1||\sum_j w_j f(C_j)$. It is easy to see—even from the 2D-Gantt chart—that $\int_0^\infty f(V^{\mathcal{S}}(w))dw$ corresponds to the cost of the weight-schedule $\mathcal{S}$. Also, notice that $f(v)$ is non-decreasing, so that $V^{\mathcal{S}}(w) \leq V^{\mathcal{S}'}(w)$ for all w implies that the cost of $\mathcal{S}$ is at most the cost of $\mathcal{S}'$.

Lemma 1. *The cost of $\mathcal{S}'$ is at most $1 + \mathcal{O}(\varepsilon)$ times larger than the cost of $\mathcal{S}$.*

Proof (Idea). If we assume the position of heavy jobs as given, the schedule $\mathcal{S}_f$ returned by Algorithm Smith in Weight Space minimizes $V_f^{\mathcal{S}_f}(w)$ for any given $w \geq 0$. The result follows by combining this insight plus the ideas above.

Corollary 1. *At a loss of a $1 + \mathcal{O}(\varepsilon)$ factor in the objective function, we can assume the following. For a given interval I_u, consider any pair of jobs j, k whose weights are at most $\varepsilon|I_u|$. If both jobs are processed in I_u or later and $v_k/w_k \leq v_j/w_j$, then $C_j^w \leq C_k^w$.*

3.3 Localization and Compact Search Space

The objective of this section is to compute, for each job $j \in J$, two values r_j^w and d_j^w so that job j is scheduled completely within $[r_j^w, d_j^w)$ in some $(1+\mathcal{O}(\varepsilon))$-approximate weight-schedule. We call r_j^w and d_j^w the *release-weight* and *deadline-weight* of job j, respectively. Crucially, we need that the length of the interval $[r_j^w, d_j^w)$ is not too large, namely that $d_j \in \mathcal{O}(\text{poly}(1/\varepsilon)r_j)$. Such values can be obtained by using Corollary 1 and techniques from [1]; we skip the details.

Lemma 2. *We can compute in poly-time values r_j^w and d_j^w for each $j \in J$ such that: (i) there exists a $(1 + \mathcal{O}(\varepsilon))$-approximate weight-schedule that processes each job j within $[r_j^w, d_j^w)$, (ii) there exists a constant $s \in \mathcal{O}(\log(1/\varepsilon)/\varepsilon)$ such that $d_j^w \leq r_j^w \cdot (1+\varepsilon)^s$, (iii) r_j^w and d_j^w are integer powers of $(1+\varepsilon)$, and (iv) the values r_j^w an d_j^w are independent of the speed of the machine.*

Now we are ready to express set $\tilde{\mathcal{F}}_u$. Instead of describing a set $S \in \tilde{\mathcal{F}}_u$, we describe $V = J \setminus S$, that is, the jobs with completion weights in I_{u+1} or later. Clearly, Lemma 2 implies that we just need to decide about jobs with release weights $r_j^w = (1+\varepsilon)^v$ with $v \in \{u+1-s, \ldots, u-1\}$. Enumerating over (basically) all possibilities for each $v \in \{u+1-s, \ldots, u-1\}$, we obtain the following.

Lemma 3. *For each interval I_u, we can construct in poly-time a set $\tilde{\mathcal{F}}_u$ that satisfies the following: (i) there exists a $(1+\mathcal{O}(\varepsilon))$-approximate weight-schedule in which the set of jobs with completion weight at most $(1+\varepsilon)^u$ belongs to $\tilde{\mathcal{F}}_u$ for each interval I_u, (ii) the set $\tilde{\mathcal{F}}_u$ has cardinality at most $2^{\mathcal{O}(\log^2(1/\varepsilon)/\varepsilon^3)}$, and (iii) the set $\tilde{\mathcal{F}}_u$ is completely independent of the speed of the machine.*

With the discussion at the beginning of this section we obtain a PTAS, which is a best possible approximation since the problem is strongly NP-hard [14].

Theorem 1. *There exists an efficient PTAS for minimizing the weighted sum of completion times on a machine with given varying speed.*

4 Speed-Scaling for Continuous Speeds

When assuming a continuous spectrum of speeds, each job will be executed at a uniform speed because of the convexity of the power function [23]. Let s_j be the speed at which job j is running. Then j's power consumption is $p_j = s_j^\alpha$, and its execution time is $x_j = v_j/s_j = v_j/p_j^{1/\alpha}$. The energy that is required for processing j is $E_j = p_j \cdot x_j = p_j \cdot \frac{v_j}{s_j} = s_j^{\alpha-1} \cdot v_j = v_j^\alpha / x_j^{\alpha-1}$.

Let π be a sequence of jobs in a schedule, where $\pi(j)$ is the index of the j-th job in the sequence for each $i \in \{1,\dots,n\}$. Computing the optimal energy assignment for all jobs, given a fixed sequence π and using a total amount of energy E, can be done with a convex program. We rewrite the objective function as $\sum_{j=1}^n w_j C_j = \sum_{j=1}^n w_{\pi(j)} \sum_{k=1}^j x_{\pi(k)} = \sum_{j=1}^n x_{\pi(j)} \sum_{k=j}^n w_{\pi(k)}$ and define $W^\pi_{\pi(j)} = \sum_{k=j}^n w_{\pi(k)}$. Note that $x_j = \left(v_j^\alpha / E_j\right)^{1/(\alpha-1)}$, and that W_j^π is the total remaining weight just before j is completed in any schedule concordant with π.

$$\min \left\{ \sum_{j=1}^n W_j^\pi \cdot \left(\frac{v_j^\alpha}{E_j}\right)^{1/(\alpha-1)} : \quad \sum_{j=1}^n E_j \le E, \text{ and } E_j \ge 0 \quad \forall j \in \{1,\dots,n\} \right\}.$$

This program has linear constraints and a convex objective function. The next theorem easily follows by the well-known KKT conditions.

Theorem 2. *For a given job sequence π, a power function $P(s) = s^\alpha$ and an energy budget E, the optimal energy assignment in an optimal schedule for minimizing $\sum_j w_j C_j$ subject to $C_{\pi(1)} < \ldots < C_{\pi(n)}$ is determined by*

$$E_j = v_j \cdot \left(W_j^\pi\right)^{(\alpha-1)/\alpha} \cdot \frac{E}{\gamma_\pi}, \text{ where } \gamma_\pi = \sum_{j=1}^n v_j \cdot \left(W_j^\pi\right)^{(\alpha-1)/\alpha}.$$

Interestingly, the optimal job sequence is independent of the energy distribution, and even stronger, it is independent of the overall energy budget. In other words, one scheduling sequence is universally optimal for all energy budgets. Furthermore, this sequence is obtained by solving *in weight-space* a (standard) scheduling problem with a cost function that depends on the power function.

Theorem 3. *Given a power function $P(s) = s^\alpha$, there is a universal sequence that minimizes $\sum_j w_j C_j$ for any energy budget. The sequence is given by reversing an optimal solution of the scheduling problem $1|\,|\sum w_j C_j^{(\alpha-1)/\alpha}$ on a single machine of unit speed.*

Thus, the scheduling part of the speed-scaling scheduling problem reduces to a problem which can be solved by our PTAS from Sect. 3. Since the cost function $f(x) = x^{(\alpha-1)/\alpha}$ is concave for $\alpha > 1$, the specialized PTAS in [20] also solves it. Combining Theorems 2 and 3 gives the main result.

Theorem 4. *There is a PTAS for the continuous speed-scaling and scheduling problem with a given energy budget E.*

5 Speed-Scaling for Discrete Speeds

In this section we consider a more realistic setting, where the machine can choose from a set of κ different speeds available $s_1 > \ldots > s_\kappa \geq 1$.

5.1 A PTAS for Discrete Speeds

Let the power function $P(s)$ be an arbitrary computable function. To derive our algorithm, we adopt the PTAS for scheduling on a machine with given varying speed (Sect. 3) and incorporate the allocation of energy.

We adopt the same definitions of weight intervals I_u and sets $\mathcal{F}_u$ as in Sect. 3. For a subset of jobs $S \in \mathcal{F}_u$ and a value $z \geq 0$, let $E[u, S, z]$ be the minimum total energy necessary for scheduling S such that $C_j^w \leq (1+\varepsilon)^u$ for each $j \in S$, and the scheduling cost is at most z, i.e., $\sum_{j\in S} x_j \cdot C_j^w \leq z$ where x_j is the execution time under some feasible speed assignment. Recall that the speed assignment determines the energy. The recursive definition of a state is as follows:

$$E(u, S, z) = \min\{E(u-1, S', z') + \mathrm{APX}_u(S \setminus S', z - z') : S' \in \mathcal{F}_{u-1}, S' \subseteq S\}.$$

Here $\mathrm{APX}_u(S \setminus S', z - z')$ is the minimum energy necessary for scheduling all jobs $j \in S \setminus S'$ with $C_j^w \in I_u$, such that their partial (rounded) cost $\sum_{j\in S\setminus S'} x_j(1+\varepsilon)^u$ is at most $z - z'$.

Lemma 4. *The value $APX_u(S \setminus S', z - z')$ can be computed with an LP.*

We let the DP fill the table for $u \in \{0, \ldots, \nu\}$ with $\nu = \lceil \log \sum_{j\in J} w_j \rceil$ and $z \in [1, z_{\mathrm{UB}}]$ for some upper bound such as $z_{\mathrm{UB}} = \sum_{j\in J} w_j \sum_{k=1}^{j} v_j/s_\kappa$. Then among all end states $[\nu, J, \cdot\,]$ with value at most the energy budget E we choose the one with minimum cost z. Then we obtain the corresponding $(1+\varepsilon)$-approximate solution for energy E by backtracking.

This DP has an exponential number of entries. However, we can apply results from Section 3 and standard rounding techniques to reduce the running time.

Theorem 5. *There is an efficient PTAS for minimizing the total scheduling cost for speed-scaling with a given energy budget.*

5.2 Speed-Scaling for Discrete Speeds Is NP-Hard

We complement Theorem 5 by showing that our problem is NP-hard. Our reduction is based on the NP-hard problem $1|d_j = d|\sum w_j T_j$ [24].

Theorem 6. *The problem of minimizing $\sum_j w_j C_j$ on a single machine for discrete speeds is NP-hard, even if the number of available power levels is 2.*

5.3 An FPTAS for a Constant Number of Speed States

Let $s_1 > \ldots > s_\kappa \geq 1$. A simple interchange argument shows that an optimal solution chooses the speed non-increasing over time. We construct a schedule in weight-space. There are at most κ jobs that run at more than one speed; call them *split jobs*. We guess the split jobs together with their completion weight up to a factor $1 + \varepsilon$. This partitions the weight space into κ subintervals I_i, which we have to fill with the remaining jobs *non-preemptively*. By construction all jobs in one subinterval run at the same uniform speed. The high-level idea now is to use a DP for partitioning the remaining jobs and keeping control on the power consumption and the total cost. One critical point is that we do not know the execution time x_j for split jobs and we cannot guess them: this would cause a running time dependency on the max-speed ratio. However, for each possible objective value for the split jobs, we can compute the minimum energy with an LP similar to the one in the PTAS in Section 5.1.

The main challenge is to reduce the exponential number DP states to a polynomial size. The intuition behind our algorithm is to remove the states with the same (rounded) objective value and nearly the same total work (differing by at most $\varepsilon|I_i|/n$) assigned to an interval I_i. Among them, we want to store those with smallest amount of work in an interval I_i, in order to make sure that enough space remains for further jobs. To show that this approach is feasible we show bounds on the change in the total cost. This yields the next theorem. The theorem after follows by applying these techniques in time-space.

Theorem 7. *There is an FPTAS for speed-scaling with a given energy budget for $\min \sum w_j C_j$ on a single machine with constantly many discrete speeds.*

Theorem 8. *There exists an FPTAS for non-preemptive[2] scheduling to minimize $\sum w_j C_j$ on a single machine with a constant number of intervals of different, but uniform speed. For the resumable[1] setting, there is an FPTAS in the same setting when the maximum ratio of speeds is bounded.*

References

1. Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation schemes for minimizing average weighted completion time with release dates. In: Proc. of FOCS, pp. 32–43 (1999)
2. Albers, S.: Energy-efficient algorithms. Commun. ACM 53(5), 86–96 (2010)
3. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Trans. Algorithms 3 (2007)

[2] Resumable jobs may run during a speed-0 interval; non-preemptive jobs must not.

4. Angel, E., Bampis, E., Kacem, F.: Energy aware scheduling for unrelated parallel machines. In: Proc. of GreenCom, pp. 533–540 (2012)
5. Bansal, N., Pruhs, K.: The geometry of scheduling. In: Proc. of FOCS, pp. 407–414 (2010)
6. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. SIAM J. Comput. 39(4), 1294–1308 (2009)
7. Carrasco, R.A., Iyengar, G., Stein, C.: Energy aware scheduling for weighted completion time and weighted tardiness. arXiv:1110.0685 (2011)
8. Chan, S.-H., Lam, T.-W., Lee, L.-K.: Non-clairvoyant speed scaling for weighted flow time. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 23–35. Springer, Heidelberg (2010)
9. Chekuri, C., Motwani, R., Natarajan, B., Stein, C.: Approximation techniques for average completion time scheduling. SIAM J. Comput. 31(1), 146–166 (2001)
10. Cheung, M., Shmoys, D.B.: A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In: Goldberg, L.A., Jansen, K., Ravi, R., Rolim, J.D.P. (eds.) APPROX/RANDOM 2011. LNCS, vol. 6845, pp. 135–146. Springer, Heidelberg (2011)
11. Eastman, W.L., Even, S., Isaacs, I.M.: Bounds for the optimal scheduling of n jobs on m processors. Management Sci. 11(2), 268–279 (1964)
12. Epstein, L., Levin, A., Marchetti-Spaccamela, A., Megow, N., Mestre, J., Skutella, M., Stougie, L.: Universal sequencing on a single machine. SIAM J. Comp. 41(3), 565–586 (2012)
13. Goemans, M.X., Williamson, D.P.: Two-dimensional Gantt charts and a scheduling algorithm of Lawler. SIAM J. Disc. Math. 13, 281–294 (2000)
14. Höhn, W., Jacobs, T.: On the performance of Smith's rule in single-machine scheduling with nonlinear cost. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 482–493. Springer, Heidelberg (2012)
15. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News 36(2), 63–76 (2005)
16. Kacem, I., Mahjoub, A.: Fully polynomial time approximation scheme for the weighted flow-time minimization on a single machine with a fixed non-availability interval. Computers & Industrial Engineering 56(4), 1708–1712 (2009)
17. Kellerer, H., Strusevich, V.: Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. Algorithmica 57(4), 769–795 (2010)
18. Lee, C.-Y.: Machine scheduling with availability constraints. In: Leung, J.-T. (ed.) Handbook of Scheduling. CRC Press (2004)
19. Pruhs, K., Uthaisombut, P., Woeginger, G.J.: Getting the best response for your erg. ACM Transactions on Algorithms 4(3) (2008)
20. Stiller, S., Wiese, A.: Increasing speed scheduling and flow scheduling. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part II. LNCS, vol. 6507, pp. 279–290. Springer, Heidelberg (2010)
21. Vásquez, O.C.: Energy in computing systems with speed scaling: optimization and mechanisms design. arXiv:1212.6375 (2012)
22. Wang, G., Sun, H., Chu, C.: Preemptive scheduling with availability constraints to minimize total weighted completion times. Ann. Oper. Res. 133, 183–192 (2005)
23. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. of FOCS, pp. 374–382 (1995)
24. Yuan, J.: The NP-hardness of the single machine common due date weighted tardiness problem. Systems Science and Mathematical Sciences 5(4), 328–333 (1992)

Improved Space Bounds for Strongly Competitive Randomized Paging Algorithms*

Gabriel Moruz and Andrei Negoescu

Goethe University Frankfurt am Main. Robert-Mayer-Str. 11-15,
60325 Frankfurt am Main, Germany
{gabi,negoescu}@cs.uni-frankfurt.de

Abstract. Paging is a prominent problem in the field of online algorithms. While in the deterministic setting there exist simple and efficient strongly competitive algorithms, in the randomized setting a tradeoff between competitiveness and memory is still not settled. In this paper we address the conjecture in [2], that there exist strongly competitive randomized paging algorithms using $o(k)$ bookmarks, i.e. pages not in cache that the algorithm keeps track of. We prove tighter bounds for EQUITABLE2 [2], showing that it requires less than k bookmarks, more precisely $\approx 0.62k$. We then give a lower bound for EQUITABLE2 showing that it cannot both be strongly competitive and use $o(k)$ bookmarks. Our main result proves the conjecture that there exist strongly competitive paging algorithms using $o(k)$ bookmarks. We propose an algorithm, denoted PARTITION2, which is a variant of the PARTITION algorithm in [3]. While PARTITION is unbounded in its space requirements, PARTITION2 uses $\Theta(k/\log k)$ bookmarks.

1 Introduction

The paging problem is defined as follows. We have a two-level memory hierarchy consisting of a fast cache which can accommodate k pages, and a slow memory of infinite size. The input consists of requests to pages which are processed sequentially as follows. If the currently requested page is not in cache, a *cache miss* occurs and the requested page must be brought into cache. If the cache is full, a page must be evicted to accommodate the new one. The cost is given by the number of misses incurred.

Online algorithms in general and paging algorithms in particular are typically analyzed in the framework of *competitive analysis*[4,5]. An algorithm A is said to have a *competitive ratio* of c (or c-competitive) if its cost satisfies for any input $cost(A) \leq c \cdot cost(OPT) + b$, where b is a constant and $cost(OPT)$ is the cost of an optimal offline algorithm, i.e. an algorithm which is presented with the input in advance and processes it optimally; for randomized algorithms, $cost(A)$

* Partially supported by the DFG grants ME 3250/1-3 and MO 2057/1-1, and by MADALGO (Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation). A full version of the paper is available as technical report [1].

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 757–768, 2013.

is the expected cost of A. An algorithm achieving an optimal competitive ratio is *strongly competitive*. For paging, an optimal offline algorithm (OPT) evicts the page whose next request occurs the furthest in the future [6]. For comprehensive surveys on online algorithms in general and paging algorithms in particular, we refer the interested reader to [7,8].

Competitive ratio has often been criticized for its pessimistic quality guarantees. Especially in the deterministic setting, the empirically measured performance for practical algorithms is far below the theoretical guarantee of k provided by competitive analysis [9]. This gap is significantly smaller for randomized algorithms, since the best possible competitive ratio is H_k. Although using only the quality guarantees provided by competitive analysis is a naive way to distinguish good paging algorithms from bad ones, we have shown in [10] that ideas from competitive analysis for randomized algorithms can be successfully employed to design algorithms with good performance on real-world inputs. That is because an optimal randomized algorithm can be viewed as a collection of reasonable deterministic algorithms, and the algorithm designer can simply look for suitable algorithms in this collection.

Randomized paging algorithms have been well studied over the past two decades. In [11] a lower bound of H_k on the competitive ratio of randomized paging algorithms has been given[1]. Also in [11], a simple $(2H_k - 1)$-competitive algorithm, denoted MARK, has been proposed. In [12] it was shown that no randomized marking algorithm can achieve a competitive ratio better than $(2-\varepsilon)H_k$ for any $\varepsilon > 0$. The first strongly competitive paging algorithm, PARTITION, was proposed in [3]. While it is strongly competitive, its time and space usage are in the worst case proportional to the input size independent of the cache size, which is hopelessly high. More recent research focused on improving these bounds, especially the space requirements. Apart from the k pages in cache, a paging algorithm may store information about pages not in cache. In the literature, these "extra" pages are denoted *bookmarks*. An H_k-competitive algorithm, denoted EQUITABLE, using $O(k^2 \log k)$ bookmarks was proposed in [13]. A better version of EQUITABLE, denoted EQUITABLE2, improved this bound to $2k$ bookmarks [14]. This solved the open question in [8] that there exist H_k-competitive paging algorithms using $O(k)$ space. In [15] we proposed an algorithm, ONLINEMIN, which further improved EQUITABLE2 by reducing its runtime for processing a page from $O(k^2)$ to $O(\log k / \log \log k)$ while maintaining its space usage. Note that MARK and most deterministic algorithms use no bookmarks.

A distinct line of research for randomized paging algorithms considers fixed cache sizes ($k = 2$ and $k = 3$ to our best knowledge) to obtain tighter bounds than for general k. In [16], for $k = 2$, a $\frac{3}{2}$-competitive algorithm using only one bookmark was proposed. Still for $k = 2$, for randomized algorithms using no bookmarks lower and upper bounds on the competitive ratio of $\frac{37}{24} \approx 1.5416$ and ≈ 1.6514 respectively were given in [12,16]. In [14], strongly competitive randomized paging algorithms were proposed for $k = 2$ and $k = 3$, using 1 and 2 bookmarks respectively.

[1] $H_k = \sum_{i=1}^{k} 1/i$ is the kth harmonic number.

Our Contributions. This work focuses on the number of bookmarks needed by randomized algorithms to achieve the optimal competitive ratio of H_k. The best previously known result is $2k$ [14]. In [2] it was conjectured that there exist algorithms that use $o(k)$ bookmarks and are H_k-competitive. We first give a tighter analysis for EQUITABLE2 improving the amount of bookmarks from $2k$ to $\approx 0.62k$, which is the first solution using less than k bookmarks. We give a negative result showing that EQUITABLE2 cannot be H_k-competitive and use $o(k)$ bookmarks. Nonetheless, we show that it can trade competitiveness for space: if it is allowed to be (H_k+t)-competitive, it requires $k/(1+t)$ bookmarks.

We propose PARTITION2 which is a modification of the PARTITION algorithm. PARTITION2 improves the bookmark requirements of PARTITION from proportional to input size to $\Theta(k/\log k)$ and thus proves the $o(k)$ conjecture. For our analysis we provide a constructive equivalent between the two representations of the *offset functions* in [17] and [3]. Since offset functions are the key ingredient for optimal competitive paging algorithms, this may be of independent interest.

2 Preliminaries

Offset Functions and Layer Representation. For the paging problem it is possible to track online the exact minimal cost using *offset functions*. For a fixed input sequence σ and an arbitrary cache configuration C (i.e., a set of k pages), the *offset function* ω assigns to C the difference between the minimal cost of processing σ ending in configuration C and the minimal cost of processing σ. A configuration is called *valid* iff $\omega(C) = 0$. In [17] it was shown that the class of valid configurations $\mathcal{V}$ determines the value of ω on any configuration C by $\omega(C) = \min_{X\in\mathcal{V}}\{|C \setminus X|\}$. We can assume that OPT is always in a valid configuration. More precisely, if p is requested and there exists a valid configuration containing p, then the cost of OPT is 0; otherwise OPT pays 1 to process p.

In [17] it was shown for the paging problem that the offset function can be represented as a partitioning of the pageset in $k+1$ disjoint sets $L = (L_0|L_1|\ldots|L_k)$, denoted layers. An update rule for the layers when processing a page was also provided. Initially, the first k pairwise distinct requested pages are stored in layers $L_1, \ldots, L_k$, one page per layer, and L_0 contains the remaining pages. Upon processing page p, let L^p be the partitioning after processing p; we have[2]:

- $L^p = (L_0 \setminus \{p\}|L_1|\ldots|L_{k-2}|L_{k-1} \cup L_k|\{p\})$, if $p \in L_0$
- $L^p = (L_0|\ldots|L_{i-2}|L_{i-1} \cup L_i \setminus \{p\}|L_{i+1}|\ldots|L_k|\{p\})$, if $p \in L_i$, $i > 0$

This layer representation can keep track of all valid configurations. More specifically, a set C of k pages is valid iff $|C \cap L_i| \leq i$ holds for all $0 \leq i \leq k$ [17]. For a given L, denote by *support* $S(L) = L_1 \cup \ldots \cup L_k$. Also, a layer containing a single page is a *singleton*. Let r be the smallest index such that $L_r, \ldots, L_k$ are singletons. The pages in $L_r, \ldots, L_k$ are denoted *revealed*, the pages in support

[2] We use the layer representation introduced in [15], which is equivalent to the ones in [13,17].

which are not revealed are *unrevealed*, and the pages in L_0 are denoted *Opt-miss*. OPT faults on a request to p iff $p \in L_0$ and all revealed pages are (independent of the current request) in OPT's cache. If L has only revealed pages it is denoted a *cone* and we know the content of OPT's cache. We define the *signature* $\chi(L)$ as a k-dimensional vector $\chi = (x_1, \ldots, x_k)$, with $x_i = |L_i| - 1$ for each $i = 1, \ldots, k$.

Selection Process. In [15] we defined a priority-based selection process on L which is guaranteed to construct any valid configuration. Assuming that support pages have pairwise distinct priorities, we build a hierarchy of sets $C_0, \ldots, C_k$:

- $C_0 = \emptyset$
- C_i has the i pages in $C_{i-1} \cup L_i$ having the highest priorities, for all $i > 0$.

Note that, by definition, when constructing C_i there are $i + x_i$ candidates and i slots. Also, if L_i is singleton we have $x_i = 0$ and $C_i = C_{i-1} \cup L_i$; for singleton layers and only for singleton layers, all elements in both C_{i-1} and L_i make it to C_i and we say that no competition occurs. The outcome C_k contains k pages and is always a valid configuration.

Equitable, OnlineMin, and Forgiveness. The cache content of the Equitable algorithms [13,14] is defined by a probability distribution over the set of valid configurations. This distribution is achieved by OnlineMin using the previously introduced priority-based selection process, when priorities are assigned to support pages such that each permutation of the ranks of these pages is equally likely [15]. The cache content of OnlineMin is at all times the outcome C_k of the selection process. Nonetheless, the resulting probability distribution on cache configurations is the same as for Equitable [15], and in the rest of the paper we refer to this distribution and the associated algorithm as Equitable.

Note that the support size increases only when pages in L_0 are requested. As the number of Opt-miss requests may be very large, the support size and together with it the space usage of algorithms, such as Equitable, using it to decide their cache content may also be arbitrarily large. To circumvent this problem, the *forgiveness* mechanism is used. Intuitively, if the support size exceeds a given threshold, then the adversary did not play optimally and we can afford to use an approximation of the offset function with a layer representation bounded in size.

3 Better Bounds for Equitable2

There are two Equitable algorithms, Equitable [13] and Equitable2 [14][3]. For a fixed offset function, they have the same distribution as previously introduced. The difference between them is given by *forgiveness* mechanisms, which are used to approximate the current offset functions. In this section we focus on the Equitable2 algorithm using the forgiveness mechanism described in [14]

[3] In [14] Equitable2 is denoted K_Equitable. In this paper we use its original name.

which works as follows. Whenever the support size reaches $3k$ and an Opt-miss page is requested, the requested page is artificially inserted in L_1 and processed as a L_1 page. All pages in L_1 move to L_0 and the support size never exceeds $3k$. We give a tighter analysis and show that using the same forgiveness the algorithm uses less than k bookmarks and prove that $o(k)$ bookmarks is not possible. Finally, we show that it can trade competitiveness for space: if the algorithm uses $k/(1+t)$ bookmarks, then it is $(H_k + t)$-competitive, for $t \geq 0$.

To accommodate the selection process for OnlineMIN, all pages in support have pairwise distinct priorities, such that each priority ordering of the support pages is equally likely. We say that some page p has *rank* i in a set if its priority is the i'th largest among the elements in the given set.

In [13] an elegant potential function, based only on the current offset function, was introduced. Given the layer representation L, the potential $\Phi(L)$ is defined to be the cost of a so-called *lazy attack sequence*, that is, a sequence of consecutive requests to unrevealed pages until reaching a cone. The potential Φ is well defined because in the case of the Equitable distribution, all lazy attack sequences have the same overall cost for a given offset function [13].

Initially, we are in a cone and $\Phi = 0$. Upon a request to a page p in support, having cache miss probability $pb(p)$, by definition we have that $\Delta\Phi = -pb(p)$. On lazy requests OPT does not fault and thus $\Delta cost + \Delta\Phi = \Delta cost_{OPT} = 0$. Upon a request from L_0 both Equitable and OPT have cost 1 and it was shown that $\Delta\Phi \leq H_k - 1$ [13,14]. We thus have:

$$\Delta cost + \Delta\Phi \leq H_k \cdot \Delta cost_{OPT}.$$

If L is a cone, it is easy to verify that $\Delta\Phi = H_k - 1$ for a request in L_0. If the support size exceeds k, the difference in potential is smaller, i.e. $\Delta\Phi < H_k - 1$. This means that the algorithm pays less than its allowed cost and thus it can make *savings*, which can be tracked by a second potential function and pay for the forgiveness step when the support is large enough. While Φ is very convenient to use for requests in support, for arbitrary offset functions there is no known closed form for its exact actual value or for its exact change upon a request in L_0.

3.1 Approximation of Φ

The key ingredient to our analysis is to get a bound for $\Delta\Phi$ that is as tight as possible on requests in L_0. A tighter bound for this value implies larger savings, which in turn means that these savings can pay earlier (i.e. for a smaller support size) for a forgiveness step, which in the end means fewer bookmarks. We therefore analyze $\Delta\Phi$ for requests to pages in L_0 when no forgiveness step is applied. Note that Φ depends only on the signature $\chi = (x_1, \ldots, x_k)$ of the layer representation. We use $\chi = 0$ for the cone signature $(0|0|\ldots|0)$ and $\chi = e_i$ for the i-th unit vector $(0|\ldots|x_i = 1|\ldots|0)$. If $\chi = 0$ we have $\Phi = 0$. Otherwise, let i be the largest index such that $x_i > 0$. Since all lazy attack sequences have the same cost, we consider Φ as the cost of i consecutive requests, each of them to a page in the (current) first layer. For the layer representation L of the current

offset function, we let $cost_1(L)$ denote the probability of cache miss for a page p in L_1, i.e. $pb(p \notin C_k)$ in the selection process.

We start with the case when all layers are singletons except some layer L_i. The potential Φ for this particular case is given in Lemma 1. For some arbitrary values i, n, and γ, where $0 < i < \gamma \leq k$ consider the signatures $\chi = n \cdot e_i$ and $\chi' = n \cdot e_i + e_{\gamma-1}$; let L and L' be their corresponding layer representations. We define the difference in the cost for a request in L_1: $f(i, n, \gamma) = cost_1(\chi') - cost_1(\chi)$. In the special case $\gamma = k$ it represents $\Delta cost_1$ upon a request in L_0. The value for $f(i, n, \gamma)$ can be computed exactly and is given in Lemma 2, and in Lemma 3 we show that $f(i, n, \gamma)$ is an upper bound on $\Delta cost_1$ for a whole class of signatures. Lemma 4 provides an identity for approximating $\Delta\Phi$ for a request in L_0. The proofs for all these results are given in the full version.

Lemma 1. *Let $\chi = n \cdot e_i$ be the signature of L, where $n > 0$ and $0 < i < k$. We have $\Phi(\chi) = n \cdot (H_{i+n} - H_n)$.*

Lemma 2. *It holds that $f(i, n, \gamma) = \frac{1}{n+\gamma} \prod_{j=i}^{\gamma-1} \frac{j}{n+j}$.*

Lemma 3. *Consider a signature $\chi = (x_1| \ldots |x_k)$, and let i be the minimal index with $x_j = 0$ for all $j > i$. Also, let $\chi' = \chi + e_{\gamma-1}$, $i < \gamma \leq k$. For $n = x_1 + \cdots + x_i$, we have $cost_1(\chi') - cost_1(\chi) \leq f(i, n, \gamma)$.*

Lemma 4. *It holds that $\sum_{j=1}^{i} f(i-j+1, 1, \gamma-j+1) = H_\gamma - H_{\gamma-i} - \frac{i}{\gamma+1}$, for any i and γ with $i < \gamma$.*

Theorem 1. *For a request to a page $p \in L_0$ where no forgiveness is applied, let i be the largest index with $x_i > 0$; $i = 0$ if we are in a cone. We have that:*

$$H_{k-i} - H_1 \leq \Delta\Phi \leq H_k - H_1 - i/(k+1).$$

Proof. For $i = 0$, in a cone we have $\Delta\Phi = H_k - 1$ by Lemma 1. If $i > 0$, let L and L', and χ and $\chi' = \chi + e_{k-1}$ denote the layers and their corresponding signatures before and after the request to p respectively. We consider the cost of a sequence of i consecutive requests $p_1, \ldots, p_i$, each of these to pages in the current L_1. For each $j = 1, \ldots, i$ let χ^j and χ'^j denote the signatures before processing p_j. After the whole sequence is processed, we have $\chi = 0$ with $\Phi = 0$ and $\chi' = e_{k-i-1}$ with $\Phi' = H_{k-i} - H_1$ by Lemma 1. We get:

$$\Delta\Phi = H_{k-i} - H_1 + \sum_{j=1}^{i} \left(cost_1(\chi'^j) - cost_1(\chi^j) \right)$$

Since $cost_1(\chi'^j) - cost_1(\chi^j)$ is non-negative, the left inequality holds.

Now we bound $cost_1(\chi'^j) - cost_1(\chi^j)$ using Lemma 3. Before processing page p_j we have $x^j_{i-j+1} > 0$, $x^j_l = 0$ for all indices $l > i - j + 1$ and $\chi'^j = \chi^j + e_{\gamma-1}$ with $\gamma = k - j + 1$. Denoting $n^j = x^j_1 + \cdots + x^j_{i-j+1}$, and using the fact that f is decreasing in n, $n^j > 0$ for all $j \leq i$, and the result in Lemma 4, we get:

$$\Delta\Phi \leq \sum_{j=1}^{i} f(i-j+1, n_j, k-j+1) + H_{k-i} - H_1 \leq H_k - H_{k-i} - \frac{i}{k+1} + H_{k-i} - H_1 \, .$$

3.2 Competitiveness and Bookmarks

Having obtained a tighter bound on $\Delta\Phi$ for requests in L_0, we get improved savings using a second potential Ψ. To define $\Psi(L)$, we first introduce the concept of *chopped signature*. For some signature $\chi = (x_1|\dots|x_k)$, let i be the largest index such that $x_i > 0$. The chopped signature corresponding to χ is $\overline{\chi} = (\overline{x_1}|\dots|\overline{x_k})$, where $\overline{x_i} = x_i - 1$ and $\overline{x_j} = x_j$ for all $j \neq i$. If we are in a cone and $\chi = 0$ we define $\overline{\chi} = \chi$. We define Ψ as $\Psi(L) = \frac{1}{k+1}\sum_{i=1}^{k-1} i \cdot \overline{x_i}$. Note that $\Psi(L) = 0$ if $\chi = 0$ or $\chi = e_i$ and otherwise we have $\Psi(L) > 0$.

Fact 1. *For a request to page $p \in L_i$, $i > 0$, we have $\Delta\Psi = -\frac{1}{k+1}\sum_{j=i}^{k-1} \overline{x_j}$.*

To prove that EQUITABLE2 is H_k-competitive, it suffices to show that for each request $cost + \Phi + \Psi \leq H_k \cdot cost_{OPT}$, as both Φ and Ψ are non-negative.

Lemma 5. *If no forgiveness is done then $\Delta cost + \Delta\Phi + \Delta\Psi \leq H_k \cdot \Delta cost_{OPT}$.*

Proof. We first analyze the case for a request $p \in L_i$, with $i > 0$. We have $\Delta cost + \Delta\Phi = 0$ by the definition of Φ and $\Delta cost_{OPT} = 0$. By Fact 1 $\Delta\Psi \leq 0$ and we are done. For requests to pages in L_0, both the algorithm and OPT incur a cost of one, and thus $\Delta cost = 1$ and $\Delta cost_{OPT} = 1$. It remains to show that $\Delta\Psi + \Delta\Phi \leq H_k - 1$. We analyze separately the case when we are in a cone. In this case, by definition $\Delta\Psi = 0$, and by Lemma 1 we obtain $\Delta\Phi = H_k - 1$. In the following we assume we are not in a cone upon the L_0 request. Let i be the largest index with $x_i \neq 0$. By the update rule, we get that $x'_{k-1} = x_{k-1} + 1$ and $x'_j = x_j$ for all $j \neq k-1$. For the chopped signature $\overline{\chi'}$ this implies $\overline{x'_j} = \overline{x_j}$ for all $j \neq i$ and $\overline{x'_i} = \overline{x_i} + 1$, because $i \neq k$ as L_k is always singleton. It follows $\Delta\Psi = i/(k+1)$. On the other hand we have by Theorem 1 that $\Delta\Phi \leq H_k - H_1 - i/(k+1)$.

Theorem 2. EQUITABLE2 *is H_k-competitive and uses $2 + \frac{\sqrt{5}-1}{2} \cdot k$ bookmarks.*

Proof. If the support size reaches the threshold $k + x$, i.e. x bookmarks, we apply upon a request from L_0 the forgiveness mechanism from [14]. Recall that we move the requested page artificially into L_1, and then we process it as if it was requested from L_1. We have $\Delta cost = 1$ and $\Delta cost_{OPT} = 0$. Like in [14], we need to prove that $1 + \Delta\Phi + \Delta\Psi \leq 0$. Denote by χ the current signature, and let $x = \sum_{i=1}^{k} x_i$ be the number of bookmarks used by the algorithm. We have that $\Delta\Phi = -cost_1(\chi)$. We get that $1 + \Delta\Phi$ is the probability that a page in L_1 is in the algorithm's cache, which by the selection process of ONLINEMIN is at most $k/|S| = k/(x+k)$. Using the result in Fact 1 and the fact that $\sum_{j=1}^{k-1} \overline{x_j} = x - 1$, we need to ensure that: $\frac{k}{x+k} - \frac{x-1}{k+1} \leq 0$. Solving this inequality, we get x is at most $\frac{\sqrt{5}-1}{2}k + 2$. Therefore, EQUITABLE2 needs only $\frac{\sqrt{5}-1}{2}k + 2 \approx 0.62k$ bookmarks. The cases where no forgiveness occurs are covered by Lemma 5.

In Theorem 3 and Theorem 4 we show that EQUITABLE2 cannot be both H_k-competitive and use $o(k)$ bookmarks, but that it can trade competitiveness for bookmarks. The proofs of these results are provided in the full version.

Theorem 3. *If* EQUITABLE2 *uses* $t \leq k/4$ *bookmarks, it is not* H_k*-competitive.*

Theorem 4. *There exist implementations of* EQUITABLE2 *that use* $k/(1+c)$ *bookmarks and are* $(H_k + c)$*-competitive, for* $k > 1$ *and* $c \geq 1$.

4 Partition

In this section we prove the conjecture in [14] that there exists a strongly competitive paging algorithm using $o(k)$ bookmarks. We propose a variation of the PARTITION algorithm [3], denoted PARTITION2, using $\Theta(k/\log k)$ bookmarks. We also give a simple lower bound showing that for any H_k-competitive randomized paging algorithm, the number of pages having non-zero probability of being in cache is at least $k + k/H_k$. This leads to a lower bound of k/H_k bookmarks for all algorithms which store all non-zero probability pages, i.e. representation of the approximated offset function, and have a deterministic forgiveness step. Note that this bound holds for all known H_k-competitive algorithms.

4.1 Partition

We give a brief description of the PARTITION algorithm in [3]. A crucial difference compared to EQUITABLE is that while the distribution of the cache configurations depends only on the current offset function for EQUITABLE, PARTITION uses a more detailed representation of the offset function, which we denote in the following *set-partition*. It partitions the whole pageset into a sequence of disjoint sets $S_\alpha, S_{\alpha+1}, \ldots, S_{\beta-1}, S_\beta$ and each set S_i with $i < \beta$ has a *label* k_i. Initially $\beta = \alpha + 1$, S_β contains the first k pairwise distinct pages, the remaining pages are in S_α, and $k_\alpha = 0$. Throughout the computation S_β contains all revealed pages and S_α all Opt-miss pages. The set-partition is updated as follows:

- if $p \in S_\alpha$: $S_\alpha = S_\alpha \setminus \{p\}$, $S_{\beta+1} = \{p\}$, $k_\beta = k - 1$, and $\beta = \beta + 1$.
- if $p \in S_i$, with $\alpha < i < \beta$: $S_i = S_i \setminus \{p\}$, $S_\beta = S_\beta \cup \{p\}$, and $k_j = k_j - 1$, $(i \leq j < \beta)$. Additionally, if there are labels which become zero, let j be the largest index such that $k_j = 0$; we set $S_j = S_\alpha \cup \cdots \cup S_j$ and $\alpha = j$.
- if $p \in S_\beta$: nothing changes

In [3] it was shown that the following invariants on the labels hold: $k_\alpha = 0$ and $k_i > 0$ for all $i > 0$; $k_{\beta-1} = k - |S_\beta|$. Furthermore, it holds at all times that $k_i = (k_{i-1} + |S_i|) - 1$. The probability distribution of the cache content can be described as the outcome of the following selection process on the set-partition:

- $\mathcal{C}_\alpha = \emptyset$
- For $\alpha < i < \beta$ choose p uniformly at random from $\mathcal{C}_{i-1} \cup S_i$ and then set $\mathcal{C}_i = (\mathcal{C}_{i-1} \cup S_i) \setminus \{p\}$
- $\mathcal{C}_\beta = \mathcal{C}_{\beta-1} \cup S_\beta$.

Note that, whereas for the selection process of ONLINEMIN it holds that $|C_i| = i$ $(0 \leq i \leq k)$, for PARTITION we have that $|\mathcal{C}_j| = k_j$ $(\alpha \leq j \leq \beta - 1)$.

Lemma 6 ([3, Lemma 3]). *If p is requested from S_i, where $\alpha < i < \beta$, the probability that p is not in the cache of* PARTITION *is at most* $\sum_{i \le j < \beta} \frac{1}{k_j+1}$.

Apart from obeying the probability distribution, PARTITION must satisfy two constraints: it must not evict pages upon a cache hit and it must not evict more than one page upon a cache miss. For any set $\mathcal{C}_i$, the membership of a page to $\mathcal{C}_i$ is encoded with a marking system on pages as follows. If a page is in set S_i, where $\alpha < i < \beta$, it has either no mark or a series of marks $i, i+1, \ldots, j-1, j$. If p has no mark then $p \notin \mathcal{C}_i$ and otherwise it is in the selection sets $\mathcal{C}_i, \mathcal{C}_{i+1}, \ldots, \mathcal{C}_{j-1}, \mathcal{C}_j$. The cache of PARTITION is at all times $\mathcal{C}_\beta$, with $|\mathcal{C}_\beta| = k$. For a page $p \in S_i$ it suffices to store the value m_p of the highest mark or $i-1$ if p has no mark.

Initially there are only the two sets S_α and S_β and thus no marks. If the requested page $p \in S_\beta$ nothing changes. If $p \in S_\alpha$ first the set-partition is updated, where β is increased by 1 and we have to determine $\mathcal{C}_{\beta-1}$. A page q is chosen uniformly at random from the k elements $\mathcal{C}_{\beta-2} \cup S_{\beta-1}$ (the cache content before the request), and this element is the only one not receiving a $\beta - 1$ mark. The page q is replaced in the cache by the requested page p. We now turn to the case $p \in S_i$, where $\alpha < i < \beta$. If p is in cache then $m_p = \beta - 1$ and we are done. Otherwise let $j \le \beta - 1$ be the lowest index such that $p \notin \mathcal{C}_j$. We choose uniformly at random a page $q \in \mathcal{C}_j$ and set $m_p = m_q$ and $m_q = j - 1$, i.e. p steals the marks of q. We repeat this until $m_p = \beta - 1$. The page which loses its $\beta - 1$ mark is replaced in cache by p. Afterwards the set-partition is updated.

4.2 Partition2

PARTITION2 is a variant of PARTITION which uses (deterministic) forgiveness to reduce the space usage from arbitrarily high bookmarks to $O(k/\log k)$ bookmarks. A lower bound is provided which shows that this bound is asymptotically optimal for algorithms using deterministic forgiveness. Unlike previous works, when a forgiveness step must be applied, we distinguish between two cases and apply two distinct forgiveness rules accordingly. The first of them is the same one used by EQUITABLE2 and covers only a single request, and the second one is a *forgiveness phase* which spans consecutive requests. To apply the forgiveness step of EQUITABLE2, we provide an embedding of the set-partition into the layer representation of the offset function. Based on this embedding, we give a simple potential function which depends only on the signature of the offset function.

Layer Embedding. We provide an embedding of the set-partition into the layer representation of the offset functions, as used by EQUITABLE. The layers become ordered sets and contain pages and set identifiers, the latter of which we visualize by $\star$. The initialization does not change and no set identifiers are present. The update rule changes mainly for the case $p \in L_0$:

$$L_{k-1} = (L_{k-1}, L_k, \star),\ L_k = \{p\}.$$

In the case $p \in L_i$, upon the merge operation $L_{i-1} \cup L_i \setminus \{p\}$, we remove p from L_i and concatenate L_{i-1} with L_i without removing any set identifier. Upon

merging L_1 into L_0 we delete all set identifiers from the resulting layer L_0. The following fact follows inductively.

Fact 2. *For L_i, with $i > 0$ and $|L_i| = 1 + x_i$, it holds that L_i contains exactly x_i set identifiers. Moreover, if $x_i > 0$ then the last element in L_i is a set identifier.*

We describe how to obtain the sets of the set-representation. Let j be maximal such that $x_j > 1$. We have $S_\beta = L_{j+1} \cup \cdots \cup L_k$ and $S_\alpha = L_0$. A set $S_{\alpha+j}$, where $1 < j < \beta - \alpha$ consists of all pages between the $(j-1)$-th and the j-th set identifier; for $j = 1$, $S_{\alpha+1}$ consists of all support pages until the first set identifier. We say that each set $S_{\alpha+j}$, $0 < j < \beta - \alpha$, is *represented* by the j'th set identifier. As long as no pages are moved into S_α, the correspondence between the layer representation and the set-partition follows immediately from the update rules. Otherwise, by Lemma 7 and noticing that each L_i with $x_i > 0$ ends in a set delimiter, we obtain that p is in L_1 and moreover the pages moved to S_α correspond to $L_1 \setminus \{p\}$.

Lemma 7. *Let $S_a, S_{a+1}, \ldots, S_b$ be the sets whose identifiers are in layer L_i, $i \geq 0$. We have $k_b = i$, $k_{a+j} \geq i$ for $0 \leq j < b - a$.*

Proof. Due to space limitations, the proof is provided in the full version.

Lemma 8. *If p is requested from L_i, where $i > 0$, the probability that p is not in the cache of* Partition *is at most $\sum_{j \geq i} \frac{x_j}{j+1}$.*

Proof. If $p \in S_\beta$, then it is in a revealed layer L_i and thus $x_j = 0$ for all $j \geq i$ and the result holds. Let S_{i^*} be the set with $p \in S_{i^*}$, $\alpha < i^* < \beta$. Then by Lemma 6 we have the probability bounded by $\sum_{i^* \leq j^* < \beta} \frac{1}{k_{j^*}+1}$. All sets S^*_j, where $i^* \leq j^* < \beta$ have their identifier in some layer L_j with $j \geq i$ and using Lemma 7 we obtain $\frac{1}{k_{j^*}+1} \leq \frac{1}{j+1}$. Since each layer L_j contains exactly x_j identifiers the statement follows.

Forgiveness. Forgiveness is applied when the support size reaches a threshold of $k + 3t$ (we define t later) and a page in L_0 is requested. Depending on the support we have two kinds of forgiveness: *regular forgiveness* and an *extreme forgiveness mode*. The regular forgiveness is applied if $|L_1| + \cdots + |L_t| > 2t$ and is an adaptation of the forgiveness step of Equitable2. If a page p is requested from L_0 (equivalent to S_α), we first identify a page q satisfying that $q \in S_{\alpha+1} \cap L_1$. Note that there always exists such a page, since $k_{\alpha+1} \geq 1$ and $|S_1| = k_1 + 1$ and at least one of them is in L_1. We move q to L_0 and replace it, together with its marks, by p. Then we perform the set-partition and mark update where p is requested from $S_{\alpha+1}$. We stress that in terms of the layer representation (used by e.g. Equitable), we replace the requested page with an existing page in L_1, and replacing $q \in L_1$ by p and requesting p leads to the same offset function when the forgiveness step in [14] is applied. This has a cost of 1 for Partition and a cost of 0 for OPT. The support size decreases by $|L_1| - 1 \geq 0$.

The extreme forgiveness mode is applied if $|L_1| + \cdots + |L_t| \leq 2t$. We simply apply regular forgiveness for any page request in L_0 starting with the current one. This extreme forgiveness mode ends when reaching a cone.

Competitive Ratio. We use PARTITION with the forgiveness rule for $t = \lceil \frac{k}{\ln k} \rceil$ from the previous paragraph if $k > 10$ and denote the resulting algorithm PARTITION2. For $k \leq 10$ we use the regular forgiveness if the support size reaches $2k$.

Theorem 5. PARTITION2 *uses* $\Theta(\frac{k}{\log k})$ *bookmarks and is* H_k*-competitive.*

Proof. The space bound follows from the fact that the support size never exceeds $k + 3t$ for $k > 10$, where $t = \lceil \frac{k}{\ln k} \rceil$. It remains to show that PARTITION2 is still H_k-competitive. We use the following potential function on the layer embedding:

$$\Phi = \sum_{j=1}^{k-1} x_j \cdot (H_{j+1} - 1) \; .$$

We denote by *cost* the cost of PARTITION2 and by OPT the cost of the optimal offline algorithm. We have to show that $cost \leq H_k \cdot OPT$ holds after each request. In all cases except the extreme forgiveness we show that the following holds before and after each request: $\Phi + cost \leq H_k \cdot OPT$. This leads to $cost \leq H_k \cdot OPT$ since $\Phi \geq 0$. When applying the extreme forgiveness we assume that the potential inequality holds before the phase and show that it holds at the end of the phase, but not necessarily during the phase. For requests during the phase we argue directly that it always holds $cost \leq H_k \cdot OPT$.

Let p be the requested page. If $p \in L_0$ without forgiveness, $\Delta OPT = 1$ and x_{k-1} increases by 1, which implies that $\Delta\Phi + \Delta cost = H_k - 1 + 1 = 1 \cdot H_k$. If p is from some layer L_i, where $0 < i \leq k$, we use the bound on the cache miss probability from Lemma 8

$$\Delta\Phi + \Delta cost \leq -\sum_{j \geq i} \frac{x_j}{j+1} + \sum_{j \geq i} \frac{x_j}{j+1} \leq 0 \leq H_k \cdot \Delta OPT.$$

Now we analyze the cases where forgiveness occurs for $k > 10$. Assume that $|L_1| + \cdots + |L_t| \geq 2t+1$ which implies that $x_1 + \cdots + x_t \geq t+1$. We perform just one forgiveness step, yielding $\Delta cost = 1$ and $\Delta OPT = 0$. We show $\Delta\Phi \leq -1$:

$$\Delta\Phi = -\sum_{j=1}^{k-1} \frac{x_j}{j+1} \leq -\sum_{j=1}^{t} \frac{x_j}{t+1} = -\frac{t+1}{t+1} = -1.$$

Now assume that $x_{t+1} + \cdots + x_{k-1} \geq 2t$. Before we start the extreme forgiveness mode, we have that $\Phi \geq \sum_{j=t+1}^{k-1} x_j (H_{j+1} - 1) \geq 2t(H_{t+2} - 1)$. For $t = \lceil \frac{k}{\ln k} \rceil$ and $H_x \geq \ln x$ we obtain: $\Phi \geq \frac{2k}{\ln k}(\ln k - \ln\ln k - 1) \geq k$, if $k > 10$. Right before the phase starts we have $cost + \Phi \leq H_k \cdot OPT$, where $\Phi \geq k$ which is equivalent to $cost \leq H_k \cdot OPT - k$. Reaching the next cone implies at most $k-1$ unrevealed requests and thus the cost during this phase is bounded by $k-1$. This implies that $cost \leq H_k \cdot OPT$ holds. Since in a cone $\Phi = 0$ we also have at the end of the phase the invariant $cost + \Phi \leq H_k \cdot OPT$.

For the case $k \leq 10$ the analysis of the extreme forgiveness does not hold. In this case we use only the regular forgiveness step if we have k bookmarks. Using $x_1 + \cdots + x_{k-1} = k$ the same argument as before leads to $\Delta\Phi \leq -1$.

Lemma 9. *For any H_k-competitive algorithm A there exists an input such that the maximal number of pages with non-zero probability of being in A's cache is at lest $k + k/H_k$.*

Proof. Due to space limitations, the proof is provided in the full version.

References

1. Moruz, G., Negoescu, A.: Improved space bounds for strongly competitive randomized paging algorithms. Technical report, Goethe University Frankfurt am Main (2013)
2. Bein, W.W., Larmore, L.L., Noga, J.: Equitable revisited. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 419–426. Springer, Heidelberg (2007)
3. McGeoch, L.A., Sleator, D.D.: A strongly competitive randomized paging algorithm. Algorithmica 6(6), 816–825 (1991)
4. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3, 77–119 (1988)
5. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)
6. Belady, L.A.: A study of replacement algorithms for virtual-storage computer. IBM Systems Journal 5(2), 78–101 (1966)
7. Albers, S.: Online algorithms: a survey. Mathematical Programming 97(1-2), 3–26 (2003)
8. Borodin, A., El-Yaniv, R.: Online computation and competitive anlysis. Cambridge University Press (1998)
9. Young, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica 11(6), 525–541 (1994)
10. Moruz, G., Negoescu, A.: Outperforming LRU via competitive analysis on parametrized inputs for paging. In: Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1669–1680 (2012)
11. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. Journal of Algorithms 12(4), 685–699 (1991)
12. Chrobak, M., Koutsoupias, E., Noga, J.: More on randomized on-line algorithms for caching. Theoretical Computer Science 290(3), 1997–2008 (2003)
13. Achlioptas, D., Chrobak, M., Noga, J.: Competitive analysis of randomized paging algorithms. Theoretical Computer Science 234(1-2), 203–218 (2000)
14. Bein, W.W., Larmore, L.L., Noga, J., Reischuk, R.: Knowledge state algorithms. Algorithmica 60(3), 653–678 (2011)
15. Brodal, G.S., Moruz, G., Negoescu, A.: Onmin: A fast strongly competitive randomized paging algorithm. In: Theory of Computing Systems (2012)
16. Bein, W.W., Fleischer, R., Larmore, L.L.: Limited bookmark randomized online algorithms for the paging problem. Information Processing Letters 76(4-6), 155–162 (2000)
17. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. In: Proc. 35th Symposium on Foundations of Computer Science, pp. 394–400 (1994)

No-Wait Flowshop Scheduling Is as Hard as Asymmetric Traveling Salesman Problem

Marcin Mucha[1,⋆] and Maxim Sviridenko[2,⋆⋆]

[1] University of Warsaw
mucha@mimuw.edu.pl
[2] University of Warwick
M.I.Sviridenko@warwick.ac.uk

Abstract. In this paper we study the classical no-wait flowshop scheduling problem with makespan objective ($F|no-wait|C_{max}$ in the standard three-field notation). This problem is well-known to be a special case of the asymmetric traveling salesman problem (ATSP) and as such has an approximation algorithm with logarithmic performance guarantee. In this work we show a reverse connection, we show that any polynomial time α-approximation algorithm for the no-wait flowshop scheduling problem with makespan objective implies the existence of a polynomial-time $\alpha(1+\varepsilon)$-approximation algorithm for the ATSP, for any $\varepsilon > 0$. This in turn implies that all non-approximability results for the ATSP (current or future) will carry over to its special case. In particular, it follows that no-wait flowshop problem is APX-hard, which is the first non-approximability result for this problem.

1 Introduction

1.1 Problem Statement and Connection with ATSP

A *flowshop* is a multi-stage production process with the property that all jobs have to pass through several stages. There are n jobs J_j, with $j = 1, \ldots, n$, where each job J_j is a chain of m operations $O_{j1}, \ldots, O_{jm}$. Every operation O_{ji} is preassigned to the machine M_i. The operation O_{ji} has to be processed for t_{ji} time units at its stage; the value t_{ji} is called its *processing time* or its *length*. In a feasible schedule for the n jobs, at any moment in time every job is processed by at most one machine and every machine executes at most one job. For each job J_j, operation O_{ji-1} always is processed before operation O_{ji}, and each operation is processed without interruption on the machine to which it was assigned.

In the *no-wait flowshop problem* (No-Wait-Flowshop) we require two additional constraints to be satisfied:

⋆ Part of this work was done while visiting DIMAP at the University of Warwick. Work supported by ERC StG project PAAl no. 259515.
⋆⋆ Work supported by EPSRC grants EP/J021814/1, EP/D063191/1, FP7 Marie Curie Career Integration Grant and Royal Society Wolfson Research Merit Award.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 769–779, 2013.

- There is no waiting time allowed between the execution of consecutive operations of the same job. Once a job has been started, it has to be processed without interruption, operation by operation, until it is completed.
- Each machine processes the jobs in the same order (i.e. we only allow so-called *permutation schedules*). Note that this applies to all jobs, not only those with non-zero processing time on a given machine. In other words, one should treat zero length operations as having an infinitely small, but non-zero length.

Our goal is to find a permutation σ of jobs that minimizes the *makespan* (or *length*) $C_{\max}(\sigma)$ of the schedule, i.e., the maximum completion time among all jobs. The minimum makespan among all feasible schedules is denoted by $C^*_{\max}$.

Consider two jobs J_i and J_j that are processed one after another in a no-wait permutation schedule. Let $\delta(i,j)$ be the minimum time we need to wait to start the job J_j after starting the job J_i. What is the value of $\delta(i,j)$? Clearly we need to wait at least t_{i1}. But since job j cannot wait on the second machine, we also need to wait at least $t_{i1} + t_{i2} - t_{j1}$. Generalizing this leads to the following expression

$$\delta(i,j) = \max_{q=1,\ldots,m} \left\{ \sum_{k=1}^{q} t_{ik} - \sum_{k=1}^{q-1} t_{jk} \right\}. \tag{1}$$

Note that δ satisfies the triangle inequality, i.e. $\delta(i,j) \leq \delta(i,k) + \delta(k,j)$ for any jobs J_i, J_j, J_k. The easiest way to see this is by considering the 'waiting time' intuition that led to its definition.

Let $L(j) = \sum_{k=1}^{m} t_{jk}$ be the total processing time (or length) of job J_j. Then a no-wait schedule that processes the jobs in order σ has makespan

$$C_{\max}(\sigma) = \sum_{k=1}^{n-1} \delta(\sigma_k, \sigma_{k+1}) + L(\sigma_n). \tag{2}$$

In the asymmetric traveling salesman problem (ATSP), we are given a complete directed graph $G = (V, E)$ with arc weights $d(u,v)$ for each $u, v \in V$. It is usually assumed that the arc weights satisfy the semimetric properties, i.e. $d(u,u) = 0$ for all $u \in V$ and $d(u,v) \leq d(u,w) + d(w,v)$ for all $u, w, v \in V$. The goal is to find a Hamiltonian cycle, i.e. a cycle that visits every vertex exactly once, of minimum total weight. The asymmetric traveling salesman path problem (ATSPP) is defined analogously, the only difference is that we are looking for a path that starts and ends in arbitrary but distinct vertices and visits all other vertices exactly once along the way. The distance function δ can be used to cast No-Wait-Flowshop as ATSP by introducing a dummy job consisting of m zero length operations, and modifying δ slightly by setting $\delta(i,i) = 0$ for all $i = 1, \ldots, n$. The role of the dummy job is to emulate the $L(\sigma_n)$ term in (2).

The No-Wait-Flowshop was first defined in the 1960 by Piehler [11] who also noticed this problem is a special case of the ATSP. This connection was also later noticed by Wismer [17]. The No-Wait-Flowshop is usually denoted $F|no-wait|C_{max}$ using the standard three-field scheduling notation (see e.g.

Lawler et al. [9]). Although no-wait shop scheduling problems arise naturally in many real-life scenarios (steel manufacturing, hot potato routing) they sometime behave in a way uncommon for other scheduling problems, e.g. speeding up a machine may actually increase the makespan [14].

1.2 Known Results

For the No-Wait-Flowshop with two machines, the distance matrix of the corresponding ATSP has a very special combinatorial structure, and the famous subtour patching technique of Gilmore and Gomory [5] yields an $O(n \log n)$ time algorithm for this case. Röck [12] proves that the three-machine no-wait flowshop is strongly $\mathcal{NP}$-hard, refining the previous complexity result by Papadimitriou and Kanellakis [10] for four machines. Hall and Sriskandarajah [6] provide a thorough survey of complexity and algorithms for various no-wait scheduling models.

We say that a solution to an instance I of a problem is *ρ-approximate* if its value is at most $\rho|OPT|$, where $|OPT|$ is the value of the optimum solution to I. We say that an approximation algorithm has *performance guarantee* ρ for some real $\rho > 1$, if it delivers ρ-approximate solutions for all instances. Such an approximation algorithm is then called a *ρ-approximation* algorithm. A family of polynomial time $(1 + \varepsilon)$-approximation algorithms over all $\varepsilon > 0$ is called a *polynomial time approximation scheme* (PTAS).

For the No-Wait-Flowshop with fixed number of machines, i.e. $Fm|no - wait|C_{max}$ in standard notation, there exists a polynomial time approximation scheme [15]. The only known approximability results for the general case are $\lceil m/2 \rceil$-approximation algorithm from [13] or algorithms designed for the ATSP with performance guarantees $\log_2 n$ [4], $0.999 \log_2 n$ [2], $0.84 \log_2 n$ [7], $0.66 \log_2 n$ [3], $O\left(\frac{\log n}{\log \log n}\right)$ [1].

We remark that the strongest known negative result for the general ATSP with the triangle inequality is due to Karpinski et al. [8]. They prove that unless $\mathcal{P} = \mathcal{NP}$, the ATSP with triangle inequality cannot have a polynomial time approximation algorithm with performance guarantee better than 75/74. We are not aware of any known non-approximability results for the No-Wait-Flowshop.

1.3 Our Results and Organization of the Paper

In this paper we show that No-Wait-Flowshop is as hard to approximate as ATSP, i.e. given an α-approximation algorithm for No-Wait-Flowshop one can approximate ATSP with ratio arbitrarily close to α. In particular, this gives APX-hardness for No-Wait-Flowshop. It is worth noting that No-Wait-Flowshop has recently received increased interest, since it was viewed as a (potentially) easy case of ATSP, and possibly a reasonable first step towards resolving the general case. It is for this reason that it was mentioned by Shmoys and Williamson [16] in their discussion of open problems in approximation algorithms. Our results settle this issue.

We also give an $O(\log m)$-approximation algorithm for No-Wait-Flowshop. On one hand, this can be seen as an improvement over the $\lceil m/2 \rceil$-approximation from [13]. But this result also shows that, unless we obtain an improved approximation for ATSP, the number of machines used by any reduction from ATSP to No-Wait-Flowshop has to be $e^{\Omega(\log n / \log \log n)}$. In this sense our reduction, which uses a number of machines polynomial in n, cannot be significantly improved.

The paper is organized as follows. In Section 2 we give the reduction from ATSP to No-Wait-Flowshop. We begin by showing in Subsection 2.1 that instead of general ATSP instances, it is enough to consider instances of ATSPP with integer edge weights that are small relative to $|OPT|$ and polynomial in n. We then proceed with the reduction. We start by showing in Subsection 2.2 that any semi-metric can be represented as a No-Wait-Flowshop distance function with only a small additive error. This already shows that No-Wait-Flowshop distance functions are in no way "easier" than general semi-metrics. However, this is not enough to reduce ATSPP to No-Wait-Flowshop, because of the last term in the objective function (2). To make this last term negligible, we blow-up the ATSPP instance without significantly increasing the size of the corresponding No-Wait-Flowshop instance, by using a more efficient encoding. This is done in Subsection 2.3.

Finally, in Section 3 we present the $O(\log m)$-approximation algorithm for No-Wait-Flowshop.

2 Non-approximability Results for No-Wait-Flowshop

2.1 Properties of the ATSP Instances

In the rest of the paper we will use OPT to denote an optimal solution of the given ATSP instance and $|OPT|$ the value of such an optimal solution.

Lemma 1. *For any instance $G = (V, d)$ of* ATSP *and any $\varepsilon > 0$, one can construct in time poly$(n, 1/\varepsilon)$ another instance $G' = (V', d')$ of* ATSP *with $|V'| = O(n/\varepsilon)$, such that:*

1. *all arc weights in G' are positive integers and the maximal arc weight $W' = O\left(\frac{n \log n}{\varepsilon}\right)$ (regardless of how large the original weights are);*
2. *$W' \leq \varepsilon |OPT'|$, where OPT' is an optimal solution to G'.*

and given an α-approximate solution to G' one can construct an $\alpha(1 + O(\varepsilon))$-approximate solution to G in time poly$(n, 1/\varepsilon)$.

Proof. Given an instance $G = (V, d)$ of ATSP, we first run the $\log n$-approximation algorithm for the ATSP from [4]. Let R be the value of the approximate solution found by the algorithm. We know that $|OPT| \leq R \leq \log n \cdot |OPT|$.

Then we add $\Phi = \frac{\varepsilon R}{n \log_2 n}$ to each arc weight and round each arc weight up to the closest multiple of Φ. Let $\bar{d}(u,v)$ be the new weight of the arc (u,v). We claim that the triangle inequality is still satisfied for new edge weights. Indeed, for any $u, w, v \in V$ we have

$$\bar{d}(u,v) \le d(u,v) + 2\Phi \le d(u,w) + d(w,v) + 2\Phi \le \bar{d}(u,w) + \bar{d}(w,v).$$

Moreover, the value of any feasible solution for the two arc weight functions d and $\bar{d}$ differs by at most $2\varepsilon R / \log n \le 2\varepsilon \cdot |OPT|$. We now divide the arc weights in the new instance by Φ. The resulting graph $\hat{G} = (V, \hat{d})$ has integral arc weights. Moreover, they all have values at most $O(\frac{n \log n}{\varepsilon})$, since $d(u,v) \le OPT$ for all $u, v \in V$ by triangle inequality. Finally, any α-approximate solution for $\hat{G}$ is also an $\alpha(1 + O(\varepsilon))$-approximate solution for G.

To guarantee the second property we apply the following transformation to $\hat{G}$. We take $N = \lceil 2/\varepsilon \rceil$ copies of $\hat{G}$. Choose a vertex u in $\hat{G}$ arbitrarily and let $u_1, \ldots, u_N$ be the copies of the vertex u in the copies of $\hat{G}$. We define a new graph $G' = (V', d')$ that consists of $N(n-1)+1$ vertices by merging the vertices $u_1, \ldots, u_N$ into a supervertex U, the remaining vertices of G' consist of N copies of $V \setminus \{u\}$.

If an arc of G' connects two vertices of the same copy of $\hat{G}$ then it has the same weight as the corresponding arc in $\hat{G}$. If an arc (x'_1, x'_2) connects a copy of a vertex x_1 and a copy of a vertex x_2 belonging to different copies of $\hat{G}$ then we define $d'(x'_1, x'_2) = \hat{d}(x_1, u) + \hat{d}(u, x_2)$, i.e. the weight is defined by the travel distance from w_1 to w_2 through the special supervertex U. By definition the maximal weight W' of an arc in G' is at most $2\hat{W}$, where $\hat{W}$ is the maximum weight of an arc in $\hat{G}$, and so the first constraint holds for G'.

Moreover, we claim that the value of the optimal Hamiltonian cycle OPT' in G' is exactly $N|O\hat{P}T|$, where $O\hat{P}T$ is an optimum solution for $\hat{G}$. Indeed, it is easy to see that there is a tour of length $\le N|O\hat{P}T|$ obtained by concatenating and short-cutting N optimal tours, one in each copy of $\hat{G}$. On the other hand, for any feasible tour T in G' we can replace any arc of T that connects vertices (say w_1 and w_2) in different copies of $\hat{G}$ by two arcs (w_1, U) and (U, w_2). Now we have a walk $\hat{T}$ through G' of the same length as T. $\hat{T}$ visits all the vertices of G' exactly once except for the vertex U which is visited multiple times. Therefore, $\hat{T}$ consists of a set of cycles that cover all vertices except U exactly once and vertex U is covered multiple times. We can reorder these cycles so that the walk first visits all vertices of one copy then all vertices of the second copy and so on. By applying short-cutting we obtain a collection of N Hamiltonian cycles, one for each copy of $\hat{G}$. Therefore, the original tour T in G' cannot be shorter than $N|O\hat{P}T|$, and so $|OPT'| = N|O\hat{P}T|$.

We now have

$$W' \le 2\hat{W} \le 2|O\hat{P}T| = 2|OPT'|/N \le \varepsilon |OPT'|,$$

so the second constraint is satisfied. The above argument is constructive, i.e. given a Hamiltonian cycle of length L in G', it produces a Hamiltonian cycle in G of length at most L/N in time $poly(n, 1/\varepsilon)$. □

Lemma 2. *Let $G = (V, d)$ be an instance of* ATSP *with $|V| = n$ and $d : V \times V \to \{0, \ldots, W\}$. Then, one can construct in time $O(n)$ an instance $G' = (V', d')$ of* ATSPP *with $|V'| = n+1$ and $d' : V \times V \to \{0, \ldots, 2W\}$, such that the optimal values of the two instances are the same. Moreover, given a solution S' of G', one can construct in time $O(n)$ a solution of G with value at most the value of S'.*

Proof. We fix a vertex $v \in V$ and define G' as follows:

- $V' = V \setminus \{v\} \cup \{v_{in}, v_{out}\}$, i.e. we split v into two vertices.
- For all pairs $x, y \in V \setminus \{v\}$ we put $d'(x, y) = d(x, y)$.
- For all $x \in V \setminus \{v\}$ we put $d'(v_{out}, x) = d(v, x)$ and $d'(x, v_{in}) = d(x, v)$, i.e. v_{in} inherits the incoming arcs of v and v_{out} inherits the outgoing arcs. We also put $d'(v_{out}, v_{in}) = 0$.
- All the remaining arcs get length of $2W$.

It is easy to verify that d' satisfies the triangle inequality.

We now need to show that the shortest Hamiltonian tour in G has the same length as the shortest Hamiltonian path in G'. Note that Hamiltonian tours in G correspond to Hamiltonian paths in G' starting in v_{out} and ending in v_{in}, and that this correspondence maintains the total length. Using this observation, for any tour in G, one can obtain a path in G' of the same length.

In the opposite direction, let us consider a path S' in G'. We will show how to transform S' without increasing its length, so that it begins in v_{out} and ends in v_{in}. We proceed in two steps. First, if S' does not begin in v_{out}, we break it before v_{out} and swap the order of the two resulting subpaths. This does not increase the length since any incoming arc of v_{out} has length $2W$.

Now, suppose that v_{in} is not the last vertex on S', i.e. it is visited between x and y for some $x, y \in V$ (v_{in} cannot be the first vertex, since v_{out} is). We remove v_{in} from S' and append it on the end. The total change in the length of the path is

$$\Delta = d'(x, y) + d'(z, v_{in}) - d'(x, v_{in}) - d'(v_{in}, y),$$

where z is the last vertex of the path. Using the definition of d' we get

$$\Delta \leq W + W - 0 - 2W \leq 0,$$

so this transformation does not increase the length of the path, which ends the proof. □

2.2 A Simple Embedding

In this section we show that jobs with the distance function δ in some sense form a universal space for all semi-metrics (approximately). More precisely, let $\mathcal{J}_{m,T}$ be the set of all m-machine jobs with all operations of length at most T, i.e. $\mathcal{J}_{m,T} = \{0, 1, \ldots, T\}^m$. Then

Theorem 1. *For any n-point semi-metric (V, d), where $d : V \to \{0, .., D\}$, there exists a mapping $f : V \to \mathcal{J}_{2nD,1}$, such that*

$$\delta(f(u), f(v)) = d(u, v) + 1 \text{ for all } u, v \in V,$$

where δ is the distance function defined by (1).

Proof. First we define a collection of $D + 1$ jobs $\mathcal{J}(D) = \{B_0^D, ..., B_D^D\}$ on $2D$ machines with all operations of length either zero or one. Obviously, $\mathcal{J}(D) \subseteq J_{2D,1}$. The job B_i^D consists of $D-i$ zero length operations that must be processed on machines $M_1, \ldots, M_{D-i}$, followed by D unit length operations that must be processed on machines $M_{D-i+1}, \ldots, M_{2D-i}$. The last i operations have zero length. By construction, $L(B_i^D) = D$ for $i = 0, \ldots, D$. Moreover, $\delta(B_i^D, B_j^D) = \max(i - j + 1, 0)$.

In the following, we will use the symbol $\cdot$ to denote concatenation of sequences, and in particular sequences of jobs. Let $a_i \in \mathcal{J}(D)$ and $b_i \in \mathcal{J}(D)$ for $i = 1, \ldots, k$. Consider the job $A = a_1 \cdot a_2 \cdot \ldots \cdot a_k \in J_{k2D,1}$ processed on $k2D$ machines $M'_1, \ldots, M'_{k2D}$. That is, job A has the same operation length on machine $M'_{(i-1)2D+r}$ as job a_i on machine M_r for $r = 1, \ldots, 2D$. Analogously, let $B = b_1 \cdot b_2 \cdot \ldots \cdot b_k \in J_{k2D,1}$. Then

$$\delta(A, B) = \max\{\delta(a_1, b_1), \ldots, \delta(a_k, b_k)\}. \tag{3}$$

This is because by making job B start X time steps after job A, we ensure that each sequence b_i of operations starts X time steps after the sequence a_i of operations. Therefore, in any feasible schedule $X \leq \max\{\delta(a_1, b_1), \ldots, \delta(a_k, b_k)\}$. On the other side, starting job B exactly $\max\{\delta(a_1, b_1), \ldots, \delta(a_k, b_k)\}$ time steps after the start of the job A gives a feasible schedule.

Let $V = \{v_1, ..., v_n\}$. We define $f(v_i) = B^D_{d(v_i,v_1)} \cdot \ldots \cdot B^D_{d(v_i,v_n)}$. Then by (3) we have

$$\delta(f(v_i), f(v_j)) = \max_k \{d(v_i, v_k) - d(v_j, v_k) + 1, 0\} = d(v_i, v_j) + 1.$$

The last equality follows from the triangle inequality and the fact that $d(v_j, v_j) = 0$. □

2.3 A More Efficient Embedding

Our main result concerning the relationship between ATSP and No-Wait-Flowshop is the following.

Theorem 2. *Let $G = (V, d)$ be an instance of* ATSP *with $|V| = n$ and let OPT be the optimum TSP tour for G. Then, for any constant $\varepsilon > 0$, there exists an instance I of* No-Wait-Flowshop*, such that given an α-approximate solution to I, we can find a solution to G with length at most*

$$\alpha(1 + O(\varepsilon))|OPT|.$$

Both I and the solution to G can be constructed in time poly$(n, 1/\varepsilon)$.

Proof. We start by applying Lemma 1 to G and then Lemma 2 to the resulting instance of ATSP. Finally, we scale all the distances up by a factor of $\lceil 1/\varepsilon \rceil$. In this way we obtain an instance $G' = (V', d')$ of ATSPP, such that:

- $n' = |V'| = O(n/\varepsilon)$.
- $n' \leq \varepsilon OPT'$ (this is due to scaling, since all arc weights are positive integers before scaling).
- G' has integral arc weights.
- $W' = O(n \log_2 n/\varepsilon^2)$ and $W' \leq \varepsilon |OPT'|$, where W' and OPT' are the maximum arc weight and the optimum solution for G', respectively.
- Given an α-approximate solution to G', one can obtain an $\alpha(1 + O(\varepsilon))$-approximate solution to G (using Lemma 1 and Lemma 2).

Note that one can simply encode G' as a No-Wait-Flowshop instance using Theorem 1. The problem with this approach is that the objective value in No-Wait-Flowshop contains an additional term that is not directly related to the distances in G'. If this term dominates the makespan, approximation algorithms for No-Wait-Flowshop are useless for the original ATSPP instance.

To overcome this obstacle we first blow G' up by creating N (to be chosen later) copies of it. Let $G_1 = (V_i, d_i), \ldots, G_N = (V_N, d_N)$ be these copies. We join these copies into a single instance $\hat{G} = (\hat{V}, \hat{d})$ by putting edges of length $2W'$ between all pairs of vertices from different copies. Note that any TSP path in $\hat{G}$ can be transformed into a path in which vertices of the same copy form a subpath, without increasing its cost. Therefore

Observation 3. *The cost of the optimum solution for $\hat{G}$ is $N|OPT'|+2W'(N-1)$. In the opposite direction, given a TSP path of cost C in $\hat{G}$, one can obtain a TSP path of cost $\frac{C-2W'(N-1)}{N}$ in G'.*

We transform $\hat{G}$ into a No-Wait-Flowshop instance as follows. We apply the construction of Theorem 1 to each G_i to obtain N identical jobsets $J_1, \ldots, J_N$. We then augment these jobs to enforce correct distances between jobs in different J_i. To this end we introduce new gadgets.

Lemma 3. *For any $N \in \mathbb{N}$ there exists a set of N jobs $H_1, \ldots, H_N$, each of the jobs using the same number of machines $O(D \log N)$ and of the same total length $O(D \log N)$, such that $\delta(H_i, H_i) = 1$ and $\delta(H_i, H_j) = D$ for $i \neq j$.*

Proof. We will use the following two jobs as building blocks: $H^0 = (10)^{2D}$ and $H^1 = 1^{2D}0^{2D}$ (x^D here means a sequence constructed by repeating the symbol x exactly D times). Note that they have the same total length of $2D$, the same number of machines $4D$, and that $\delta(H^0, H^1) = \delta(H^0, H^0) = \delta(H^1, H^1) = 1$ and $\delta(H^1, H^0) = D$.

Let k be smallest integer such that $\binom{2k}{k} \geq N$. Clearly $k = O(\log N)$. Consider characteristic vectors of all k-element subsets of $\{1, \ldots, 2k\}$, pick N such vectors $R_1, \ldots, R_N$. Now, construct H_i by substituting H^0 for each 0 in R_i and H^1 for each 1. Analogously to (3), we derive that the distances between H_i are as claimed. Also, the claimed bounds on the sizes of H_i follow directly from the construction. □

Using the above lemma it is easy to ensure correct distances for jobs in different J_i. Simply augment all jobs with gadgets described in Lemma 3, same gadgets for the same J_i, different gadgets for different J_i. Here $D = 2W' + 1$, so the augmentation only requires $O(W' \log N)$ extra machines and extra processing time.

This ends the construction of the instance of No-Wait-Flowshop. The optimum solution in this instance has cost

$$N|OPT'| + 2W'(N-1) + (Nn' - 1) + 2W'n' + O(W' \log N).$$

The $Nn' - 1$ term here comes from the additive error in Theorem 1, and the $2W'n' + O(W' \log N)$ term corresponds to the processing time of the last job in the optimum solution.

Given an α-approximate solution to the flowshop instance, we can obtain a TSP path $ALG_{\hat{G}}$ for $\hat{G}$ with cost

$$|ALG_{\hat{G}}| \leq \alpha(N|OPT'| + 2W'(N-1) + (Nn' - 1) + 2W'n' + O(W \log N)) \\ -(Nn' - 1) - 2W'n' - O(W' \log N),$$

which is just

$$\alpha(N|OPT'| + 2W'(N-1)) + (\alpha - 1)((Nn' - 1) + 2W'n' + O(W' \log N)).$$

As observed earlier, from this we can obtain a solution to G' with value at most

$$\frac{|ALG_{\hat{G}}| - 2W'(N-1)}{N}$$

which is bounded by

$$\frac{\alpha N|OPT'| + (\alpha - 1)(2W'(N-1) + (Nn' - 1) + 2W'n' + O(W' \log N))}{N}.$$

By taking $N = n'$ we can upper-bound this expression by

$$\alpha|OPT'| + (\alpha - 1)(2W' + n') + 2W' + O\left(\frac{W' \log N}{N}\right).$$

Using the fact that $\max\{W', n'\} \leq \varepsilon|OPT'|$ we can upper-bound this by

$$\alpha|OPT'| + O(\alpha\varepsilon)|OPT'| = \alpha(1 + O(\varepsilon))|OPT'|.$$

As noted earlier, we can transform this $\alpha(1 + O(\varepsilon))$-approximate solution to G' into a $\alpha(1 + O(\varepsilon))$-aproximate solution to G.

As for the running time, it is polynomial in the size of the No-Wait-Flowshop instance constructed. We have $O(Nn') = O(n^2/\varepsilon^2)$ jobs in this instance, and $O(W'n') + O(W' \log N) = O(n^2 \log n/\varepsilon^3)$ machines, so the running time is $poly(n, 1/\varepsilon)$. □

Using the result of the Karpinski et al. [8] for the ATSP we derive

Theorem 4. No-Wait-Flowshop *is not approximable with factor better than* $\frac{75}{74}$, *unless* $P = NP$.

3 An $O(\log m)$-Approximation Algorithm for No-Wait-Flowshop

Theorem 5. *There exists an $O(\log m)$-approximation algorithm for* No-Wait-Flowshop.

Proof. Consider any instance I of No-Wait-Flowshop. Let $G = G_I$ be the ATSP instance resulting from a standard reduction from No-Wait-Flowshop to ATSP, i.e. G is obtained by adding a dummy all-zero job to I and using δ as the distance function.

Our algorithm is a refinement of the approximation algorithm of Frieze, Galbiati and Maffioli [4]. This algorithm starts by finding a minimum cost cycle cover C_0 in G. Since OPT is a cycle cover we know that $|C_0| \leq |OPT|$. After that we choose a single vertex from each cycle – one that corresponds to the shortest job (i.e. we choose a vertex j with smallest $L(j)$). We then take G_1 to be the subgraph of G induced by the selected vertices. As was noted in [4] the optimal ATSP solution (OPT_1) for G_1 is at most as long as OPT, i.e. $|OPT_1| \leq |OPT|$.

We now reiterate the above procedure: We find a minimum cycle cover C_1 in G_1. We again have $|C_1| \leq |OPT_1|$. We choose a single vertex per cycle of C_1, again corresponding to the job with smallest length, define G_2 to be the subgraph of G_1 induced by the selected vertices, and so on. In each iteration we decrease the cardinality of the set of vertices by a factor of at least two. If the cycle cover C_i consists of a single cycle for some $i = 0, \ldots, \log_2 m - 1$, we consider a subgraph of G that is the union of all the C_k for $k = 0, \ldots, i$. This is an Eulerian subgraph of G of cost at most $\log_2 m \cdot |OPT|$, and can be transformed into a feasible Hamiltonian cycle by the standard procedure of short-cutting.

Otherwise, the graph $G_{\log_2 m}$ consists of more than one vertex. Let $C = \cup_{i=1}^{\log_2 m - 1} C_i$ be the subgraph of G that is the union of $\log_2 m$ cycle covers C_k for $k = 0, \ldots, \log_2 m - 1$. Consider the subgraph G' of G that is the union of C and an arbitrary Hamiltonian cycle H' in $G_{\log_2 m}$. Each connected component of C consists of at least m vertices. The vertex set of $G_{\log_2 m}$ consists of vertices corresponding to the shortest jobs in each of the connected components of C. Let S be this set of jobs. We now claim that the length of H' is at most

$$\sum_{j \in S} L(j) \leq \frac{1}{m} \sum_{j=1}^{n} L(j) \leq \frac{1}{m} m \cdot C^*_{max} = C^*_{max}.$$

The last inequality follows from the fact that sum of processing times of all operations that must be processed on a single machine is a lower bound on the value of the optimal makespan. It follows that the total weight of G' is at most $\log_2 m + 1$ times the optimal makespan. By shortcutting we can construct a Hamiltonian cycle in G, which in turn gives us an approximate solution for the original instance I of No-Wait-Flowshop. □

Acknowledgments. The first author would like to thank DIMAP, and in particular Artur Czumaj, for making his visit to the University of Warwick possible.

References

1. Asadpour, A., Goemans, M.X., Madry, A., Oveis Gharan, S., Saberi, A.: An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. In: SODA, pp. 379–389 (2010)
2. Bläser, M.: A new approximation algorithm for the asymmetric TSP with triangle inequality. ACM Transactions on Algorithms 4(4) (2008)
3. Feige, U., Singh, M.: Improved approximation ratios for traveling salesperson tours and paths in directed graphs. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX and RANDOM 2007. LNCS, vol. 4627, pp. 104–118. Springer, Heidelberg (2007)
4. Frieze, A.M., Galbiati, G., Maffioli, F.: On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. Networks 12(1), 23–39 (1982)
5. Gilmore, P., Gomory, R.: Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. Operations Research (12), 655–679 (1964)
6. Hall, N., Sriskandarajah, C.: A survey of machine scheduling problems with blocking and no-wait in process. Operations Research (44), 510–525 (1996)
7. Kaplan, H., Lewenstein, M., Shafrir, N., Sviridenko, M.: Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. Journal of the ACM 52(4), 602–626 (2005)
8. Karpinski, M., Lampis, M., Schmied, R.: New inapproximability bounds for TSP. CoRR abs/1303.6437 (2013)
9. Lawler, E., Lenstra, J., Rinnooy Kan, A., Shmoys, D.: Sequencing and scheduling: Algorithms and complexity. In: Graves, S., Rinnooy Kan, A., Zipkin, P. (eds.) Handbooks in Operations Research and Management Science. Logistics of Production and Inventory, vol. 4, pp. 445–522. North-Holland, Amsterdam (1993)
10. Papadimitriou, C.H., Kanellakis, P.C.: Flowshop scheduling with limited temporary storage. Journal of the ACM 27(3), 533–549 (1980)
11. Piehler, J.: Ein Beitrag zum Reihenfolgeproblem. Unternehmensforschung (4), 138–142 (1960)
12. Röck, H.: The three-machine no-wait flow shop is NP-complete. Journal of the ACM 31(2), 336–345 (1984)
13. Röck, H., Schmidt, G.: Machine aggregation heuristics in shop-scheduling. Methods of Operations Research (45), 303–314 (1983)
14. Spieksma, F.C.R., Woeginger, G.J.: The no-wait flow-shop paradox. Oper. Res. Lett. 33(6), 603–608 (2005)
15. Sviridenko, M.: Makespan minimization in no-wait flow shops: A polynomial time approximation scheme. SIAM Journal of Discrete Mathematics 16(2), 313–322 (2003)
16. Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms. Cambridge University Press (2011)
17. Wismer, D.A.: Solution of the flow shop scheduling problem with no intermediate queues. Operations Research (20), 689–697 (1972)

A Composition Theorem for the Fourier Entropy-Influence Conjecture

Ryan O'Donnell[1,⋆] and Li-Yang Tan[2,⋆⋆]

[1] Carnegie Mellon University
[2] Columbia University

Abstract. The Fourier Entropy-Influence (FEI) conjecture of Friedgut and Kalai [1] seeks to relate two fundamental measures of Boolean function complexity: it states that $\mathbf{H}[f] \leq C \cdot \mathbf{Inf}[f]$ holds for every Boolean function f, where $\mathbf{H}[f]$ denotes the spectral entropy of f, $\mathbf{Inf}[f]$ is its total influence, and $C > 0$ is a universal constant. Despite significant interest in the conjecture it has only been shown to hold for a few classes of Boolean functions.

Our main result is a composition theorem for the FEI conjecture. We show that if $g_1, \ldots, g_k$ are functions over disjoint sets of variables satisfying the conjecture, and if the Fourier transform of F taken with respect to the product distribution with biases $\mathbf{E}[g_1], \ldots, \mathbf{E}[g_k]$ satisfies the conjecture, then their composition $F(g_1(x^1), \ldots, g_k(x^k))$ satisfies the conjecture. As an application we show that the FEI conjecture holds for read-once formulas over arbitrary gates of bounded arity, extending a recent result [2] which proved it for read-once decision trees. Our techniques also yield an explicit function with the largest known ratio of $C \geq 6.278$ between $\mathbf{H}[f]$ and $\mathbf{Inf}[f]$, improving on the previous lower bound of 4.615.

1 Introduction

A longstanding and important open problem in the field of Analysis of Boolean Functions is the Fourier Entropy-Influence conjecture made by Ehud Friedgut and Gil Kalai in 1996 [1,3]. The conjecture seeks to relate two fundamental analytic measures of Boolean function complexity, the spectral entropy and total influence:

Fourier Entropy-Influence (FEI) Conjecture. *There exists a universal constant $C > 0$ such that for every Boolean function $f : \{-1,1\}^n \to \{-1,1\}$, it holds that $\mathbf{H}[f] \leq C \cdot \mathbf{Inf}[f]$. That is,*

⋆ Supported by NSF grants CCF-0747250 and CCF-1116594, and a Sloan fellowship. This material is based upon work supported by the National Science Foundation under grant numbers listed above. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

⋆⋆ Research done while visiting CMU.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 780–791, 2013.

$$\sum_{S\subseteq[n]} \widehat{f}(S)^2 \log_2\left(\frac{1}{\widehat{f}(S)^2}\right) \leq C \sum_{S\subseteq[n]} |S| \cdot \widehat{f}(S)^2.$$

Applying Parseval's identity to a Boolean function f we get $\sum_{S\subseteq[n]} \widehat{f}(S)^2 = \mathbf{E}[f(\boldsymbol{x})^2] = 1$, and so the Fourier coefficients of f induce a probability distribution $\mathcal{S}_f$ over the 2^n subsets of $[n]$ wherein $S \subseteq [n]$ has "weight" (probability mass) $\widehat{f}(S)^2$. The *spectral entropy* of f, denoted $\mathbf{H}[f]$, is the Shannon entropy of $\mathcal{S}_f$, quantifying how spread out the Fourier weight of f is across all 2^n monomials. The influence of a coordinate $i \in [n]$ on f is $\mathbf{Inf}_i[f] = \mathbf{Pr}[f(\boldsymbol{x}) \neq f(\boldsymbol{x}^{\oplus i})]$,[1] where $\boldsymbol{x}^{\oplus i}$ denotes $\boldsymbol{x}$ with its i-th bit flipped, and the *total influence* of f is simply $\mathbf{Inf}[f] = \sum_{i=1}^n \mathbf{Inf}_i[f]$. Straightforward Fourier-analytic calculations show that this combinatorial definition is equivalent to the quantity $\mathbf{E}_{\mathbf{S}\sim\mathcal{S}_f}[|\mathbf{S}|] = \sum_{S\subseteq[n]} |S| \cdot \widehat{f}(S)^2$, and so total influence measures the degree distribution of the monomials of f, weighted by the squared-magnitude of its coefficients. Roughly speaking then, the FEI conjecture states that a Boolean function whose Fourier weight is well "spread out" (*i.e.* has high spectral entropy) must have a significant portion of its Fourier weight lying on high degree monomials (*i.e.* have high total influence).[2]

In addition to being a natural question concerning the Fourier spectrum of Boolean functions, the FEI conjecture also has important connections to several areas of theoretical computer science and mathematics. Friedgut and Kalai's original motivation was to understand general conditions under which monotone graph properties exhibit sharp thresholds, and the FEI conjecture captures the intuition that having significant symmetry, hence high spectral entropy, is one such condition. Besides its applications in the study of random graphs, the FEI conjecture is known to imply the celebrated Kahn-Kalai-Linial theorem [4]:

KKL Theorem. *For every Boolean function f there exists an $i \in [n]$ such that* $\mathbf{Inf}_i[f] = \mathbf{Var}[f] \cdot \Omega(\frac{\log n}{n})$.

The FEI conjecture also implies Mansour's conjecture [5]:

Mansour's Conjecture. *Let f be a Boolean function computed by a t-term DNF formula. For any constant $\varepsilon > 0$ there exists a collection $\mathcal{S} \subseteq 2^{[n]}$ of cardinality* $\mathrm{poly}(t)$ *such that $\sum_{S\in\mathcal{S}} \widehat{f}(S)^2 \geq 1 - \varepsilon$.*

Combined with recent work of Gopalan *et al.* [6], Mansour's conjecture yields an efficient algorithm for agnostically learning the class of poly(n)-term DNF

[1] All probabilities and expectations are with respect to the uniform distribution unless otherwise stated.

[2] The assumption that f is Boolean-valued is crucial here, as the same conjecture is false for functions $f : \{-1,1\}^n \to \mathbb{R}$ satisfying $\sum_{S\subseteq[n]} \widehat{f}(S)^2 = 1$. The canonical counterexample is $f(x) = \frac{1}{\sqrt{n}} \sum_{i=1}^n x_i$ which has total influence 1 and spectral entropy $\log_2 n$.

formulas from queries. This would resolve a central open problem in computational learning theory [7]. De *et al.* also noted that sufficiently strong versions of Mansour's conjecture would yield improved pseudorandom generators for depth-2 AC^0 circuits [8]. More generally, the FEI conjecture implies the existence of sparse L_2-approximators for Boolean functions with small total influence:

Sparse L_2-approximators. Assume the FEI conjecture holds. Then for every Boolean function f there exists a $2^{O(\mathbf{Inf}[f]/\varepsilon)}$-sparse polynomial $p : \mathbb{R}^n \to \mathbb{R}$ such that $\mathbf{E}[(f(\boldsymbol{x}) - p(\boldsymbol{x}))^2] \leq \varepsilon$.

By Friedgut's junta theorem [9], the above holds unconditionally with a weaker bound of $2^{O(\mathbf{Inf}[f]^2/\varepsilon^2)}$. This is the main technical ingredient underlying several of the best known uniform-distribution learning algorithms [10,11].

For more on the FEI conjecture we refer the reader to Kalai's blog post [3].

1.1 Our Results

Our research is motivated by the following question:

Question 1. Let $F : \{-1,1\}^k \to \{-1,1\}$ and $g_1, \ldots, g_k : \{-1,1\}^\ell \to \{-1,1\}$. What properties do F and $g_1, \ldots, g_k$ have to satisfy for the FEI conjecture to hold for the disjoint composition $f(x^1, \ldots, x^k) = F(g_1(x^1), \ldots, g_k(x^k))$?

Despite its simplicity this question has not been well understood. For example, prior to our work the FEI conjecture was open even for read-once DNFs; these are the disjoint compositions of $F = \mathsf{OR}$ and $g_1, \ldots, g_k = \mathsf{AND}$, perhaps two of the most basic Boolean functions with extremely simple Fourier spectra. Indeed, Mansour's conjecture, a weaker conjecture than FEI, was only recently shown to hold for read-once DNFs [12,8]. Besides being a fundamental question concerning the behavior of spectral entropy and total influence under composition, Question 1 (and our answer to it) also has implications for a natural approach towards disproving the FEI conjecture; we elaborate on this at the end of this section.

A particularly appealing and general answer to Question 1 that one may hope for would be the following: "if $\mathbf{H}[F] \leq C_1 \cdot \mathbf{Inf}[F]$ and $\mathbf{H}[g_i] \leq C_2 \cdot \mathbf{Inf}[g_i]$ for all $i \in [k]$, then $\mathbf{H}[f] \leq \max\{C_1, C_2\} \cdot \mathbf{Inf}[f]$." While this is easily seen to be false[3], our main result shows that this proposed answer to Question 1 *is* in fact true for a carefully chosen sharpening of the FEI conjecture. To arrive at a formulation that bootstraps itself, we first consider a slight strengthening of the FEI conjecture which we call FEI$^+$, and then work with a generalization of FEI$^+$ that concerns the Fourier spectrum of f not just with respect to the uniform distribution, but an arbitrary product distribution over $\{-1,1\}^n$:

Conjecture 1 (FEI$^+$ for product distributions). There is a universal constant $C > 0$ such that the following holds. Let $\mu = \langle \mu_1, \ldots, \mu_n \rangle$ be any sequence of biases and $f : \{-1,1\}^n_\mu \to \{-1,1\}$. Here the notation $\{-1,1\}^n_\mu$ means that

[3] For example, by considering $F = \mathsf{OR}_2$, the 2-bit disjunction, and $g_1, g_2 = \mathsf{AND}_2$, the 2-bit conjunction.

we think of $\{-1,1\}^n$ as being endowed with the μ-biased product probability distribution in which $\mathbf{E}_\mu[x_i] = \mu_i$ for all $i \in [n]$. Let $\{\widetilde{f}(S)\}_{S\subseteq[n]}$ be the μ-biased Fourier coefficients of f. Then

$$\sum_{S\neq\emptyset} \widetilde{f}(S)^2 \log\left(\frac{\prod_{i\in S}(1-\mu_i^2)}{\widetilde{f}(S)^2}\right) \leq C\cdot(\mathbf{Inf}^\mu[f] - \mathop{\mathbf{Var}}_{\mu}[f]).$$

We write $\mathbf{H}^\mu[f]$ to denote the quantity $\sum_{S\subseteq[n]} \widetilde{f}(S)^2 \log\left(\prod_{i\in S}(1-\mu_i^2)/\widetilde{f}(S)^2\right)$, and so the inequality of Conjecture 1 can be equivalently stated as $\mathbf{H}^\mu[f^{\geq 1}] \leq C\cdot(\mathbf{Inf}^\mu[f] - \mathbf{Var}_\mu[f])$.

In Proposition 1 we show that Conjecture 1 with $\mu = \langle 0,\ldots,0\rangle$ (the uniform distribution) implies the FEI conjecture. We say that a Boolean function f "satisfies μ-biased FEI$^+$ with factor C" if the μ-biased Fourier transform of f satisfies the inequality of Conjecture 1. Our main result, which we prove in Section 3, is a composition theorem for FEI$^+$:

Theorem 1. *Let $f(x^1,\ldots,x^k) = F(g_1(x^1),\ldots,g_k(x^k))$, where the domain of f is endowed with a product distribution μ. Suppose $g_1,\ldots,g_k$ satisfy μ-biased FEI$^+$ with factor C_1 and F satisfies η-biased FEI$^+$ with factor C_2, where $\eta = \langle \mathbf{E}_\mu[g_1],\ldots,\mathbf{E}_\mu[g_k]\rangle$. Then f satisfies μ-biased FEI$^+$ with factor* $\max\{C_1, C_2\}$.

Theorem 1 suggests an inductive approach towards proving the FEI conjecture for read-once de Morgan formulas: since the dictators $\pm x_i$ trivially satisfy uniform-distribution FEI$^+$ with factor 1, it suffices to prove that both AND_2 and OR_2 satisfy μ-biased FEI$^+$ with some constant *independent of* $\mu \in [-1,1]^2$. In Section 4 we prove that in fact *every* $F : \{-1,1\}^k \to \{-1,1\}$ satisfies μ-biased FEI$^+$ with a factor depending only on its arity k and not the biases $\mu_1,\ldots,\mu_k$.

Theorem 2. *Every $F : \{-1,1\}^k \to \{-1,1\}$ satisfies μ-biased FEI$^+$ with factor $C = 2^{O(k)}$ for any product distribution $\mu = \langle \mu_1,\ldots,\mu_k\rangle$.*

Together, Theorems 1 and 2 imply:

Theorem 3. *Let f be computed by a read-once formula over the basis $\mathcal{B}$ and μ be any sequences of biases. Then f satisfies μ-biased FEI$^+$ with factor C, where C depends only on the arity of the gates in $\mathcal{B}$.*

Since uniform-distribution FEI$^+$ is a strengthening of the FEI conjecture, Theorem 3 implies that the FEI conjecture holds for read-once formulas over arbitrary gates of bounded arity. As mentioned above, prior to our work the FEI conjecture was open even for the class of read-once DNFs, a small subclass of read-once formulas over the de Morgan basis $\{\mathsf{AND}_2, \mathsf{OR}_2, \mathsf{NOT}\}$ of arity 2. Read-once formulas over a rich basis $\mathcal{B}$ are a natural generalization of read-once de Morgan formulas, and have seen previous study in concrete complexity (see *e.g.* [13]).

Improved Lower Bound on the FEI Constant. Iterated disjoint composition is commonly used to achieve separations between complexity measures for Boolean functions, and represents a natural approach towards disproving the FEI conjecture. For example, one may seek a function F such that iterated compositions of F with itself achieves a super-constant amplification of the ratio between $\mathbf{H}[F]$ and $\mathbf{Inf}[F]$, or consider variants such as iterating F with a different combining function G. Theorem 3 rules out as potential counterexamples all such constructions based on iterated composition.

However, the tools we develop to prove Theorem 3 also yield an explicit function f achieving the best-known separation between $\mathbf{H}[f]$ and $\mathbf{Inf}[f]$ (*i.e.* the constant C in the statement of the FEI conjecture). In Section 5 we prove:

Theorem 4. *There exists an explicit family of functions* $f_n : \{-1,1\}^n \to \{-1,1\}$ *such that*

$$\lim_{n\to\infty} \frac{\mathbf{H}[f_n]}{\mathbf{Inf}[f_n]} \geq 6.278.$$

This improves on the previous lower bound of $C \geq 60/13 \approx 4.615$ [2].

Previous Work. The first published progress on the FEI conjecture was by Klivans *et al.* who proved the conjecture for random poly(n)-term DNF formulas [12]. This was followed by the work of O'Donnell *et al.* who proved the conjecture for the class of symmetric functions and read-once decision trees [2].

The FEI conjecture for product distributions was studied in the recent work of Keller *et al.* [14], where they consider the case of all the biases being the same. They introduce the following generalization of the FEI conjecture to these measures, and show via a reduction to the uniform distribution that it is equivalent to the FEI conjecture:

Conjecture 2 (Keller-Mossel-Schlank). There is a universal constant C such that the following holds. Let $0 < p < 1$ and $f : \{-1,1\}^n \to \{-1,1\}$, where the domain of f is endowed with the product distribution where $\mathbf{Pr}[x_i = -1] = p$ for all $i \in [n]$. Let $\{\widetilde{f}(S)\}_{S\subseteq[n]}$ be the Fourier coefficients of f with respect to this distribution. Then

$$\sum_{S\subseteq[n]} \widetilde{f}(S)^2 \log_2\left(\frac{1}{\widetilde{f}(S)^2}\right) \leq C \cdot \frac{\log(1/p)}{1-p} \sum_{S\subseteq[n]} |S| \cdot \widetilde{f}(S)^2.$$

Notice that in this conjecture, the constant on the right-hand side, $C \cdot \frac{\log(1/p)}{1-p}$, depends on p. By way of contrast, in our Conjecture 1 the right-hand side constant has no dependence on p; instead, the dependence on the biases is built into the definition of spectral entropy. We view our generalization of the FEI conjecture to arbitrary product distributions (where the biases are not necessarily identical) as a key contribution of this work, and point to our composition theorem as evidence in favor of Conjecture 1 being a good statement to work with.

2 Preliminaries

Notation. We will be concerned with functions $f : \{-1,1\}^n_\mu \to \mathbb{R}$ where $\mu = \langle \mu_1, \ldots, \mu_n \rangle \in [0,1]^n$ is a sequence of biases. Here the notation $\{-1,1\}^n_\mu$ means that we think of $\{-1,1\}^n$ as being endowed with the μ-biased product probability distribution in which $\mathbf{E}_\mu[x_i] = \mu_i$ for all $i \in [n]$. We write σ_i^2 to denote variance of the i-th coordinate $\mathbf{Var}_\mu[x_i] = 1 - \mu_i^2$, and $\varphi : \mathbb{R} \to \mathbb{R}$ as shorthand for the function $t \mapsto t^2 \log_2(1/t^2)$, adopting the convention that $\varphi(0) = 0$. All logarithms in this paper are in base 2 unless otherwise stated.

Definition 1 (Fourier expansion). *Let $\mu = \langle \mu_1, \ldots, \mu_n \rangle$ be a sequence of biases. The μ-biased Fourier expansion of $f : \{-1,1\}^n \to \mathbb{R}$ is*

$$f(x) = \sum_{S \subseteq [n]} \widetilde{f}(S)\phi_S^\mu(x), \quad \text{where } \phi_S^\mu(x) = \prod_{i \in S} \frac{x_i - \mu_i}{\sigma_i} \text{ and } \widetilde{f}(S) = \mathop{\mathbf{E}}_{\mu}[f(\boldsymbol{x})\phi_S^\mu(\boldsymbol{x})].$$

The μ-biased spectral support of f is the collection $\mathcal{S} \subseteq 2^{[n]}$ of subsets $S \subseteq [n]$ such that $\widetilde{f}(S) \neq 0$. We write $f^{\geq k}$ to denote $\sum_{|S| \geq k} \widetilde{f}(S)\phi_S^\mu(x)$, the projection of f onto its monomials of degree at least k.

Definition 2 (Influence). *Let $f : \{-1,1\}^n_\mu \to \mathbb{R}$. The influence of variable $i \in [n]$ on f is $\mathbf{Inf}_i^\mu[f] = \mathbf{E}_\rho[\mathbf{Var}_{\mu_i}[f_\rho]]$, where ρ is a μ-biased random restriction to the coordinates in $[n] \backslash \{i\}$. The total influence of f, denoted $\mathbf{Inf}^\mu[f]$, is $\sum_{i=1}^n \mathbf{Inf}_i^\mu[f]$.*

We recall a few basic Fourier formulas. The expectation of f is given by $\mathbf{E}_\mu[f] = \widetilde{f}(\emptyset)$ and its variance $\mathbf{Var}_\mu[f] = \sum_{S \neq \emptyset} \widetilde{f}(S)^2$. For each $i \in [n]$, $\mathbf{Inf}_i^\mu[f] = \sum_{S \ni i} \widetilde{f}(S)^2$ and so $\mathbf{Inf}^\mu[f] = \sum_{S \subseteq [n]} |S| \cdot \widetilde{f}(S)^2$. We omit the sub- and superscripts when $\mu = \langle 0, \ldots, 0 \rangle$ is the uniform distribution.

Recall that the i-th discrete derivative operator for $f : \{-1,1\}^n \to \{-1,1\}$ is defined to be $D_{x_i}(x) = (f(x^{i \leftarrow 1}) - f(x^{i \leftarrow -1}))/2$, and for $S \subseteq [n]$ we write $D_{x^S} f$ to denote $\circ_{i \in S} D_{x_i} f$.

Definition 3 (Discrete derivative). *The i-th discrete derivative operator $D_{\phi_i^\mu}$ with respect to the μ-biased product distribution on $\{-1,1\}^n$ is defined by $D_{\phi_i^\mu} f(x) = \sigma_i D_{x_i} f(x)$.*

With respect to the μ-biased Fourier expansion of $f : \{-1,1\}^n_\mu \to \mathbb{R}$ the operator $D_{\phi_i^\mu}$ satisfies $D_{\phi_i^\mu} f = \sum_{S \ni i} \widetilde{f}(S)\phi^\mu_{S \backslash \{i\}}$, and so for any $S \subseteq [n]$ we have $\widetilde{f}(S) = \mathbf{E}[\circ_{i \in S} D_{\phi_i^\mu} f] = \prod_{i \in S} \sigma_i \, \mathbf{E}_\mu[D_{x^S} f]$. The next proposition is a simple consequence of the Poincaré inequality $\mathbf{Inf}[f] \geq \mathbf{Var}[f]$, and we defer its proof to the full version of this paper.

Proposition 1 (FEI$^+$ implies FEI). *Suppose f satisfies uniform-distribution FEI$^+$ with factor C. Then f satisfies the FEI conjecture with factor* $\max\{C, 1/\ln 2\}$.

3 Composition Theorem for FEI$^+$

We will be concerned with compositions of functions $f = F(g_1(x^1), \dots, g_k(x^k))$ where $g_1, \dots, g_k$ are over disjoint sets of variables each of size ℓ. The domain of each g_i is endowed with a product distribution $\mu^i = \langle \mu^i_1, \dots, \mu^i_\ell \rangle$, which induces an overall product distribution $\mu = \langle \mu^1_1, \dots, \mu^1_\ell, \dots, \mu^k_1, \dots, \mu^k_\ell \rangle$ over the domain of $f : \{-1,1\}^{k\ell} \to \{-1,1\}$. For notational clarity we will adopt the equivalent view of $g_1, \dots, g_k$ as functions over the same domain $\{-1,1\}^{k\ell}_\mu$ endowed with the same product distribution μ, with each g_i depending only on ℓ out of $k\ell$ variables.

Our first lemma gives formulas for the spectral entropy and total influence of the product of functions $\Phi_1, \dots, \Phi_k$ over disjoint sets of variables. The lemma holds for real-valued functions Φ_i; we require this level of generality as we will not be applying the lemma directly to the Boolean-valued functions $g_1, \dots, g_k$ in the composition $F(g_1(x^1), \dots, g_k(x^k))$, but instead to their normalized variants $\Phi(g_i) = (g_i - \mathbf{E}[g_i])/\mathbf{Var}[g_i]^{1/2}$.

Lemma 1. *Let $\Phi_1, \dots, \Phi_k : \{-1,1\}^{k\ell}_\mu \to \mathbb{R}$ where each Φ_i depends only on the ℓ coordinates in $\{(i-1)\ell + 1, \dots, i\ell\}$. Then*

$$\mathbf{H}^\mu[\Phi_1 \cdots \Phi_k] = \sum_{i=1}^k \mathbf{H}^\mu[\Phi_i] \prod_{j \neq i} \mathop{\mathbf{E}}_\mu[\Phi_j^2] \text{ and } \mathbf{Inf}^\mu[\Phi_1 \cdots \Phi_k] = \sum_{i=1}^k \mathbf{Inf}^\mu[\Phi_i] \prod_{j \neq i} \mathop{\mathbf{E}}_\mu[\Phi_j^2].$$

Due to space considerations we defer the proof of Lemma 1 to the full version of our paper. We note that this lemma recovers as a special case the folklore observation that the FEI conjecture "tensorizes": for any f if we define $f^{\oplus k}(x^1, \dots, x^k) = f(x^1) \cdots f(x^k)$ then $\mathbf{H}[f^{\oplus k}] = k \cdot \mathbf{H}[f]$ and $\mathbf{Inf}[f^{\oplus k}] = k \cdot \mathbf{Inf}[f]$. Therefore $\mathbf{H}[f] \leq C \cdot \mathbf{Inf}[f]$ if and only if $\mathbf{H}[f^{\oplus k}] \leq C \cdot \mathbf{Inf}[f^{\oplus k}]$.

Our next proposition relates the basic analytic measures – spectral entropy, total influence, and variance – of a composition $f = F(g_1(x^1), \dots, g_k(x^k))$ to the corresponding quantities of the combining function F and base functions $g_1, \dots, g_k$. As alluded to above, we accomplish this by considering f as a linear combination of the normalized functions $\Phi(g_i) = (g_i - \mathbf{E}[g_i])/\mathbf{Var}[g_i]^{1/2}$ and applying Lemma 1 to each term in the sum. We mention that this proposition is also the crux of our new lower bound of $C \geq 6.278$ on the constant of the FEI conjecture, which we present in Section 5.

Proposition 2. *Let $F : \{-1,1\}^k \to \mathbb{R}$, and $g_1, \dots, g_k : \{-1,1\}^{k\ell}_\mu \to \{-1,1\}$ where each g_i depends only on the ℓ coordinates in $\{(i-1)\ell + 1, \dots, i\ell\}$.*

Let $f(x) = F(g_1(x), \ldots, g_k(x))$ *and* $\{\widetilde{F}(S)\}_{S\subseteq[k]}$ *be the* η*-biased Fourier coefficients of* F *where* $\eta = \langle \mathbf{E}_\mu[g_1], \ldots, \mathbf{E}_\mu[g_k]\rangle$*. Then*

$$\mathbf{H}^\mu[f^{\geq 1}] = \mathbf{H}^\eta[F^{\geq 1}] + \sum_{S\neq\emptyset} \widetilde{F}(S)^2 \sum_{i\in S} \frac{\mathbf{H}^\mu[g_i^{\geq 1}]}{\mathbf{Var}_\mu[g_i]}, \tag{1}$$

$$\mathbf{Inf}^\mu[f] = \sum_{S\neq\emptyset} \widetilde{F}(S)^2 \sum_{i\in S} \frac{\mathbf{Inf}^\mu[g_i]}{\mathbf{Var}_\mu[g_i]}, \quad \textit{and} \tag{2}$$

$$\mathbf{Var}_\mu[f] = \sum_{S\neq\emptyset} \widetilde{F}(S)^2 = \mathbf{Var}_\eta[F]. \tag{3}$$

Proof. By the η-biased Fourier expansion of $F : \{-1,1\}_\eta^k \to \mathbb{R}$ and the definition of η we have

$$F(y_1, \ldots, y_k) = \sum_{S\subseteq[k]} \widetilde{F}(S) \prod_{i\in S} \frac{y_i - \eta_i}{\sqrt{1-\eta_i^2}} = \sum_{S\subseteq[k]} \widetilde{F}(S) \prod_{i\in S} \frac{y_i - \mathbf{E}_\mu[g_i]}{\mathbf{Var}_\mu[g_i]^{1/2}},$$

and so $F(g_1(x), \ldots, g_k(x)) = \sum_{S\subseteq[k]} \widetilde{F}(S) \prod_{i\in S} \Phi(g_i(x))$, where $\Phi(g_i(x)) = (g_i(x) - \mathbf{E}_\mu[g_i])/\mathbf{Var}_\mu[g_i]^{1/2}$. Note that Φ normalizes g_i such that $\mathbf{E}_\mu[\Phi(g_i)] = 0$ and $\mathbf{E}_\mu[\Phi(g_i)^2] = 1$. First we claim that

$$\mathbf{H}^\mu[f^{\geq 1}] = \mathbf{H}^\mu\Big[\sum_{S\neq\emptyset} \widetilde{F}(S) \prod_{i\in S} \Phi(g_i)\Big] = \sum_{S\neq\emptyset} \mathbf{H}^\mu\Big[\widetilde{F}(S) \prod_{i\in S} \Phi(g_i)\Big].$$

It suffices to show that for any two distinct non-empty sets $S, T \subseteq [k]$, no monomial ϕ_U^μ occurs in the μ-biased spectral support of both $\widetilde{F}(S)\prod_{i\in S}\Phi(g_i)$ and $\widetilde{F}(T)\prod_{i\in T}\Phi(g_i)$. To see this recall that $\Phi(g_i)$ is balanced with respect to μ (*i.e.* $\mathbf{E}_\mu[\Phi(g_i)] = \mathbf{E}_\mu[\Phi(g_i)\phi_\emptyset^\mu] = 0$), and so every monomial ϕ_U^μ in the support of $\widetilde{F}(S)\prod_{i\in S}\Phi(g_i)$ is of the form $\prod_{i\in S}\phi_{U_i}^\mu$ where U_i is a non-empty subset of the relevant variables of g_i (*i.e.* $\{(i-1)\ell+1, \ldots, i\ell\}$); likewise for monomials in the support of $\widetilde{F}(T)\prod_{i\in T}\Phi(g_i)$. In other words the non-empty subsets of $[k]$ induce a partition of the μ-biased Fourier support of f, where ϕ_U^μ is mapped to $\emptyset \neq S \subseteq [k]$ if and only if U contains a relevant variable of g_i for every $i \in S$ and none of the relevant variables of g_j for any $j \notin S$.

With this identity in hand we have

$$\begin{aligned}
\mathbf{H}^\mu[f^{\geq 1}] &= \sum_{S\neq\emptyset} \mathbf{H}^\mu\Big[\widetilde{F}(S)\prod_{i\in S}\Phi(g_i)\Big] \\
&= \sum_{S\neq\emptyset} \varphi(\widetilde{F}(S)) + \widetilde{F}(S)^2 \sum_{i\in S} \mathbf{H}^\mu[\Phi(g_i)]. \\
&= \sum_{S\neq\emptyset} \varphi(\widetilde{F}(S)) + \widetilde{F}(S)^2 \sum_{i\in S} \left(\frac{\mathbf{H}^\mu[g_i - \mathbf{E}_\mu[g_i]]}{\mathbf{Var}_\mu[g_i]} + \varphi\left(\frac{1}{\mathbf{Var}_\mu[g_i]^{1/2}}\right) \mathop{\mathbf{Var}}_{\mu}[g_i] \right) \\
&= \mathbf{H}^\eta[F^{\geq 1}] + \sum_{S\neq\emptyset} \widetilde{F}(S)^2 \sum_{i\in S} \frac{\mathbf{H}^\mu[g_i^{\geq 1}]}{\mathbf{Var}_\mu[g_i]},
\end{aligned}$$

where the second and third equalities are two applications of Lemma 1 (for the second equality we view $\widetilde{F}(S)$ as a constant function with $\mathbf{H}^{\mu}[\widetilde{F}(S)] = \varphi(\widetilde{F}(S))$). By the same reasoning, we also have

$$\begin{aligned}\mathbf{Inf}^{\mu}[f] \;&=\; \sum_{S\neq\emptyset} \mathbf{Inf}^{\mu}\Big[\widetilde{F}(S)\prod_{i\in S}\Phi(g_i(x^i))\Big] = \sum_{S\neq\emptyset}\widetilde{F}(S)^2\sum_{i\in S}\mathbf{Inf}^{\mu}[\Phi(g_i)]\\ &= \sum_{S\neq\emptyset}\widetilde{F}(S)^2\sum_{i\in S}\frac{\mathbf{Inf}^{\mu}[g_i]}{\mathbf{Var}_{\mu}[g_i]}.\end{aligned}$$

Here the second equality is by Lemma 1, again viewing $\widetilde{F}(S)$ as a constant function with $\mathbf{Inf}^{\mu}[\widetilde{F}(S)] = 0$, and the third equality uses the fact that $\mathbf{Inf}^{\mu}[\alpha f] = \alpha^2 \cdot \mathbf{Inf}^{\mu}[f]$ and $\mathbf{Inf}^{\mu}[g_i - \mathbf{E}_{\mu}[g_i]] = \mathbf{Inf}^{\mu}[g_i]$. Finally we see that

$$\mathbf{Var}_{\mu}[f] = \sum_{S\neq\emptyset}\mathbf{Var}_{\mu}\Big[\widetilde{F}(S)\prod_{i\in S}\Phi(g_i)\Big] = \sum_{S\neq\emptyset}\widetilde{F}(S)^2\prod_{i\in S}\mathbf{Var}_{\mu}[\Phi(g_i)] = \sum_{S\neq\emptyset}\widetilde{F}(S)^2,$$

where the last quantity is $\mathbf{Var}_{\eta}[F]$. Here the second equality uses the fact that the functions $\Phi(g_i)$ are on disjoint sets of variables (and therefore statistically independent when viewed as random variables), and the third equality holds since $\mathbf{Var}_{\mu}[\Phi(g_i)] = \mathbf{E}[\Phi(g_i)^2] - \mathbf{E}[\Phi(g_i)]^2 = 1$. □

We are now ready to prove our main theorem:

Theorem 1. *Let $F : \{-1,1\}^k \to \mathbb{R}$, and $g_1, \ldots, g_k : \{-1,1\}^{k\ell}_{\mu} \to \{-1,1\}$ where each g_i depends only on the ℓ coordinates in $\{(i-1)\ell+1, \ldots, i\ell\}$. Let $f(x) = F(g_1(x), \ldots, g_k(x))$ and suppose $C > 0$ satisfies*

1. $\mathbf{H}^{\mu}[g_i^{\geq 1}] \leq C \cdot (\mathbf{Inf}^{\mu}[g_i] - \mathbf{Var}_{\mu}[g_i])$ *for all* $i \in [k]$.
2. $\mathbf{H}^{\eta}[F^{\geq 1}] \leq C \cdot (\mathbf{Inf}^{\eta}[F] - \mathbf{Var}_{\eta}[F])$, *where* $\eta = \langle \mathbf{E}_{\mu}[g_1], \ldots, \mathbf{E}_{\mu}[g_k]\rangle$.

Then $\mathbf{H}^{\mu}[f^{\geq 1}] \leq C \cdot (\mathbf{Inf}^{\mu}[f] - \mathbf{Var}_{\mu}[f])$.

Proof. By our first assumption each g_i satisfies $\mathbf{Inf}^{\mu}[g_i] \geq \frac{1}{C}\mathbf{H}^{\mu}[g^{\geq 1}] + \mathbf{Var}_{\mu}[g_i]$, and so combining this with equation (2) of Proposition 2 we have

$$\begin{aligned}\mathbf{Inf}^{\mu}[f] \;&=\; \sum_{S\neq\emptyset}\widetilde{F}(S)^2\sum_{i\in S}\frac{\mathbf{Inf}^{\mu}[g_i]}{\mathbf{Var}_{\mu}[g_i]} \geq \sum_{S\neq\emptyset}\widetilde{F}(S)^2\sum_{i\in S}\left(\frac{\mathbf{H}^{\mu}[g_i^{\geq 1}]}{C\,\mathbf{Var}_{\mu}[g_i]} + 1\right)\\ &= \mathbf{Inf}^{\eta}[F] + \frac{1}{C}\sum_{S\neq\emptyset}\widetilde{F}(S)^2\sum_{i\in S}\frac{\mathbf{H}^{\mu}[g_i^{\geq 1}]}{\mathbf{Var}_{\mu}[g_i]}. \quad (4)\end{aligned}$$

This along with equations (1) and (3) of Proposition 2 completes the proof:

$$\begin{aligned}\mathbf{H}^{\mu}[f^{\geq 1}] &= \mathbf{H}^{\eta}[F^{\geq 1}] + \sum_{S\neq\emptyset}\widetilde{F}(S)^2\sum_{i\in S}\frac{\mathbf{H}^{\mu}[g_i^{\geq 1}]}{\mathbf{Var}_{\mu}[g_i]}\\ &\leq C\cdot(\mathbf{Inf}^{\eta}[F] - \mathbf{Var}_{\eta}[F]) + \sum_{S\neq\emptyset}\widetilde{F}(S)^2\sum_{i\in S}\frac{\mathbf{H}^{\mu}[g_i^{\geq 1}]}{\mathbf{Var}_{\mu}[g_i]}\\ &\leq C\cdot(\mathbf{Inf}^{\mu}[f] - \mathbf{Var}_{\eta}[F]) \;=\; C\cdot(\mathbf{Inf}^{\mu}[f] - \mathbf{Var}_{\mu}[f]).\end{aligned}$$

Here the first equality is by (1), the first inequality by our second assumption, the second inequality by (4), and finally the last identity by (3). □

4 Distribution-Independent Bound for FEI$^+$

In this section we prove that μ-biased FEI$^+$ holds for all Boolean functions $F : \{-1,1\}^k_\mu \to \{-1,1\}$ with factor C independent of the biases $\mu_1, \ldots, \mu_k$ of μ. When $\mu = \langle 0, \ldots 0\rangle$ is the uniform distribution it is well-known that the FEI conjecture holds with factor $C = O(\log k)$, and a bound of $C \leq 2^k$ is trivial since $\mathbf{Inf}[F]$ is always an integer multiple of 2^{-k} and $\mathbf{H}[F] \leq 1$; neither proofs carry through to the setting of product distributions. We remark that even verifying the seemingly simple claim "there exists a universal constant C such that $\mathbf{H}^\mu[\mathsf{MAJ}_3] \leq C \cdot (\mathbf{Inf}^\mu[\mathsf{MAJ}_3] - \mathbf{Var}_\mu[\mathsf{MAJ}_3])$ for all product distributions $\mu \in [0,1]^3$", where MAJ_3 the majority function over 3 variables, turns out to be technically cumbersome.

The high-level strategy is to bound each of the $2^k - 1$ terms of $\mathbf{H}^\mu[F^{\geq 1}]$ separately; due to space considerations we defer the proof the main lemma to the full version of our paper.

Lemma 2. *Let $F : \{-1,1\}^k_\mu \to \{-1,1\}$. Let $S \subseteq [k]$, $S \neq \emptyset$, and suppose $\widetilde{F}(S) \neq 0$. For any $j \in S$ we have*

$$\widetilde{F}(S)^2 \log\left(\frac{\prod_{i\in S}\sigma_i^2}{\widetilde{F}(S)^2}\right) \leq \frac{2^{2k}}{\ln 2} \cdot \mathop{\mathbf{Var}}_{\mu}[D_{\phi_j^\mu} F].$$

Theorem 2. *Let $F : \{-1,1\}^k_\mu \to \{-1,1\}$. Then $\mathbf{H}^\mu[F^{\geq 1}] \leq 2^{O(k)} \cdot (\mathbf{Inf}^\mu[F] - \mathbf{Var}_\mu[F])$.*

Proof. The claim can be equivalently stated as $\mathbf{H}^\mu[F^{\geq 1}] \leq 2^{O(k)} \sum_{i=1}^n \mathbf{Var}_\mu[D_{\phi_i^\mu} F]$, since

$$\sum_{i=1}^n \mathbf{Var}[D_{\phi_i^\mu} F] = \sum_{|S|\geq 2} |S| \cdot \widetilde{F}(S)^2 \leq 2 \sum_{|S|\geq 2} (|S|-1) \cdot \widetilde{F}(S)^2 = 2 \cdot (\mathbf{Inf}^\mu[F] - \mathbf{Var}_\mu[F]).$$

By Lemma 2, the contribution of each $S \neq \emptyset$ to $\mathbf{H}^\mu[F^{\geq 1}]$ is $2^{O(k)} \mathbf{Var}_\mu[D_{\phi_j^\mu} F]$, where j is any element of S. Summing over all $2^k - 1$ non-empty subsets S of $[k]$ completes the proof. □

4.1 FEI$^+$ for Read-Once Formulas

Finally, we combine our two main results so far, the composition theorem (Theorem 1) and the distribution-independent universal bound (Theorem 2), to prove Conjecture 1 for read-once formulas with arbitrary gates of bounded arity.

Definition 4. *Let $\mathcal{B}$ be a set of Boolean functions. We say that a Boolean function f is a formula over the basis $\mathcal{B}$ if f is computable by a formula with gates belonging to $\mathcal{B}$. We say that f is a read-once formula over $\mathcal{B}$ if every variable appears at most once in the formula for f.*

Corollary 1. *Let $C > 0$ and $\mathcal{B}$ be a set of Boolean functions, and suppose $\mathbf{H}^\mu[F] \le C \cdot (\mathbf{Inf}^\mu[F] - \mathbf{Var}_\mu[F])$ for all $F \in \mathcal{B}$ and product distributions μ. Let $\mathcal{C}$ be the class of read-once formulas over the basis $\mathcal{B}$. Then $\mathbf{H}^\mu[f] \le C \cdot (\mathbf{Inf}^\mu[f] - \mathbf{Var}_\mu[f])$ for all $f \in \mathcal{C}$ and product distributions μ.*

Proof. We proceed by structural induction on the formula computing f. The base case holds since the μ-biased Fourier expansion of the dictator x_1 and anti-dictator $-x_1$ is $\pm(\mu_1 + \sigma_1\phi_1^\mu(x))$ and so $\mathbf{H}^\mu[f^{\ge 1}] = \widetilde{f}(\{1\})^2 \log(\sigma_1^2/\widetilde{f}(\{1\})^2) = \sigma_1^2 \log(\sigma_1^2/\sigma_1^2) = 0$.

For the inductive step, suppose $f = F(g_1, \dots, g_k)$, where $F \in \mathcal{B}$ and $g_1, \dots, g_k$ are read-once formulas over $\mathcal{B}$ over disjoint sets of variables. Let μ be any product distribution over the domain of f. By our induction hypothesis we have $\mathbf{H}^\mu[g_i^{\ge 1}] \le C \cdot (\mathbf{Inf}^\mu[g_i] - \mathbf{Var}_\mu[g_i])$ for all $i \in [k]$, satisfying the first requirement of Theorem 1. Next, by our assumption on $F \in \mathcal{B}$, we have $\mathbf{H}^\eta[F^{\ge 1}] \le C \cdot (\mathbf{Inf}^\eta[F] - \mathbf{Var}_\eta[F])$ for all product distributions η, and in particular, $\eta = \langle \mathbf{E}_\mu[g_1], \dots, \mathbf{E}_\mu[g_k] \rangle$, satisfying the second requirement of Theorem 1. Therefore, by Theorem 1 we conclude that $\mathbf{H}^\mu[f] \le C \cdot (\mathbf{Inf}^\mu[f] - \mathbf{Var}_\mu[f])$. □

By Theorem 2, for any set $\mathcal{B}$ of Boolean functions with maximum arity k and product distribution μ, every $F \in \mathcal{B}$ satisfies $\mathbf{H}^\mu[F] \le 2^{O(k)} \cdot (\mathbf{Inf}^\mu[F] - \mathbf{Var}_\mu[F])$. Combining this with Corollary 1 yields the following:

Theorem 3. *Let $\mathcal{B}$ be a set of Boolean functions with maximum arity k, and $\mathcal{C}$ be the class of read-once formulas over the basis $\mathcal{B}$. Then $\mathbf{H}^\mu[f] \le 2^{O(k)} \cdot (\mathbf{Inf}^\mu[f] - \mathbf{Var}_\mu[f])$ for all $f \in \mathcal{C}$ and product distributions μ.*

5 Lower Bound on the Constant of the FEI Conjecture

The tools we develop in this paper also yield an explicit function f achieving the best-known ratio between $\mathbf{H}[f]$ and $\mathbf{Inf}[f]$ (*i.e.* a lower bound on the constant C in the FEI conjecture). We will use the following special case of Proposition 2 on the behavior of spectral entropy and total influence under composition:

Lemma 3 (Amplification lemma). *Let $F : \{-1,1\}^k \to \{-1,1\}$ be a balanced Boolean function. Let $f_0 = F$, and define $f_m = F(f_{m-1}(x^1), \dots, f_{m-1}(x^k))$ for all $m \ge 1$. Then*

$$\frac{\mathbf{H}[f_m]}{\mathbf{Inf}[f_m]} = \frac{\mathbf{H}[F]}{\mathbf{Inf}[F]} + \frac{\mathbf{H}[F]}{\mathbf{Inf}[F](\mathbf{Inf}[F]-1)} - \frac{\mathbf{H}[F]}{\mathbf{Inf}[F]^{m+1}(\mathbf{Inf}[F]-1)}.$$

Theorem 4. *There exists an infinite family of functions $f_m : \{-1,1\}^{6^m} \to \{-1,1\}$ such that $\lim_{m\to\infty} \mathbf{H}[f_m]/\mathbf{Inf}[f_m] \ge 6.278944$.*

Proof. Let

$$g = (\overline{x}_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x}_2 \wedge x_4) \vee (x_1 \wedge \overline{x}_2 \wedge x_5 \wedge x_6) \vee (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_4 \wedge x_5).$$

It can be checked that g is a balanced function with $\mathbf{H}[F] \geq 3.92434$ and $\mathbf{Inf}[F] = 1.625$. Applying Lemma 3 with $F = g$, we get

$$\lim_{m \to \infty} \frac{\mathbf{H}[f_m]}{\mathbf{Inf}[f_m]} \geq \frac{3.92434}{1.625} + \frac{3.92434}{1.625 \times 0.625} = 6.278944.$$

□

References

1. Friedgut, E., Kalai, G.: Every monotone graph property has a sharp threshold. Proceedings of the American Mathematical Society 124(10), 2993–3002 (1996)
2. O'Donnell, R., Wright, J., Zhou, Y.: The Fourier Entropy-Influence Conjecture for certain classes of boolean functions. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 330–341. Springer, Heidelberg (2011)
3. Kalai, G.: The entropy/influence conjecture. Posted on Terence Tao's *What's new* blog (2007), `http://terrytao.wordpress.com/2007/08/16/gil-kalai-the-entropyinfluence-conjecture/`
4. Kahn, J., Kalai, G., Linial, N.: The influence of variables on Boolean functions. In: Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, pp. 68–80 (1988)
5. Mansour, Y.: Learning Boolean functions via the Fourier Transform. In: Roychowdhury, V., Siu, K.Y., Orlitsky, A. (eds.) Theoretical Advances in Neural Computation and Learning, pp. 391–424. Kluwer Academic Publishers (1994)
6. Gopalan, P., Kalai, A., Klivans, A.: Agnostically learning decision trees. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 527–536 (2008)
7. Gopalan, P., Kalai, A., Klivans, A.: A query algorithm for agnostically learning DNF? In: Proceedings of the 21st Annual Conference on Learning Theory, pp. 515–516 (2008)
8. De, A., Etesami, O., Trevisan, L., Tulsiani, M.: Improved pseudorandom generators for depth 2 circuits. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010. LNCS, vol. 6302, pp. 504–517. Springer, Heidelberg (2010)
9. Friedgut, E.: Boolean functions with low average sensitivity depend on few coordinates. Combinatorica 18(1), 27–36 (1998)
10. Servedio, R.: On learning monotone DNF under product distributions. Information and Computation 193(1), 57–74 (2004)
11. O'Donnell, R., Servedio, R.: Learning monotone decision trees in polynomial time. SIAM Journal on Computing 37(3), 827–844 (2008)
12. Klivans, A., Lee, H., Wan, A.: Mansour's Conjecture is true for random DNF formulas. In: Proceedings of the 23rd Annual Conference on Learning Theory, pp. 368–380 (2010)
13. Heiman, R., Newman, I., Wigderson, A.: On read-once threshold formulae and their randomized decision in tree complexity. Theor. Comput. Sci. 107(1), 63–76 (1993)
14. Keller, N., Mossel, E., Schlank, T.: A note on the entropy/influence conjecture. Discrete Mathematics 312(22), 3364–3372 (2012)

Large Neighborhood Local Search for the Maximum Set Packing Problem

Maxim Sviridenko* and Justin Ward**

Department of Computer Science, University of Warwick
{M.I.Sviridenko,J.D.Ward}@warwick.ac.uk

Abstract. In this paper we consider the classical maximum set packing problem where set cardinality is upper bounded by a constant k. We show how to design a variant of a polynomial-time local search algorithm with performance guarantee $(k+2)/3$. This local search algorithm is a special case of a more general procedure that allows to swap up to $\Theta(\log n)$ elements per iteration. We also design problem instances with locality gap $k/3$ even for a wide class of exponential time local search procedures, which can swap up to cn elements for a constant c. This shows that our analysis of this class of algorithms is almost tight.

1 Introduction

In this paper, we consider the problem of maximum unweighted k-set packing. In this problem, we are given a collection $\mathcal{N}$ of n distinct k-element subsets of some ground set X, where k is a constant. We say that two sets $A, B \in \mathcal{N}$ *conflict* if they share an element and call a collection of mutually non-conflicting sets from $\mathcal{N}$ a *packing*. Then, the goal of the unweighted k-set packing problem is to find a packing $\mathcal{A} \subseteq \mathcal{N}$ of maximum cardinality. Here, we assume that each set has cardinality *exactly* k. This assumption is without loss of generality, since we can always add unique elements to each set of cardinality less than k to obtain such an instance.

The maximum set packing problem is one the basic optimization problems. It received a significant amount of attention from researchers in the last few decades (see e.g. [8]). It is known that a simple local search algorithm that starts with an arbitrary feasible solution and tries to add a constant number of sets to the current solution while removing a constant number of conflicting sets has performance guarantee arbitrarily close to $k/2$ [7]. It was also shown in [7] that the analysis of such an algorithm is tight, i.e. there are maximum set covering instances where the ratio between a locally optimal solution value and the globally optimal solution value is arbitrarily close to $k/2$.

Surprisingly, Halldórsson[5] showed that if one increases the size of allowable swap to $\Theta(\log n)$ the performance guarantee can be shown to be at most $(k+2)/3$.

* Work supported by EPSRC grants EP/J021814/1, EP/D063191/1, FP7 Marie Curie Career Integration Grant, and Royal Society Wolfson Research Merit Award.
** Work supported by EPSRC grants EP/J021814/1 and EP/D063191/1.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 792–803, 2013.

Recently, Cygan, Grandoni and Mastrolilli [3] improved the guarantee for the same algorithm to $(k+1)/3$. This performance guarantee is the best currently known for the maximum set packing problem. The obvious drawback of these algorithms is that it runs in time $O(n^{\log n})$ and therefore its running time is not polynomial.

Both algorithms rely only on the subset of swaps of size $\Theta(\log n)$ to be able to prove their respective performance guarantees. The Halldórsson's swaps are particularly well structured and have a straightforward interpretation in the graph theoretic language. In section 4 we employ techniques from fixed-parameter tractability to yield a procedure for finding well-structured improvements of size $O(\log n)$ in polynomial time. Our algorithm is based on color coding technique introduced by Alon, Yuster, and Zwick [1] and its extension by Fellows et al. [4], and solves a dynamic program to locate an improvement if one exists. Combining with Halldórsson's analysis, we obtain a polynomial time $\frac{k+2}{3}$-approximation algorithm. In Section 6 we show that it is not possible to improve this result beyond $\frac{k}{3}$, even by choosing significantly larger improvements. Specifically, we construct a family of instances in which the locality gap for a local search algorithm applying all improvements of size t remains at least $\frac{k}{3}$ even when t is allowed to grow linearly with n. Our lower bound thus holds even for local search algorithms that are allowed to examine some exponential number of possible improvements at each stage.

2 A Quasi-Polynomial Time Local Search Algorithm

Let $\mathcal{A}$ be a packing. We define an auxiliary multigraph $G_{\mathcal{A}}$ whose vertices correspond to sets[1] in $\mathcal{A}$ and whose edges correspond to sets in $\mathcal{N} \setminus \mathcal{A}$ that conflict with at most 2 sets in $\mathcal{A}$. That is, $E(G_{\mathcal{A}})$ contains a separate edge (S,T) for each set $X \in \mathcal{N} \setminus \mathcal{A}$ that conflicts with exactly two sets S and T in $\mathcal{A}$, and a loop on S for each set $X \in \mathcal{N}$ that conflicts with exactly one set S in $\mathcal{A}$. In order to simplify our analysis, we additionally say that each set $X \in \mathcal{A}$ conflicts with itself, and place such a loop on each set of $\mathcal{A}$. Note that $G_{\mathcal{A}}$ contains $O(n)$ vertices and $O(n)$ edges, for any value of $\mathcal{A}$.

Our local search algorithm uses $G_{\mathcal{A}}$ to search for improvements to the current solution $\mathcal{A}$. Formally, we call a set I of t edges in $G_{\mathcal{A}}$ a *t-improvement* if I covers at most $t-1$ vertices of $G_{\mathcal{A}}$ and the sets of $\mathcal{N} \setminus \mathcal{A}$ corresponding to the edges in I are mutually disjoint.

Note that if I is a t-improvement for a packing $\mathcal{A}$, we can obtain a larger packing by removing the at most $t-1$ sets covered by I from $\mathcal{A}$ and then adding the t sets corresponding to the edges of I to the result. We limit our search for improvements in $G_{\mathcal{A}}$ to those that exhibit the following particular form: an improvement is a *canonical* improvement if it forms a connected graph containing two distinct cycles. Then, a canonical improvement comprises either 2

[1] To emphasize this correspondance, we shall use capital letters to refer to individual vertices of $G_{\mathcal{A}}$.

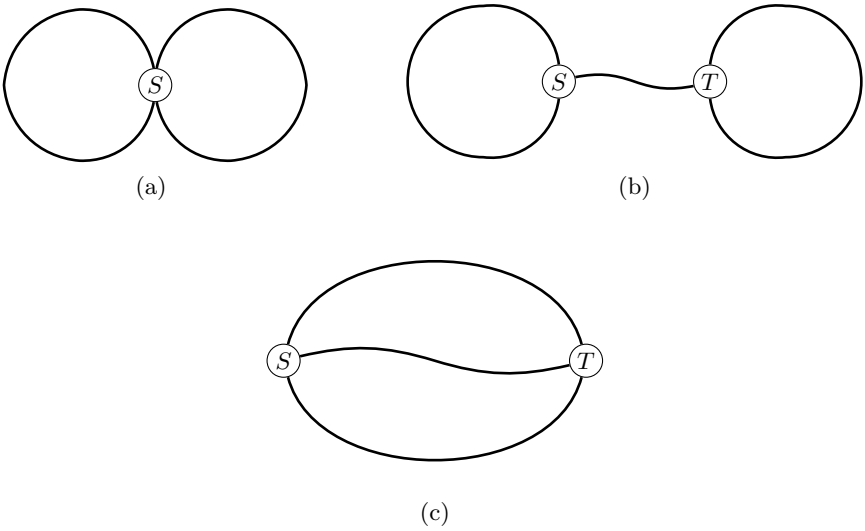


Fig. 1. Canonical Improvements

edge-disjoint cycles joined by a path, two edge-disjoint cycles that share a single vertex, or two distinct vertices joined by 3 edge-disjoint paths (see Figure 1).[2]

Our algorithm, shown in Figure 2 proceeds by repeatedly calling the procedure IMPROVE($G_{\mathcal{A}}$), which searches for a canonical $(4 \log n + 1)$-improvement in the graph $G_{\mathcal{A}}$. Before searching for a canonical improvement, we first ensure that $\mathcal{A}$ is a maximal packing by greedily adding sets from $\mathcal{N} \setminus \mathcal{A}$ to $\mathcal{A}$. If IMPROVE($G_{\mathcal{A}}$) returns an improvement I, then I is applied to the current solution and the search continues. Otherwise, the current solution $\mathcal{A}$ is returned.

In Section 3, we analyze the approximation performance of the local search algorithm under the assumption that IMPROVE($G_{\mathcal{A}}$) always finds a canonical $(4 \log n + 1)$-improvement, whenever such an improvement exists. In Section 4, we provide such an implementation IMPROVE($G_{\mathcal{A}}$) that runs in deterministic polynomial time.

3 Locality Gap of the Algorithm

In this section we prove the following upper bound on the locality gap for our algorithm. We consider an arbitrary instance $\mathcal{N}$ of k-set packing, and let $\mathcal{A}$ be the packing in $\mathcal{N}$ produced by our local search algorithm and $\mathcal{B}$ be any other packing in $\mathcal{N}$.

[2] It can be shown that every t-improvement must contain a canonical t-improvement, and so we are not in fact restricting the search space at all by considering only canonical improvements. However, this fact will not be necessary for our analysis.

```
A ← ∅
loop
    for all S ∈ N \ A do
        if S does not conflict with any set of A then
            A ← A ∪ {S}
        end if
    end for

    Construct the auxiliary graph G_A for A
    I ← Improve(G_A)
    if I = ∅ then
        return A
    else
        A ← (A \ V(I)) ∪ E(I)
    end if
end loop
```

Fig. 2. The General Local Search Procedure

Theorem 1. $|\mathcal{B}| \leq \frac{k+2}{3}|\mathcal{A}|$.

For the purpose of our analysis, we consider the subgraph $H_{\mathcal{A},\mathcal{B}}$ of $G_{\mathcal{A}}$ consisting of only those edges of $G_{\mathcal{A}}$ corresponding to sets in $\mathcal{B}$. Then, every collection of edges in $H_{\mathcal{A},\mathcal{B}}$ is also present in $G_{\mathcal{A}}$. Moreover, because the edges of $H_{\mathcal{A},\mathcal{B}}$ all belong to the packing $\mathcal{B}$, any subset of them must be mutually disjoint. Thus, we can assume that no collection of at most $4\log n + 1$ edges from $H_{\mathcal{A},\mathcal{B}}$ form any of the structures shown in Figure 1. Otherwise, the corresponding collection of edges in $G_{\mathcal{A}}$ would form a canonical $(4\log n + 1)$-improvement.

In order to prove Theorem 1, we make use of the following lemma of Berman and Fürer [2], which gives conditions under which the multigraph $H_{\mathcal{A},\mathcal{B}}$ must contain a canonical improvement.[3]

Lemma 1 (Lemma 3.2 in [2]). *Assume that $|E| \geq \frac{p+1}{p}|V|$ in a multigraph $H = (V, E)$, for some integer p. Then, H contains a canonical improvement with at most $4p\log n - 1$ vertices.*

It will also be necessary to bound the total number of loops in $H_{\mathcal{A},\mathcal{B}}$. In order to do this, we shall make use of the following fact, which is implicit in the analysis Halldórsson [5]. We consider a second auxiliary graph $H'_{\mathcal{A},\mathcal{B}}$ that is obtained from $H_{\mathcal{A},\mathcal{B}}$ in the following fashion:

Lemma 2. *Let $H = (V, E)$ be a multigraph and let $H' = (V', E')$ be obtained from H by deleting all vertices of H with loops on them and, for each edge with one endpoint incident to a deleted vertex, introducing a new loop on this edge's remaining vertex. Let $t \geq 3$. Then, if H' contains a canonical t-improvement, H contains a canonical $(t+2)$-improvement.*

[3] Berman and Fürer call structures of the form shown in Figure 1 "binoculars." Here, we have rephrased their lemma in our own terminology.

We now turn to the proof of Theorem 1. Every set in $\mathcal{B}$ must conflict with some set in $\mathcal{A}$, or else $\mathcal{A}$ would not be maximal. We partition the sets of $\mathcal{B}$ into three collections of sets, depending on how many sets in $\mathcal{A}$ they conflict with. Let $\mathcal{B}_1$,$\mathcal{B}_2$, and $\mathcal{B}_3$ be collections of those sets of $\mathcal{B}$ that conflict with, respectively, exactly 1, exactly 2, and 3 or more sets in $\mathcal{A}$ (note that each set of $\mathcal{A} \cap \mathcal{B}$ is counted in $\mathcal{B}_1$, since we have adopted the convention that such sets conflict with themselves).

Because each set in $\mathcal{A}$ contains at most k elements and the sets in $\mathcal{B}$ are mutually disjoint, we have the inequality

$$|\mathcal{B}_1| + 2|\mathcal{B}_2| + 3|\mathcal{B}_3| \leq k|\mathcal{A}|. \tag{1}$$

We now bound the size of $\mathcal{B}_1$ and $\mathcal{B}_2$.

Let $\mathcal{A}_1$ be the collection of sets from $\mathcal{A}$ that conflict with sets of $\mathcal{B}_1$. Then, note that each set of $\mathcal{B}_1$ corresponds to a loop in $H_{\mathcal{A},\mathcal{B}}$ and the sets of $\mathcal{A}_1$ correspond to the vertices on which these loops occur. Any vertex of $H_{\mathcal{A},\mathcal{B}}$ with two loops would form an improvement of the form shown in Figure 1a. Thus, each vertex in $H_{\mathcal{A},\mathcal{B}}$ has at most 1 loop and hence

$$|\mathcal{B}_1| = |\mathcal{A}_1|. \tag{2}$$

Now, we show that $|\mathcal{B}_2| \leq 2|A \setminus \mathcal{A}_1|$. By way of contradiction, suppose that $|\mathcal{B}_2| \geq 2|\mathcal{A} \setminus \mathcal{A}_1|$. We construct an auxiliary graph $H'_{\mathcal{A},\mathcal{B}}$ from $H_{\mathcal{A},\mathcal{B}}$ as in Lemma 2. The number of edges in this graph is exactly $|\mathcal{B}_2|$ and the number of vertices is exactly $|A \setminus \mathcal{A}_1|$. Thus, if $|\mathcal{B}_2| \leq 2|A \setminus \mathcal{A}_1|$, then from Lemma 1 (with $p = 1$), there is a canonical improvement in $H'_{\mathcal{A},\mathcal{B}}$ of size at most $4 \log n - 1$. But, from Lemma 2 this means there must be a canonical improvement in $H_{\mathcal{A},\mathcal{B}}$ of size at most $4 \log n + 1$, contradicting the local optimality of $\mathcal{A}$. Thus,

$$|\mathcal{B}_2| < 2|\mathcal{A} \setminus \mathcal{A}_1| \tag{3}$$

Adding (1), twice (2), and (3), we obtain

$$3|\mathcal{B}_1| + 3|\mathcal{B}_2| + 3|\mathcal{B}_3| \leq k|\mathcal{A}| + 2|\mathcal{A}_1| + 2|\mathcal{A} \setminus \mathcal{A}_1|,$$

which implies that $3|\mathcal{B}| \leq (k + 2)|\mathcal{A}|$.

4 Finding Canonical Improvements

A naïve implementation of the local search algorithm described in Section 2 would run in only quasi-polynomial time, since at each step there are $n^{\Omega(\log n)}$ possible improvements of size $t = 4 \log n + 1$. In contrast, we now show that it is possible to find a *canonical* improvement of size t in polynomial time whenever one exists.

We first give a randomized algorithm, using the color coding approach of Alon, Yuster, and Zwick [1]. If some t-improvement exists, our algorithm finds it with polynomially small probability. In Section 5, we show how to use this algorithm

to implement a local search algorithm that succeeds with high probability, and how to obtain a deterministic variant via derandomization.

We now describe the basic, randomized color coding algorithm. Again, consider an arbitrary instance $\mathcal{N}$ of k-set packing and let X be the ground set of $\mathcal{N}$. Let K be a collection of kt colors. We assign each element of X a color from K uniformly at random, and assign each k-set from $\mathcal{N}$ the set of all its elements' colors. We say that a collection of sets $\mathcal{A} \subseteq \mathcal{N}$ is *colorful* if no color appears twice amongst the sets of $\mathcal{A}$. We note that if a collection of sets $\mathcal{A}$ is colorful, then $\mathcal{A}$ must form a packing, since no two sets in $\mathcal{A}$ can share an element.

We assign each edge of $G_{\mathcal{A}}$ the same set of colors as its corresponding set in $\mathcal{N}$, and, similarly, say that a collection of edges is colorful if the corresponding collection of sets from $\mathcal{N}$ is colorful. Now, we consider a subgraph of $G_{\mathcal{A}}$ made up of some set of at most t edges I. If this graph has the one of the forms shown in Figure 1 and I is colorful, then I must be a canonical t-improvement. We now show that, although the converse does not hold, our random coloring makes any given canonical t-improvement colorful with probability only polynomially small in n.

Consider a canonical improvement I of size $1 \leq i \leq t$. The i sets corresponding to the edges of I must be disjoint and so consist of ki separate elements. The probability that I is colorful is precisely the probability that all of these ki elements are assigned distinct colors. This probability can be estimated as

$$\frac{\binom{kt}{ki}(ki)!}{(kt)^{ki}} = \frac{(kt)!}{(kt-ki)!(kt)^{ki}} \geq \frac{(kt)!}{(kt)^{kt}} > e^{-kt} = e^{-4k\log n - k} = e^{-k} n^{-8k}, \quad (4)$$

where in the last line, we have used the fact that $e^{\log n} = e^{\ln n \log e} = n^{\log e} < n^2$.

We now show how to use this random coloring to find canonical improvements in $G_{\mathcal{A}}$. Our approach is based on finding colorful paths and cycles and employs dynamic programming.

We give a dynamic program that, given a coloring for edges of $G_{\mathcal{A}}$, as described above, finds a colorful path of length at most t in $G_{\mathcal{A}}$ between each pair of vertices S and T, if such a path exists. For each vertex S and T of $G_{\mathcal{A}}$, each value $i \leq t$, and each set C of ki colors from K, we have an entry $\mathcal{D}(S, T, i, C)$ that records whether or not there is some colorful path of length i between S and T whose edges are colored with precisely those colors in C. In our table, we explicitly include the case that $S = T$.

We compute the entries of $\mathcal{D}$ bottom-up in the following fashion. We set $\mathcal{D}(S, T, 0, C) = 0$ for all pairs of vertices S, T, and C. Then, we compute the entries $\mathcal{D}(S, T, i, C)$ for $i > 0$ as follows. We set $\mathcal{D}(S, T, i, C) = 1$, if there is some edge (V, T) incident to vertex T in $G_{\mathcal{A}}$ such that (V, T) is colored with a set of k distinct colors $B \subseteq C$ and the entry $\mathcal{D}(S, V, i-1, C \setminus B) = 1$. Otherwise, we set $\mathcal{D}(S, T, iC) = 0$.

To determine if $G_{\mathcal{A}}$ contains a colorful path of given length $i \leq t$ from S to T, we simply check whether $\mathcal{D}(S, T, i, C) = 1$ for some set of colors C. Similarly, we can use our dynamic program to find colorful *cycles* of length j that include some given vertex U by consulting $\mathcal{D}(U, U, j, C)$ for each set of colors C. The

actual path or cycle can then be found by backtracking through the table $\mathcal{D}$. We note that while the cycles and paths found by this procedure are not necessarily simple, they are edge-disjoint.

For each value of i, there are at most $n^2\binom{kt}{ki}$ entries in $\mathcal{D}(S,T,i,C)$. To compute each such entry, we examine each edge (V,T) incident to T, check if (V,T) is colored with a set of k colors $B \subseteq C$ and consult $\mathcal{D}(S,T,i,C \setminus B)$, all which can be accomplished in time $O(nki)$. Thus, the total time to compute $\mathcal{D}$ up to $i = t$ is of order

$$\sum_{i=1}^{t} n^3 ki \binom{kt}{ki} \leq n^4 kt 2^{kt}.$$

In order to find a canonical t-improvement, we first compute the table $\mathcal{D}$ up to $i = t$. Then, we search for improvements of each kind shown in Figure 1 by enumerating over all choices of S and T, and looking for an appropriate collection of cycles or paths involving these vertices that use mutually disjoint sets of colors. Specifically:

- To find improvements of the form shown in Figure 1a, we enumerate over all n vertices S. For all disjoint sets of ka and kb colors C_a and C_b with $a + b \leq t$, we check if $\mathcal{D}(S,S,a,C_a) = 1$ and $\mathcal{D}(S,S,b,C_b) = 1$. This can be accomplished in time

$$n \sum_{a=1}^{t} \sum_{b=1}^{t-a} 2^{ka} 2^{kb} kt = O(nkt^3 2^{kt}).$$

- To find improvements of the form shown in Figure 1b we enumerate over all distinct vertices S and T. For all disjoint sets of ka, kb, and kc colors C_a,C_b,and C_c with $|C_a| + |C_b| + |C_c| \leq t$, we check if $\mathcal{D}(S,S,a,C_a) = 1$, $\mathcal{D}(T,T,b,C_b) = 1$, and $\mathcal{D}(S,T,c,C_c) = 1$. This can be accomplished in time

$$n^2 \sum_{a=1}^{t} \sum_{b=1}^{t-a} \sum_{c=1}^{t-a-b} 2^{ka} 2^{kb} 2^{kc} kt = O(n^2 kt^4 2^{kt}).$$

- To find improvements of the form shown in Figure 1c we again enumerate over all distinct vertices S and T. For all disjoint sets of ka, kb, and kc colors C_a,C_b,and C_c with $|C_a| + |C_b| + |C_c| \leq t$, we check if $\mathcal{D}(S,T,a,C_a) = 1$, $\mathcal{D}(S,T,b,C_b) = 1$, and $\mathcal{D}(S,T,c,C_c) = 1$. This can be accomplished in time

$$n^2 \sum_{a=1}^{t} \sum_{b=1}^{t-a} \sum_{c=1}^{t-a-b} 2^{ka} 2^{kb} 2^{kc} kt = O(n^2 kt^4 2^{kt}).$$

The total time spent searching for a canonical t-improvement is thus at most

$$O(n^2 kt 2^{kt} + nkt^3 2^{kt} + 2n^2 kt^4 2^{kt}) = O(n^2 kt^4 2^{kt}) = O(2^k k \cdot n^{4k+2} \log^4 n).$$

5 The Deterministic, Large Neighborhood Local Search Algorithm

The analysis of the local search algorithm in Section 3 supposed that every call to IMPROVE($G_{\mathcal{A}}$) returns $\emptyset$ only when no canonical t-improvement exists in $G_{\mathcal{A}}$. Under this assumption, the algorithm is a $\frac{k+2}{3}$-approximation. In contrast, the dynamic programming implementation given in Section 4 may fail to find a canonical improvement I if the chosen random coloring does not make I colorful. As we have shown in (4), this can happen with probability at most $(1-e^{-k}n^{-8k})$.

Suppose that we implement each call to IMPROVE($G_{\mathcal{A}}$) by running the algorithm of Section 4 $cN = ce^k n^{8k} \ln n$ times, each with a different random coloring. We now show that the resulting algorithm is a polynomial time $\frac{k+2}{3}$-approximation with high probability $1 - n^{1-c}$.

We note that each improvement found by our local search algorithm must increase the size of the packing $\mathcal{A}$, and so the algorithm makes at most n calls to IMPROVE($G_{\mathcal{A}}$). We set $N = e^k n^{8k+1} \ln n$, and then implement each such call by repeating the color coding algorithm of Section 4 cN times for some $c > 1$, each with an new random coloring. The probability that any given call IMPROVE($G_{\mathcal{A}}$) succeeds in finding a canonical t-improvement when one exists is then at least:

$$1 - (1 - e^{-k}n^{-8k})^{cN} \geq 1 - \exp\{e^{-k}n^{-8k} \cdot ce^k n^{8k} \ln n\} = 1 - n^{-c}.$$

And so, the probability that all calls to IMPROVE($G_{\mathcal{A}}$) satisfy the assumptions of Theorem 1 is at least

$$(1 - n^{-c})^n \geq 1 - n^{1-c}$$

The resulting algorithm is therefore a $\frac{k+2}{3}$-approximation with high probability. It requires at most n calls to IMPROVE($G_{\mathcal{A}}$), each requiring total time

$$O(cN \cdot 2^k k n^{4k+2} \log^4 n) = O(c(2e)^k k n^{12k+2} \log^n n \ln n) = cn^{O(k)}$$

Using the general approach described by Alon, Yuster, and Zwick [1], we can in fact give a *deterministic* implementation of IMPROVE($G_{\mathcal{A}}$), which *always* succeeds in finding a canonical t-improvement in $G_{\mathcal{A}}$ if such an improvement exists. Rather than choosing a coloring of the ground set X at random, we use a collection $\mathcal{K}$ of colorings (each of which is given as a mapping $X \to K$) with the property that every canonical t-improvement $G_{\mathcal{A}}$ is colorful with respect to some coloring in $\mathcal{K}$. For this, it is sufficient to find a collection of $\mathcal{K}$ of colorings such that for every set of at most kt elements in X, there is some coloring in $\mathcal{K}$ that assigns these kt elements kt distinct colors from K. Then, we implement IMPROVE($G_{\mathcal{A}}$) by running the dynamic programming algorithm of Section 5 on each such coloring, and returning the first improvement found. Because every canonical t-improvement contains at most kt distinct elements of the ground set, every such improvement must be made colorful by some coloring in $\mathcal{K}$, and so IMPROVE($G_{\mathcal{A}}$) will always find a canonical t-improvement if one exists.

We now show how to construct the desired collection of colorings $\mathcal{K}$ by using a *kt-perfect family* of hash functions from $X \to K$. Briefly, a perfect hash function

for a set $S \subseteq A$ is a mapping from A to B that is one-to-one on S. A p-perfect family is then collection of perfect hash functions, one for each set $S \subseteq A$ of size at most p. Building on work by Fredman, Komlós and Szemerédi [10] and Schmidt and Siegal [9], Alon and Naor show (in Theorem 3 of [11]) how to explicitly construct a perfect hash function from $[m]$ to $[p]$ for some $S \subset [m]$ of size p in time $\tilde{O}(p \log m)$. This hash function is described in $O(p + \log p \cdot \log\log m)$ bits. The maximum size of a p-perfect family of such functions is therefore $2^{O(p+\log p \cdot \log\log m)}$. Moreover, the function can be evaluated in time $O(\log m / \log p)$.

Then, we can obtain a deterministic, polynomial time $\frac{k+2}{3}$ approximation as follows. Upon receiving the set packing instance $\mathcal{N}$ with ground set X, we compute a kt-perfect family $\mathcal{K}$ of hash functions from X to a set of kt colors K. Then, we implement each call to $\textsc{Improve}(G_{\mathcal{A}})$ as described, by enumerating over the colorings in $\mathcal{K}$. We note that since each set in $\mathcal{N}$ has size k, $|X| \leq |\mathcal{N}|k = nk$, so each improvement makes at most

$$2^{O(kt+\log kt \cdot \log\log kn)} = 2^{O(k \log n + \log(k \log n) \cdot \log\log kn)} = n^{O(k)}$$

calls to the dynamic programming algorithm of Section 5 (one per coloring in $\mathcal{K}$) and each of these calls takes time at most $n^{O(k)}$ (including the time to evaluate the coloring on each element of the ground set). Moreover, the initial construction of $\mathcal{K}$ takes time at most $2^{kt}\tilde{O}(kt \log kn) = n^{O(k)}$.

6 A Lower Bound

We now show that our analysis is almost tight. Specifically, we show that the locality gap of t-local search is at least $\frac{k}{3}$, even when t is allowed to grow on the order of n.

Theorem 2. *Let $c = \frac{9}{2e^5 k}$ and suppose that $t \leq cn$ for all sufficiently large n. There, there exist 2 pairwise disjoint collections of k-sets $\mathcal{S}$ and $\mathcal{O}$ with $|\mathcal{S}| = 3n$ and $|\mathcal{O}| = kn$ such that any collection of $a \leq t$ sets in $\mathcal{O}$ conflict with at least a sets in $\mathcal{S}$.*

In order to prove Theorem 2 we make use of the following (random) construction. Let X be a ground set of $3kn$ elements, and consider a collection $\mathcal{S}$ of $3n$ sets, each containing k distinct elements of X. We construct a random collection $\mathcal{R}$ of kn disjoint subsets of X, each containing 3 elements.

The number of distinct collections $\mathcal{R}$ generated by this procedure is equal to the number of ways to partition the $3kn$ elements of X into kn disjoint 3-sets. We define the quantity $\tau(m)$ to be the number of ways that m elements can be divided into $m/3$ disjoint 3-sets:

$$\tau(m) \triangleq \frac{m!}{(3!)^{m/3}(m/3)!}.$$

In order to verify the above formula, consider the following procedure for generating a random partition. We first arrange the m elements in some order and

then make a 3-set from the elements at positions $3i$, $3i-1$ and $3i-2$, for each $i \in [\frac{m}{3}]$ (that is, we group each set of 3 consecutive elements in the ordering into a triple). Now, we note that two permutations of the elements produce the same partition if they differ only in the ordering of the 3 elements within each of the $m/3$ triples or in the ordering of the $m/3$ triples themselves. Thus, any given partition is generated by exactly $(3!)^{m/3}(m/3)!$ of the $m!$ possible orderings.

The probability that any particular collection of a disjoint 3-sets occurs in $\mathcal{R}$ is given by

$$p(a) \triangleq \frac{\tau(3kn-3a)}{\tau(3kn)}.$$

This is simply the number of ways to partition the remaining $3kn-3a$ elements into $kn-a$ disjoint 3-sets, divided by the total number of possible partitions of all $3kn$ elements.

We say that a collection $\mathcal{A}$ of a sets in $\mathcal{S}$ is *unstable* if there is some collection $\mathcal{B}$ of at least a sets in $\mathcal{R}$ that conflict with only those sets in $\mathcal{A}$. Note that there is an improvement of size a for $\mathcal{S}$ only if there is some unstable collection $\mathcal{A}$ of size a in $\mathcal{S}$.[4] We now derive an upper bound on the probability that our random construction of $\mathcal{R}$ results in a given collection $\mathcal{A}$ in $\mathcal{S}$ becoming unstable.

Lemma 3. *A collection of a sets in $\mathcal{S}$ is unstable with probability less than* $\frac{\binom{ka}{3da}\binom{kn}{da}}{\binom{3kn}{3da}}$.

Proof. A collection $\mathcal{A}$ of a k-sets from $\mathcal{S}$ is unstable precisely when there is a collection $\mathcal{B}$ of a 3-sets in $\mathcal{R}$ that contain only those $k(a-1)$ elements appearing in the sets of $\mathcal{A}$. There are $\binom{ka}{3a}$ ways to choose the $3a$ elements from which we construct $\mathcal{B}$. For each such choice, there are $\tau(3a)$ possible ways to partition the elements into 3-sets, each occurring with probability $p(3a) = \tau(3kn-3a)/\tau(3n)$. Applying the union bound, the probability that $\mathcal{A}$ is unstable is then at most

$$\begin{aligned}
\binom{ka}{3a}\tau(3a)\frac{\tau(3kn-3a)}{\tau(3kn)} &= \binom{ka}{3a}\frac{(3a)!}{(3!)^a a!}\cdot\frac{(3(kn-a))!}{(3!)^{kn-a}(kn-a)!}\cdot\frac{(3!)^{kn}(kn)!}{(3kn)!} \\
&= \binom{ka}{3a}\frac{(3a)!(3(kn-a))!}{(3kn)!}\frac{(kn)!}{(kn-a)!a!} \\
&= \frac{\binom{ka}{3a}\binom{kn}{a}}{\binom{3kn}{3a}}.
\end{aligned}$$

□

Proof (Theorem 2). Let U_a be number of unstable collections of size a in $\mathcal{S}$, and consider $\mathbb{E}[U_a]$. There are precisely $\binom{3n}{a}$ such collections, and from Lemma 3, each occurs with probability less than $\frac{\binom{ka}{3a}\binom{kn}{a}}{\binom{3kn}{3a}}$. Thus:

$$\mathbb{E}[U_a] < \frac{\binom{3n}{a}\binom{ka}{3a}\binom{kn}{a}}{\binom{3kn}{3a}} \tag{5}$$

[4] In fact, for an improvement to exist, there must be some collection $\mathcal{B}$ of at least $a+1$ such sets in $\mathcal{R}$. This stronger condition is unnecessary for our bound, however.

Applying the upper and lower bounds

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k,$$

in the numerator and denominator, respectively, of (5), we obtain the upper bound

$$\frac{(e3n)^a}{a^a} \cdot \frac{(eka)^{3a}}{(3a)^{3a}} \cdot \frac{(ekn)^a}{a^a} \cdot \frac{(3a)^{3a}}{(3kn)^{3a}} = \left(\frac{e^5 3^4 k^4 a^6 n^2}{3^6 k^3 a^5 n^3}\right)^a = \left(\frac{e^5 ka}{9n}\right)^a.$$

Then, the expected number of unstable collections in $\mathcal{S}$ of size *at most* t (and hence the expected number of t-improvements for $\mathcal{S}$) is less than

$$\sum_{a=1}^{t} \mathbb{E}[U_a] < \sum_{a=1}^{t} \left(\frac{e^5 ka}{9n}\right)^a. \tag{6}$$

For all sufficiently large n, we have $a \leq t \leq cn$ and so

$$\sum_{a=1}^{t} \left(\frac{e^5 ka}{9n}\right)^a \leq \sum_{a=1}^{t} \left(\frac{e^5 kcn}{9n}\right)^a = \sum_{a=1}^{t} \left(\frac{e^5 k}{9}\frac{9}{2e^5 k}\right)^a = \sum_{a=1}^{t} \left(\frac{1}{2}\right)^a < 1.$$

Thus, there must exist some collection $\mathcal{O}$ in the support of $\mathcal{R}$ that creates no unstable collections of size at most t in $\mathcal{S}$. We add $k-3$ new, unique elements to each of the 3-sets in $\mathcal{O}$ to obtain a collection of k-sets, noting that this does not affect the stability of any collection in $\mathcal{S}$. Then, $\mathcal{O}$ is a collection of kn pairwise disjoint k-sets satisfying the conditions of the theorem. □

7 Conclusion

We have given a polynomial time $\frac{k+2}{3}$ approximation algorithm for the problem of k-set packing. Our algorithm is based on a simple local search algorithm, but incorporates ideas from fixed parameter tractability to search large neighborhoods efficiently, allowing us to achieve an approximation guarantee exceeding the $k/2$ bound of Hurkens and Schrijver [7]. In contrast, our lower bound of $k/3$ shows that local search algorithms considering still larger neighborhoods, including neighborhoods of exponential size, can yield only slight improvements.

An interesting direction for future research would be to close the gap between our $k/3$ lower bound and $\frac{k+2}{3}$ upper bound. Recently, Cygan, Grandoni, and Mastrolilli [3] have given a quasi-polynomial time local search algorithm attaining an approximation ratio of $(k+1)/3$. Their analysis is also based on that of Berman and Fürer [2] and Halldórsson[5], but their algorithm requires searching for improvements with a more general structure than those that we consider, and it is unclear how to apply similar techniques as ours in this case. Nevertheless, we conjecture that it is possible to attain an approximation ratio of $\frac{k+1}{3}$ in polynomial time, although this will likely require more sophisticated techniques than we consider here.

In contrast to all known positive results, the best known NP-hardness result for k-set packing is, due to Hazan, Safra, and Schwartz [6], is only $O(k/\ln k)$. A more general open problem is whether the gap between this result and algorithmic results can be narrowed.

Finally, we ask whether our results can be generalized to the independent set problem in $(k+1)$-claw free graphs. Most known algorithms for k-set packing, including those given by Halldórsson [5] and Cygan, Grandoni, and Mastrolilli [3] generalized trivially to this setting. However, this does not seem to be the case for the color coding approach that we employ, as it relies on the set packing representation of problem instances.

Acknowledgements. We would like to thank Oleg Pikhurko for extremely enlightening discussion on random graphs.

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. Journal of the ACM 42(4), 844–856 (1995)
2. Berman, P., Fürer, M.: Approximating maximum independent set in bounded degree graphs. In: SODA (1994)
3. Cygan, M., Grandoni, F., Mastrolilli, M.: How to sell hyperedges: the hypermatching assignment problem. In: SODA (2013)
4. Fellows, M., Knauer, C., Nishimura, N.: Faster fixed-parameter tractable algorithms for matching and packing problems. Algorithmica, 167–176 (2004)
5. Halldórsson, M.: Approximating Discrete Collections via Local Improvements. In: SODA (1995)
6. Hazan, Safra, S., Schwartz, O.: On the complexity of approximating k-set packing. Computational Complexity 15(1), 20–39 (2006)
7. Hurkens, C., Schrijver, A.: On the Size of Systems of Sets Every t of Which Have an SDR, with an Application to the Worst-Case Ratio of Heuristics for Packing Problems. SIAM Journal of Discrete Mathematics 2(1), 68–72 (1989)
8. Paschos, V.: A survey of approximately optimal solutions to some covering and packing problems. ACM Computing Surveys 29(2), 171–209 (1997)
9. Schmidt, J.P., Siegel, A.: The spatial complexity of oblivious k-probe hash functions. SIAM Journal on Computing 19(5), 775–786 (1990)
10. Fredman, M., Komlós, J., Szemerédi, E.: Storing a sparse table with O(1) worst case access time. Journal of the ACM 31(3), 538–544 (1984)
11. Alon, N., Naor, M.: Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. Algorithmica 16(4/5), 434–449 (1996)

The Complexity of Three-Element Min-Sol and Conservative Min-Cost-Hom*

Hannes Uppman**

Department of Computer and Information Science,
Linköping University, SE-581 83 Linköping, Sweden
hannes.uppman@liu.se

Abstract. Thapper and Živný [STOC'13] recently classified the complexity of VCSP for all finite-valued constraint languages. However, the complexity of VCSPs for constraint languages that are not finite-valued remains poorly understood. In this paper we study the complexity of two such VCSPs, namely Min-Cost-Hom and Min-Sol. We obtain a full classification for the complexity of Min-Sol on domains that contain at most three elements and for the complexity of conservative Min-Cost-Hom on arbitrary finite domains. Our results answer a question raised by Takhanov [STACS'10, COCOON'10].

1 Introduction

The *valued constraint satisfaction problem* (VCSP) is a very broad framework in which many combinatorial optimisation problems can be expressed. A *valued constraint language* is a fixed set of cost functions from powers of a *finite domain*. An instance of VCSP for some give constraint language is then a weighted sum of cost functions from the language. The goal is to minimise this sum. On the two-element domain the complexity of the problem is known for every constraint language [4]. Also for every language containing all $\{0,1\}$-valued unary cost functions the complexity is known [15]. In a recent paper Thapper and Živný [21] managed to classify the complexity of VCSP for all finite-valued constraint languages. However, VCSPs with other types of languages remains poorly understood.

In this paper we study the complexity of the *(extended) minimum cost homomorphism problem* (Min-Cost-Hom) and the *minimum solution problem* (Min-Sol). These problems are both VCSPs with special types of languages in which all non-unary cost-functions are crisp ($\{0,\infty\}$-valued). Despite this rather severe restriction the frameworks allow many natural combinatorial optimisation problems to be expressed. Min-Sol does e.g. generalise a large class of bounded integer linear programs. It may also be viewed as a generalisation of the problem Min-Ones [14] to larger domains. The problem Min-Cost-Hom is even more general and contains Min-Sol as a special case.

* A longer version of this paper is available at http://arxiv.org/abs/1301.0027

** Partially supported by the National Graduate School in Computer Science (CUGS), Sweden.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 804–815, 2013.

The problem Min-Sol has received a fair bit of attention in the literature and has e.g. had its complexity fully classified for all graphs of size three [13] and for all so-called homogeneous languages [9]. For more information about Min-Sol see [11] and the references therein. The "unextended version" of Min-Cost-Hom was introduced in [6] motivated by a problem in defence logistics. It was studied in a series of papers before it was completely solved in [18]. The more general version of the problem which we are interested in was introduced in [19].[1]

Methods and Results. We obtain a full classification of the complexity of Min-Sol on domains that contain at most three elements. The tractable cases are given by languages that can be solved by a certain linear programming formulation [20] and a new class that is inspired by, and generalises, languages described in [18,19]. A precise classification is given by Theorem 16. For conservative Min-Cost-Hom (i.e. Min-Cost-Hom with languages containing all unary crisp cost functions) an almost complete classification (for arbitrary finite domains) was obtained by Takhanov [19]. We are able to remove the extra conditions needed in [19] and provide a full classification for this problem. This answers a question raised in [18,19]. The main mathematical tools used througout the paper are from the so-called algebraic approach, see e.g. [2,7], and its extensions to optimisation problems [3,5]. Following [21] we also make use of Motzkin's Transposition Theorem from the theory of linear equations.

The rest of the paper is organised as follows. Section 2 contains needed concepts and results from the literature, Sect. 3 holds the description of our results and Sects. 4, 5 and 6 contain proofs of some of the theorems stated in Sect. 3.

2 Preliminaries

For a set Γ of finitary relations on a finite set D (the domain), and a finite set Δ (referred to as the domain valuations) of functions $D \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$, we define Min-Cost-Hom$(\Gamma, \Delta)$ as the following optimisation problem.

Instance: A triple (V, C, w) where
- V is a set of variables,
- C is a set of Γ-allowed constraints, i.e. a set of pairs (s, R) where the constraint-scope s is a tuple of variables, and the constraint-relation R is a member of Γ of the same arity as s, and
- w is a weight function $V \times \Delta \to \mathbb{Q}_{\geq 0}$.

Solution: A function $\varphi : V \to D$ s.t. for every $(s, R) \in C$ it holds that $\varphi(s) \in R$, where φ is applied component-wise.

Measure: The measure of a solution φ is $m(\varphi) = \sum_{v \in V} \sum_{\nu \in \Delta} w(v, \nu)\nu(\varphi(v))$.

The objective is to find a solution φ that minimises $m(\varphi)$.

[1] The definition in [19] is slightly more restrictive than the one we use. Also the notation differs; what we denote Min-Cost-Hom(Γ, Δ) is in [19] referred to as $MinHom_{\Delta}(\Gamma)$.

The problem Min-Sol(Γ, ν), which we define only for injective functions $\nu : D \to \mathbb{Q}_{\geq 0}$, is the problem Min-Cost-Hom$(\Gamma, \{\nu\})$. The "regular" *constraint satisfaction problem* (CSP) can also be defined through Min-Cost-Hom; an instance of CSP(Γ) is an instance of Min-Cost-Hom$(\Gamma, \emptyset)$, and the objective is to determine if any solution exists.

We will call the pair (Γ, Δ) a *language* (or a Min-Cost-Hom-language). The language $(\Gamma, \{\nu\})$ is written (Γ, ν). For an instance I we use Opt(I) for the measure of an optimal solution (defined only if a solution exists), Sol(I) denotes the set of all solutions and Optsol(I) the set of all optimal solutions. We define $0 \infty = \infty 0 = 0$ and for all $x \in \mathbb{Q}_{\geq 0} \cup \{\infty\}$, $x \leq \infty$ and $x + \infty = \infty + x = \infty$. The i:th projection operation will be denoted pr_i. We define $\binom{A}{2} = \{\{x, y\} \subseteq A : x \neq y\}$. $\mathcal{O}_D^{(m)}$ is used for the set of all m-ary operations on D. For binary operations f, g and h we define $\overline{f}$ through $\overline{f}(x, y) = f(y, x)$ and $f[g, h]$ through $f[g, h](x, y) = f(g(x, y), h(x, y))$. A k-ary operation f on D is called *conservative* if $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$ for every $x_1, \ldots, x_k \in D$. A ternary operation m on D is called *arithmetical* on $B \subseteq \binom{D}{2}$ if for every $\{a, b\} \in B$ the function m satisfies $m(a, b, b) = m(a, b, a) = m(b, b, a) = a$.

Polymorphisms. Let (Γ, Δ) be a language on the domain D. By Γ^c we denote Γ enriched with all constants, i.e. $\Gamma \cup \{\{c\} : c \in D\}$. An operation $f : D^m \to D$ is called a *polymorphism* of Γ if for every $R \in \Gamma$ and every sequence $t^1, \ldots, t^m \in R$ it holds that $f(t^1, \ldots, t^m) \in R$ where f is applied component-wise. The set of all polymorphisms of Γ is denoted Pol(Γ). A function $\omega : \mathcal{O}_D^{(k)} \to \mathbb{Q}_{\geq 0}$ is a k-ary *fractional polymorphism* [3] of (Γ, Δ) if

$$\sum_{g \in \mathcal{O}_D^{(k)}} \omega(g) = 1 \quad \text{and} \quad \sum_{g \in \mathcal{O}_D^{(k)}} \omega(g) \nu(g(x_1, \ldots, x_k)) \leq \frac{1}{k} \sum_{i=1}^{k} \nu(x_i)$$

holds for every $\nu \in \Delta$ and every $x_1, \ldots, x_k \in D$, and $\omega(g) = 0$ if $g \notin \mathrm{Pol}(\Gamma)$. For a k-ary fractional polymorphism ω we let $\mathrm{supp}(\omega) = \{g \in \mathcal{O}_D^{(k)} : \omega(g) > 0\}$. The set of all fractional polymorphisms of (Γ, Δ) is denoted fPol(Γ, Δ).

Min-Cores. The language (Γ, Δ) is called a *min-core* [12] if there is no non-surjective unary $f \in \mathrm{Pol}(\Gamma)$ for which $\nu(f(x)) \leq \nu(x)$ holds for every $x \in D$ and $\nu \in \Delta$. The language (Γ', Δ') is a min-core of (Γ, Δ) if (Γ', Δ') is a min-core and $(\Gamma, \Delta)|_{f(D)} = (\Gamma', \Delta')$ for some unary $f \in \mathrm{Pol}(\Gamma)$ satisfying $\nu(f(x)) \leq \nu(x)$ for every $x \in D$ and $\nu \in \Delta$. The reason why we care about min-cores is the following result [12].[2]

Theorem 1. *Let (Γ', Δ') be a min-core of (Γ, Δ). If Min-Cost-Hom(Γ', Δ') is NP-hard (in PO), then Min-Cost-Hom(Γ, Δ) is NP-hard (in PO).*

Expressive Power and Polynomial-Time Reductions. A relation R is said to be *weighted pp-definable* in (Γ, Δ) if there is an instance $I = (V, C, w)$ of

[2] The results in [12] are stated for a slightly more restricted problem than ours. It is however not hard to see that the results transfer to our setting.

Min-Cost-Hom(Γ, Δ) s.t. $R = \{(\varphi(v_1), \ldots, \varphi(v_n)) : \varphi \in \text{Optsol}(I)\}$ for some $v_1, \ldots, v_n \in V$. We use $\langle \Gamma, \Delta \rangle_w$ to denote the set of all relations that is weighted pp-definable in (Γ, Δ). Similarly R is said to be *pp-definable* in Γ if there is an instance $I = (V, C)$ of CSP(Γ) s.t. $R = \{(\varphi(v_1), \ldots, \varphi(v_n)) : \varphi \in \text{Sol}(I)\}$ for some $v_1, \ldots, v_n \in V$. $\langle \Gamma \rangle$ is used to denote the set of all relations that are pp-definable in Γ. A cost function $\nu : D \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ is called *expressible* in (Γ, Δ) if there is an instance $I = (V, C, w)$ of Min-Cost-Hom(Γ, Δ) and $v \in V$ s.t. $\nu(x) = \min\{m(\varphi) : \varphi \in \text{Sol}(I), \varphi(v) = x\}$ if $\nu(x) < \infty$ and $\min\{m(\varphi) : \varphi \in \text{Sol}(I), \varphi(v) = x\} = \infty$ or $\{\varphi \in \text{Sol}(I) : \varphi(v) = x\} = \emptyset$ if $\nu(x) = \infty$. The set of all cost functions expressible in (Γ, Δ) is denoted $\langle \Gamma, \Delta \rangle_e$. What makes all these closure operators interesting is the following result, see e.g. [3,4,10].

Theorem 2. *Let $\Gamma' \subseteq \langle \Gamma, \Delta \rangle_w$ and $\Delta' \subseteq \langle \Gamma, \Delta \rangle_e$ be finite sets. Then, Min-Cost-Hom(Γ', Δ') is polynomial-time reducible to Min-Cost-Hom(Γ, Δ).*

This of course also means that if $\Gamma' \subseteq \langle \Gamma, \Delta \rangle_w$ is finite, then Min-Cost-Hom$(\Gamma' \cup \Gamma, \Delta)$ is polynomial-time reducible to Min-Cost-Hom(Γ, Δ).

We will often use bipartite-graph-representations for relations, e.g. $\begin{smallmatrix} a & & b \\ & \diagdown\!\diagup & \\ b & & c \end{smallmatrix} = \{(a, b), (a, c), (b, b)\}$. Finally we recall a classic result, see e.g. [17, p. 94], about systems of linear equations that will be of great assistance.

Theorem 3 (Motzkin's Transposition Theorem). *For any $A \in \mathbb{Q}^{m \times n}$, $B \in \mathbb{Q}^{p \times n}$, $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^p$, exactly one of the following holds:*

- *$Ax \leq b$, $Bx < c$ for some $x \in \mathbb{Q}^n$*
- *$A^T y + B^T z = 0$ and ($b^T y + c^T z < 0$ or $b^T y + c^T z = 0$ and $z \neq 0$) for some $y \in \mathbb{Q}^m_{\geq 0}$ and $z \in \mathbb{Q}^p_{\geq 0}$*

3 Contributions

We let D denote the finite domain over which the language (Γ, Δ) is defined. To describe our results we need to introduce some definitions.

Definition 4 ((a, b)-dominating). *Let $a, b \in D$. A binary fractional polymorphism ω of (Γ, Δ) is called* (a, b)-dominating *if*

$$\sum_{g \in \mathcal{O}_D^{(2)}} \omega(g) \delta_{a, g(a,b)} \geq \frac{1}{2} > \sum_{g \in \mathcal{O}_D^{(2)}} \omega(g) \delta_{b, g(a,b)}.^3$$

The following is a generalisation of the concept of weak tournament pairs that was introduced in [19].

Definition 5 (generalised weak tournament pair). *Let $A \subseteq B \subseteq \binom{D}{2}$. A language (Γ, Δ) is said to admit a* generalised weak tournament pair *on (A, B) if there is a pair of binary functions $f_1, f_2 \in \text{Pol}(\Gamma)$ s.t. the following holds.*

[3] Here δ denotes the Kronecker delta function, i.e. $\delta_{i,j} = 1$ if $i = j$, otherwise $\delta_{i,j} = 0$.

- *For every* $\{a,b\} \in \binom{D}{2}$*;*
  1. *if* $\{a,b\} \notin B$ *then* $f_1|_{\{a,b\}}$ *and* $f_2|_{\{a,b\}}$ *are projections, and*
  2. *if* $\{a,b\} \in B \setminus A$ *then* $f_1|_{\{a,b\}}$ *and* $f_2|_{\{a,b\}}$ *are different idempotent, conservative and commutative operations.*
- *For any* $U \subseteq D$ *s.t.* $U \in \langle \Gamma \rangle$ *either no* $\{x,y\} \in A$ *satisfies* $\{x,y\} \subseteq U$*, or there is* $\{a,b\} \in A$ *s.t.* $U \setminus \{b\} \in \langle \Gamma \rangle$ *and* (Γ, Δ) *admits an* (a,b)*-dominating binary fractional polymorphism.*

The following definition is inspired by notation used in [18].

Definition 6. *For* $a,b \in D$ *we define* $\begin{smallmatrix} a \\ \uparrow \\ b \end{smallmatrix} = \{f \in \mathcal{O}_D^{(2)} : f(a,b) = f(b,a) = a\}$ *and* $\begin{smallmatrix} a \\ \downarrow \\ b \end{smallmatrix} = \{f \in \mathcal{O}_D^{(2)} : f(a,b) = f(b,a) = b\}$*. For* $x_1, \dots, x_m, y_1, \dots, y_m \in D$ *and* $\diamond_1, \dots, \diamond_m \in \{\uparrow, \downarrow\}$ *we define* $\begin{smallmatrix} x_1 & x_2 & & x_m \\ \diamond_1 & \diamond_2 & \cdots & \diamond_m \\ y_1 & y_2 & & y_m \end{smallmatrix} = \begin{smallmatrix} x_1 \\ \diamond_1 \\ y_1 \end{smallmatrix} \cap \begin{smallmatrix} x_2 \\ \diamond_2 \\ y_2 \end{smallmatrix} \cap \cdots \cap \begin{smallmatrix} x_m \\ \diamond_m \\ y_m \end{smallmatrix}$*, e.g.* $\begin{smallmatrix} a & c \\ \uparrow & \downarrow \\ b & d \end{smallmatrix} = \begin{smallmatrix} a \\ \uparrow \\ b \end{smallmatrix} \cap \begin{smallmatrix} c \\ \downarrow \\ d \end{smallmatrix}$*.*

We can now give names to some classes of languages that will be important.

Definition 7. *We say that a language* (Γ, Δ) *over* D *is of type*

- **GWTP** *(generalised weak tournament pair) if there is* $A, B \subseteq \binom{D}{2}$ *s.t.* (Γ, Δ) *admits a generalised weak tournament pair on* (A,B) *and,* $\mathrm{Pol}(\Gamma)$ *contains an idempotent ternary function* m *that is arithmetical on* $\binom{D}{2} \setminus B$ *and satisfies* $m(x,y,z) \in \{x,y,z\}$ *for every* $x,y,z \in D$ *s.t.* $|\{x,y,z\}| = 3$*,*
- **BSM** *(bisubmodular, see e.g. [4]) if* $D = \{a,b,c\}$*,* $2\nu(b) \leq \nu(a) + \nu(c)$ *for every* $\nu \in \Delta$*, and there are binary idempotent commutative operations* $\sqcap, \sqcup \in \mathrm{Pol}(\Gamma)$ *s.t.* $\sqcap \in \begin{smallmatrix} a & c \\ \downarrow & \downarrow \\ b & b \end{smallmatrix}$*,* $\sqcup \in \begin{smallmatrix} a & c \\ \uparrow & \uparrow \\ b & b \end{smallmatrix}$ *and* $a \sqcup c = a \sqcap c = b$*,*
- **GMC** *(generalised min-closed, see [9]) if there is* $f \in \mathrm{Pol}(\Gamma)$ *s.t. for every* $\nu \in \Delta$ *the following is true. For all* $a,b \in D$ *s.t.* $a \neq b$ *it holds that if* $\nu(f(a,b)) \geq \max(\nu(a), \nu(b))$*, then* $\nu(f(b,a)) < \min(\nu(a), \nu(b))$*, and for all* $a \in D$ *it holds that* $\nu(f(a,a)) \leq \nu(a)$*.*

Solving instances of Min-Cost-Hom expressed in languages of type **GWTP**, **BSM** and **GMC** can be done in polynomial time. This is demonstrated by the following results. We note that the first result describes a new tractable class while the following two are known cases.[4] A proof of Theorem 8 is given in Sect. 4.

Theorem 8. *If there is* $S \subseteq 2^D$ *s.t.* $CSP(\Gamma^c \cup S)$ *is in P and* $(\Gamma \cup S, \Delta)$ *is of type* **GWTP***, then Min-Cost-Hom*$(\Gamma \cup S, \Delta)$ *(and therefore also Min-Cost-Hom*(Γ, Δ)*) is in PO.*

Theorem 9 ([20, Corollary 6.1]). *If* (Γ, Δ) *is of type* **BSM***, then Min-Cost-Hom*(Γ, Δ) *is in PO.*

Theorem 10 ([9, Theorem 5.10]). *If* (Γ, Δ) *is of type* **GMC***, then Min-Cost-Hom*(Γ, Δ) *is in PO.*

[4] [9, Theorem 5.10] is stated for a slightly more restricted problem than ours. It is however not hard to see that the results transfer to our setting.

Instances expressed using languages of type **BSM** can, as proved in [20], be solved through a certain linear programming formulation. We note that this also holds for languages of type **GMC**. It is known that any language of type **GMC** must admit a min-set-function [16, Theorem 5.18]. From this it follows that also a symmetric fractional polymorphism of every arity must be admitted, and the claim follows from [20].

The tractability of languages of type **GWTP** on the other hand can not directly be explained by the results in [20]. It can e.g. be checked that the language $(\{\begin{smallmatrix} b \\ a \end{smallmatrix}\!\!\times\!\!\begin{smallmatrix} b \\ a \end{smallmatrix}\}, \{a \mapsto 0, b \mapsto 1\})$ is of type **GWTP**. This language does not admit any symmetric fractional polymorphism and is therefore not covered by the results in [20].

Often (as e.g. demonstrated by Theorem 8) the fact that a language admits an (a, b)-dominating binary fractional polymorphism can be useful for tractability arguments. Also the converse fact, that a language does not admit such a fractional polymorphism, can have useful consequences. An example of this is the following proposition, which will be used in the proofs of our main results.

Proposition 11. *Let $a, b \in D$, $a \neq b$. If (Γ, Δ) does not admit a binary fractional polymorphism that is (a, b)-dominating, then $\langle \Gamma, \Delta \rangle_e$ contains a unary function ν that satisfies $\infty > \nu(a) > \nu(b)$.*

The proof is given in Sect. 5.

3.1 Conservative Languages

We call (Γ, Δ) *conservative* if $2^D \subseteq \Gamma$, i.e. if the crisp language contains all unary relations. The complexity of Min-Cost-Hom(Γ, Δ) for conservative languages (Γ, Δ) was classified in [19] under the restriction that Δ contains only finite-valued functions, and that for each pair $a, b \in D$ there exists some $\nu \in \Delta$ s.t. either $\nu(a) < \nu(b)$ or $\nu(a) > \nu(b)$. It was posted in [18,19] as an open problem to classify the complexity of the problem also without restrictions on Δ. The following theorem does just that.

Theorem 12. *Let (Γ, Δ) be a conservative language on a finite domain. If CSP(Γ) is in P and (Γ, Δ) is of type* **GWTP***, then Min-Cost-Hom(Γ, Δ) is in PO, otherwise Min-Cost-Hom(Γ, Δ) is NP-hard.*

We prove the theorem in Sect. 6.

Kolmogorov and Živný [15] completely classified the complexity of conservative VCSPs. Since every Min-Cost-Hom can be stated as a VCSP, one might think that the classification provided here is implied by the results in [15]. This is not the case. A VCSP-language is called conservative if it contains all unary $\{0, 1\}$-valued cost functions. The conservative Min-Cost-Hom-languages on the other hand correspond to VCSP-languages that contain every unary $\{0, \infty\}$-valued cost function. (Note however that far from all VCSP-languages that contain every unary $\{0, \infty\}$-valued cost function correspond to a Min-Cost-Hom-language.)

3.2 Min-Sol on the Three-Element Domain

In this section we fully classify the complexity of Min-Sol on the three-element domain.

Theorem 13. *Let (Γ, ν) be a language over a three-element domain D and $\nu : D \to \mathbb{Q}_{\geq 0}$ be injective. If (Γ, ν) is a min-core and there is no $S \subseteq 2^D$ s.t. $(\Gamma \cup S, \nu)$ is of type* **GWTP**, **BSM** *or* **GMC**, *then Min-Sol(Γ, ν) is NP-hard.*

The following two lemmas provide key assistance in the proof of Theorem 13. The first of the two is a variation of Lemma 3.5 in [21].

Lemma 14. *If $\begin{smallmatrix} a & & a \\ & \diagdown\!\diagup & \\ b & & b \end{smallmatrix} \notin \langle \Gamma, \Delta \rangle_w$, then for every $\sigma \in \langle \Gamma, \Delta \rangle_e$ there is $\omega \in \mathrm{fPol}(\Gamma, \Delta)$ with $f \in \mathrm{supp}(\omega)$ s.t. $\{f(a,b), f(b,a)\} \neq \{a, b\}$ and $\sigma(f(a,b)) + \sigma(f(b,a)) \leq \sigma(a) + \sigma(b)$.*

Lemma 15. *Let (Γ, ν) be a language over a three-element domain D and $\nu : D \to \mathbb{Q}_{\geq 0}$ be injective. If (Γ, ν) is a min-core and not of type* **GMC**, *then $\Gamma^c \subseteq \langle \Gamma, \nu \rangle_w$.*

The proof of Theorem 13 contains a somewhat lengthy case-analysis and will not be presented in full. The interested reader can find it in the longer version of this paper. The case-analysis splits the proof into cases depending on what unary relations that are weighted pp-definable in (Γ, ν). In each case it is essentially shown that, unless a two-element subset $\{x, y\} \subseteq D$ is definable s.t. Min-Sol$(\Gamma \cup \langle \Gamma, \nu \rangle_w \cap \mathcal{O}_D^{(2)}, \nu)|_{\{x,y\}}$ is NP-hard, in which case also Min-Sol(Γ, ν) is NP-hard, the language $(\Gamma \cup S, \nu)$ is of type **GMC**, **BSM** or **GWTP** for some $S \subseteq 2^D$.

If $(\Gamma \cup S, \nu)$ is a min-core and of type **GWTP** (and not of type **GMC**), then from Lemma 15 it follows that CSP$(\Gamma^c \cup S) \leq_p$ Min-Sol$(\Gamma^c \cup S, \nu) \leq_p$ Min-Sol$(\Gamma \cup S, \nu)$. Since Min-Sol(Γ, ν) is a restricted variant of Min-Cost-Hom(Γ, Δ), we therefore, from Theorems 8, 9 10 and 13, obtain the following.

Theorem 16. *Let (Γ, ν) be a language over a three-element domain D and $\nu : D \to \mathbb{Q}_{\geq 0}$ be injective. Min-Sol(Γ, ν) is in PO if (Γ, ν) has a min-core (Γ', ν') that is of type* **BSM** *or* **GMC**, *or if there is $S \subseteq 2^D$ s.t. CSP$((\Gamma')^c \cup S)$ is in P and $(\Gamma' \cup S, \nu')$ is of type* **GWTP**. *Otherwise Min-Sol(Γ, ν) is NP-hard.*

The following provides an example of use of the classification. Jonsson, Nordh and Thapper [13] classified the complexity of Min-Sol$(\{R\}, \nu)$ for all valuations ν and binary symmetric relations R (i.e. graphs) on the three-element domain. One relation stood out among the others, namely: $H_5 = \{(a,c), (c,a), (b,b), (b,c), (c,b), (c,c)\}$, where $\nu(a) < \nu(b) < \nu(c)$. If $\nu(a) + \nu(c) < 2\nu(b)$ then $\mathrm{pr}_1(\arg\min_{(x,y) \in H_5}(\nu(x) + \nu(y))) = \{a, c\}$ which means that the relation $\begin{smallmatrix} c & & c \\ & \diagup\!\diagdown & \\ a & & a \end{smallmatrix} \in \langle \{H_5\}, \nu \rangle_w$, and Min-Sol$(\{H_5\}, \nu)$ is NP-hard by a reduction from the maximum independent set problem. Otherwise the problem is in PO. This was determined in [13] by linking the problem with, and generalising algorithms for, the critical independent set problem [22]. We note that $\sqcup, \sqcap \in \mathrm{Pol}(\{H_5\})$, where $\sqcup, \sqcap$ are commutative idempotent binary operations s.t. $\sqcap \in \begin{smallmatrix} a & c \\ \downarrow & \downarrow \\ b & b \end{smallmatrix}$, $\sqcup \in \begin{smallmatrix} a & c \\ \uparrow & \uparrow \\ b & b \end{smallmatrix}$ and $a \sqcap c = a \sqcup c = b$. This means that $(\{H_5\}, \nu)$ is of type **BSM**.

4 Proof of Theorem 8

Let $I = (V, C, w)$ be an instance of Min-Cost-Hom(Γ, Δ) with measure m. Since CSP(Γ^c) is in P we can, in polynomial-time, compute the reduced domain $D_v = \{\varphi(v) : \varphi \in \mathrm{Sol}(I)\}$ for every $v \in V$. Note that $D_v \in \langle\Gamma\rangle$.

Let f_1, f_2 be a generalised weak tournament pair on (A, B). If there for some $v \in V$ is some $\{x, y\} \in A$ s.t. $\{x, y\} \subseteq D_v$, then we know that there is $\{a, b\} \in A$ so that $D_v \setminus \{b\} \in \langle\Gamma\rangle$ and (Γ, Δ) admits an (a, b)-dominating binary fractional polymorphism ω. Assume that φ_a and φ_b are s.t. $m(\varphi_a) = \min\{m(\varphi) : \varphi \in \mathrm{Sol}(I), \varphi(v) = a\}$ and $m(\varphi_b) = \min\{m(\varphi) : \varphi \in \mathrm{Sol}(I), \varphi(v) = b\}$.

Certainly $g(\varphi_a, \varphi_b) \in \mathrm{Sol}(I)$ for every $g \in \mathrm{supp}(\omega)$. Because $\omega \in \mathrm{fPol}(\Gamma, \Delta)$ it follows that

$$\begin{aligned}\sum_{g\in\mathcal{O}_D^{(2)}} \omega(g)m(g(\varphi_a,\varphi_b)) &= \sum_{g\in\mathcal{O}_D^{(2)}} \omega(g) \sum_{x\in V,\nu\in\Delta} w(x,\nu)\nu(g(\varphi_a,\varphi_b)(x)) \\ &= \sum_{x\in V,\nu\in\Delta} w(x,\nu) \sum_{g\in\mathcal{O}_D^{(2)}} \omega(g)\nu(g(\varphi_a(x),\varphi_b(x))) \\ &\le \sum_{x\in V,\nu\in\Delta} w(x,\nu)\frac{1}{2}(\nu(\varphi_a(x)) + \nu(\varphi_b(x))) = \frac{1}{2}(m(\varphi_a) + m(\varphi_b)).\end{aligned}$$

Since ω is (a, b)-dominating there are functions $\varrho, \sigma : \mathcal{O}_D^{(2)} \to \mathbb{Q}_{\ge 0}$ s.t. $\omega = \varrho + \sigma$, $\sum_{g\in\mathcal{O}_D^{(2)}} \varrho(g) = \sum_{g\in\mathcal{O}_D^{(2)}} \sigma(g) = \frac{1}{2}$, $g(a, b) = a$ for every $g \in \mathrm{supp}(\varrho)$, and $f(a, b) \neq b$ for some $f \in \mathrm{supp}(\sigma)$. This implies that

$$\frac{1}{2}m(\varphi_a) + \sum_{g\in\mathcal{O}_D^{(2)}} \sigma(g)m(g(\varphi_a,\varphi_b)) \le \sum_{g\in\mathcal{O}_D^{(2)}} \omega(g)m(g(\varphi_a,\varphi_b)),$$

so

$$\sum_{g\in\mathcal{O}_D^{(2)}} 2\sigma(g)m(g(\varphi_a,\varphi_b)) \le m(\varphi_b),$$

which in turn (since $\sum_{g\in\mathcal{O}_D^{(2)}} 2\sigma(g) = 1$ and $f(a, b) \neq b$ for some $f \in \mathrm{supp}(\sigma)$) implies that there is $\varphi^* \in \mathrm{Sol}(I)$ s.t. $m(\varphi^*) \le m(\varphi_b)$ and $\varphi^*(v) \neq b$. Hence b can be removed from D_v without increasing the measure of an optimal solution. To accomplish this the constraint $(v, D_v \setminus \{b\})$ is added.

We repeat this procedure until $\binom{D_v}{2} \cap A = \emptyset$ for every $v \in V$. Clearly this takes at most $|D| \cdot |V|$ iterations.

Let $f_1' = f_1[f_1, \overline{f_1}]$ and $f_2' = f_2[f_2, \overline{f_2}]$. Note that $f_1'|_{\{x,y\}}$ and $f_2'|_{\{x,y\}}$ are different conservative, idempotent and commutative operations if $\{x, y\} \in B \setminus A$ and projections if $\{x, y\} \in \binom{D}{2} \setminus B$. If $f_1|_{\{x,y\}} = \mathrm{pr}_1$ for some $\{x, y\}$, then $f_1'|_{\{x,y\}} = f_1|_{\{x,y\}} = \mathrm{pr}_1$, and if $f_1|_{\{x,y\}} = \mathrm{pr}_2$, then $f_1'|_{\{x,y\}} = \overline{f_1}|_{\{x,y\}} = \overline{\mathrm{pr}_2} = \mathrm{pr}_1$. So $f_1'|_{\{x,y\}} = \mathrm{pr}_1$ for every $\{x, y\} \in \binom{D}{2} \setminus B$. The same arguments apply also for f_2'.

Clearly $f_1'|_{D_v}$ and $f_2'|_{D_v}$ are conservative operations for every $v \in V$. Let $g \in \mathrm{Pol}(\Gamma)$ be a ternary idempotent operation that is arithmetical on $\binom{D}{2} \setminus B$. Define g' through $g'(x, y, z) = g(f_1'(x, f_1'(y, z)), f_1'(y, f_1'(x, z)), f_1'(z, f_1'(x, y)))$. Since $f_1' = \mathrm{pr}_1$ on $\binom{D}{2} \setminus B$ also g' is arithmetical on $\binom{D}{2} \setminus B$. Since f_1' is conservative,

commutative and idempotent on $B \setminus A$ we have $f_1'(x, f_1'(x,y)) = f_1'(x, f_1'(y,x)) = f_1'(y, f_1'(x,x)) \in \{x,y\}$ for every $\{x,y\} \in B \setminus A$, so g' is conservative on $\binom{D}{2} \setminus A$. Note that $f_1', f_2', g' \in \mathrm{Pol}(\Gamma^+)$ where $\Gamma^+ = \Gamma \cup \{S : S \subseteq D_v \text{ for some } v \in V\}$). This together with the fact that only a constant number of subsets of D exists means that the modified instance I is easily turned into an instance of the multi-sorted version of Min-Cost-Hom(Γ^+, ∇_D), where ∇_D is the set of all functions $D \to \mathbb{N}$, and is solvable in polynomial time [19, Theorem 23].

5 Proof of Proposition 11

Let $\Omega = \mathcal{O}_D^{(2)} \cap \mathrm{Pol}(\Gamma)$, $\Omega_1 = \{f \in \Omega : f(a,b) = a\}$, $\Omega_2 = \{f \in \Omega : f(a,b) = b\}$ and $\Omega_3 = \Omega \setminus (\Omega_1 \cup \Omega_2)$. The language (Γ, Δ) admits a binary fractional polymorphism that is (a,b)-dominating if the following system has a solution $u_g \in \mathbb{Q}$, $g \in \Omega$.

$$\sum_{g \in \Omega} u_g \nu(g(x,y)) \le \frac{1}{2}(\nu(x) + \nu(y)) \text{ for } (x,y) \in D^2, \nu \in \Delta, \quad -u_g \le 0 \text{ for } g \in \Omega,$$
$$\sum_{g \in \Omega} u_g \le 1, \quad -\sum_{g \in \Omega} u_g \le -1, \quad -\sum_{g \in \Omega_1} u_g \le -\frac{1}{2}, \quad \text{and} \quad \sum_{g \in \Omega_2} u_g < \frac{1}{2}$$

If the system is unsatisfiable, then, by Theorem 3, there are $v_{(x,y),\nu}$, o_g, w_1, w_2, w_3, $z \in \mathbb{Q}_{\ge 0}$ for $(x,y) \in D^2, \nu \in \Delta, g \in \Omega$ s.t.

$$\sum_{(x,y) \in D^2, \nu \in \Delta} \nu(g(x,y)) v_{(x,y),\nu} - o_g + w_1 - w_2 - w_3 = 0, \qquad g \in \Omega_1,$$
$$\sum_{(x,y) \in D^2, \nu \in \Delta} \nu(g(x,y)) v_{(x,y),\nu} - o_g + w_1 - w_2 + z = 0, \qquad g \in \Omega_2,$$
$$\sum_{(x,y) \in D^2, \nu \in \Delta} \nu(g(x,y)) v_{(x,y),\nu} - o_g + w_1 - w_2 = 0, \qquad g \in \Omega_3,$$
$$\text{and} \quad \sum_{(x,y) \in D^2, \nu \in \Delta} \frac{1}{2}(\nu(x) + \nu(y)) v_{(x,y),\nu} + w_1 - w_2 - \frac{1}{2} w_3 + \frac{1}{2} z = \alpha,$$

where either $\alpha < 0$ or $\alpha = 0$ and $z > 0$. Hence, for every $g \in \Omega_1$ and $h \in \Omega_2$,

$$\sum_{(x,y) \in D^2, \nu \in \Delta} (\nu(x) + \nu(y)) v_{(x,y),\nu} + o_g + o_h = \sum_{(x,y) \in D^2, \nu \in \Delta} (\nu(g(x,y)) + \nu(h(x,y))) v_{(x,y),\nu} + \alpha.$$

Note that since $\mathrm{pr}_1 \in \Omega_1$ and $\mathrm{pr}_2 \in \Omega_2$ we must have $\alpha = 0$, $o_{\mathrm{pr}_1} = o_{\mathrm{pr}_2} = 0$, and $z > 0$. This means that

$$\min_{\substack{g \in \Omega_1 \\ (x,y) \in D^2, \nu \in \Delta}} \sum \nu(g(x,y)) v_{(x,y),\nu} = \sum_{(x,y) \in D^2, \nu \in \Delta} \nu(\mathrm{pr}_1(x,y)) v_{(x,y),\nu} = -w_1 + w_2 + w_3$$
$$> -w_1 + w_2 - z = \sum_{(x,y) \in D^2, \nu \in \Delta} \nu(\mathrm{pr}_2(x,y)) v_{(x,y),\nu} = \min_{\substack{g \in \Omega_2 \\ (x,y) \in D^2, \nu \in \Delta}} \sum \nu(g(x,y)) v_{(x,y),\nu}.$$

Create an instance I of Min-Cost-Hom(Γ, Δ) with variables D^2 and measure $m(\varphi) = \sum_{(x,y)\in D^2, \nu\in\Delta} v_{(x,y),\nu}\nu(\varphi(x,y))$. Pick, for every $g \in \mathcal{O}_D^{(2)} \setminus \Omega$, a relation $R_g \in \Gamma$ s.t. g does not preserve R_g. Add for each pair of tuples $t^1, t^2 \in R_g$ the constraint $(((t_1^1, t_1^2), \ldots, (t^1_{\mathrm{ar}(R_g)}, t^2_{\mathrm{ar}(R_g)})), R_g)$. This construction is essentially the second order indicator problem [8].

A solution to I is a function $\varphi : D^2 \to D$ that by construction is a binary polymorphism of Γ, i.e. $\varphi \in \Omega$. Clearly pr_1 and pr_2 satisfies all constraints and are therefore solutions to I. Let $\nu(x) = \min_{g\in\mathrm{Sol}(I):g(a,b)=x} m(g)$. Note that $\nu \in \langle \Gamma, \Delta\rangle_e$ and $\infty > \nu(a) > \nu(b)$. This completes the proof.

6 Proof of Theorem 12

The proof follows the basic structure of the arguments given in [18]. A key ingredient of our proof will be the use of Theorem 8 and Proposition 11.

Let $\Gamma^+ = \Gamma \cup (2^D \cup 2^{D^2}) \cap \langle \Gamma, \Delta\rangle_w$. Note that if (Γ^+, Δ) is of type **GWTP**, then so is also (Γ, Δ). Since Min-Cost-Hom(Γ^+, Δ) is polynomial-time reducible to Min-Cost-Hom(Γ, Δ) we therefore assume that $\Gamma^+ \subseteq \Gamma$. We also assume $\{c\} \in \Gamma$ for every $c \in D$. Obviously CSP(Γ) is polynomial-time reducible to Min-Cost-Hom(Γ, Δ). In what follows we therefore assume that CSP(Γ) is in P.

Let $B \subseteq \binom{D}{2}$ be a minimal set s.t. for every $\{a,b\} \in \binom{D}{2} \setminus B$ there is a ternary operation in Pol(Γ) that is arithmetical on $\{a,b\}$ and all binary operations in Pol(Γ) are projections on $\{a,b\}$. Then, let A be a maximal subset of B s.t. for every $\{a,b\} \in A$ there is $\omega \in \mathrm{fPol}(\Gamma, \Delta)$ s.t. ω is either (a,b)-dominating or (b,a)-dominating. Let T be the undirected graph (M, P), where $M = \{(a,b) : \{a,b\} \in \Gamma \cap B \setminus A\}$ and $P = \{((a,b),(c,d)) \in M^2 : \mathrm{Pol}(\Gamma) \cap \begin{smallmatrix} a & c \\ \downarrow & \downarrow \\ b & d \end{smallmatrix} = \emptyset\}$.

By Proposition 11 we know that for every $(a,b) \in M$, there are $\nu, \tau \in \langle \Gamma, \Delta\rangle_e$ s.t. $\nu(b) < \nu(a) < \infty$ and $\tau(a) < \tau(b) < \infty$. By the classification of Min-Cost-Hom on two-element domains, see e.g. [18, Theorem 3.1], and by the fact that if $f, m \in \mathrm{Pol}(\Gamma)$ are idempotent, m is arithmetical on $\{\{x,y\}\}$ and $f \in \begin{smallmatrix} x \\ \uparrow \\ y \end{smallmatrix}$, then $m'(u,v) = m(u, f(u,v), v)$ satisfies $m' \in \begin{smallmatrix} x \\ \downarrow \\ y \end{smallmatrix}$, we have the following.

Lemma 17. *Either; for every $(a,b) \in M$ there are $f, g \in \mathrm{Pol}(\Gamma)$ s.t. $f|_{\{a,b\}}$ and $g|_{\{a,b\}}$ are two different idempotent, conservative and commutative operations, or Min-Cost-Hom(Γ, Δ) is NP-hard.*

Lemma 18 ([18, Theorem 5.3]). *If T is bipartite, then there are $f, g \in \mathrm{Pol}(\Gamma)$ s.t. for every $(a,b) \in M$, $f|_{\{a,b\}}$ and $g|_{\{a,b\}}$ are different idempotent conservative and commutative operations, or Min-Cost-Hom(Γ, Δ) is NP-hard.*

Lemma 19 ([18, Theorem 5.4]). *Let $C \subseteq \binom{D}{2}$. If $C \subseteq \Gamma$ and for each $\{a,b\} \in C$ there is a ternary operation $m^{\{a,b\}} \in \mathrm{Pol}(\Gamma)$ that is arithmetical on $\{\{a,b\}\}$, then there is $m \in \mathrm{Pol}(\Gamma)$ that is arithmetical on C.*

So, if T is bipartite and (Γ, Δ) is conservative, there is a generalised weak tournament pair on (A, B) and an arithmetical polymorphism on $\binom{D}{2} \setminus B$. Here (Γ, Δ) is of type **GWTP**, and by Theorem 8, we can conclude that Min-Cost-Hom(Γ, Δ) is polynomial-time solvable.

This following lemma finishes the proof of Theorem 12. A corresponding result, for the case when Δ is the set of all functions $D \to \mathbb{N}$, is also achieved in [18]. Our proof strategy is somewhat different from that in [18], though.

Lemma 20. *If T is not bipartite, then Min-Cost-Hom(Γ, Δ) is NP-hard.*

Proof. We will show that if T is not bipartite, then $\begin{smallmatrix}b\\a\end{smallmatrix}\!\!\bigtimes\!\!\begin{smallmatrix}b\\a\end{smallmatrix} \in \langle\Gamma\rangle$ for some $(a,b) \in M$. From this it follows, using Lemma 17, that Min-Cost-Hom(Γ, Δ) is NP-hard. We will make use of the following result.

Lemma 21 ([18, Lemma 4.2]). *If $((a,b),(c,d)) \in P$, then either $\begin{smallmatrix}a\\b\end{smallmatrix}\!\!\bigtimes\!\!\begin{smallmatrix}c\\d\end{smallmatrix} \in \langle\Gamma\rangle$ or $\begin{smallmatrix}a\\b\end{smallmatrix}\!\!\bigtimes\!\!\begin{smallmatrix}c\\d\end{smallmatrix} \in \langle\Gamma\rangle$.*

Since $\Gamma^+ \subseteq \Gamma$, and since there are functions $\nu, \tau \in \langle\Gamma, \Delta\rangle_e$ s.t. $\nu(b) < \nu(a) < \infty$ and $\tau(d) < \tau(c) < \infty$, we immediately get the following.

Corollary 22. *If $((a,b),(c,d)) \in P$, then $\begin{smallmatrix}a\\b\end{smallmatrix}\!\!\bigtimes\!\!\begin{smallmatrix}c\\d\end{smallmatrix} \in \Gamma$.*

Since T is not bipartite it must contain an odd cycle (a_0,b_0), (a_1,b_1), …, (a_{2k},b_{2k}), (a_0,b_0). This means, according to Corollary 22, that Γ contains relations $\varrho_{0,1}$, $\varrho_{1,2}$, …, $\varrho_{2k-1,2k}$, $\varrho_{2k,0}$ where $\varrho_{i,j} = \begin{smallmatrix}a_i\\b_i\end{smallmatrix}\!\!\bigtimes\!\!\begin{smallmatrix}a_j\\b_j\end{smallmatrix}$. Since the cycle is odd this means that $\varrho_{0,1} \circ \varrho_{1,2} \circ \cdots \circ \varrho_{2k-1,2k} \circ \varrho_{2k,0} = \begin{smallmatrix}a_0\\b_0\end{smallmatrix}\!\!\bigtimes\!\!\begin{smallmatrix}a_0\\b_0\end{smallmatrix} \in \langle\Gamma\rangle$. □

7 Concluding Remarks

We have fully classified the complexity of Min-Sol on domains that contain at most three elements and the complexity of conservative Min-Cost-Hom on arbitrary finite domains.

Unlike for CSP there is no widely accepted conjecture for the complexity of VCSP. This makes the study of small-domain VCSPs an exciting and important task. We believe that a promising approach for this project is to study Min-Cost-Hom — it is interesting for its own sake and likely easier to analyse than the general VCSP.

A natural continuation of the work presented in this paper would be to classify Min-Cost-Hom on domains of size three. This probably is a result within reach using known techniques. Another interesting question is what the complexity of three-element Min-Sol is when the domain valuation is not injective (we note that if the valuation is constant the problem collapses to a CSP whose complexity has been classified by Bulatov [1], but situations where e.g. $\nu(a) = \nu(b) < \nu(c)$ are not yet understood).

Acknowledgements. I am thankful to Peter Jonsson for rewarding discussions and to Magnus Wahlström for helpful comments regarding the presentation of the results. I am also grateful to the anonymous reviewers for their useful feedback.

References

1. Bulatov, A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. J. ACM 53(1), 66–120 (2006)

2. Bulatov, A., Jeavons, P., Krokhin, A.: Classifying the complexity of constraints using finite algebras. SIAM J. Comput. 34(3), 720–742 (2005)
3. Cohen, D.A., Cooper, M.C., Jeavons, P.G.: An algebraic characterisation of complexity for valued constraint. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 107–121. Springer, Heidelberg (2006)
4. Cohen, D.A., Cooper, M.C., Jeavons, P.G., Krokhin, A.: The complexity of soft constraint satisfaction. Artif. Intell. 170(11), 983–1016 (2006)
5. Cohen, D.A., Creed, P., Jeavons, P.G., Živný, S.: An algebraic theory of complexity for valued constraints: Establishing a galois connection. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 231–242. Springer, Heidelberg (2011)
6. Gutin, G., Rafiey, A., Yeo, A., Tso, M.: Level of repair analysis and minimum cost homomorphisms of graphs. Discrete Appl. Math. 154(6), 881–889 (2006)
7. Jeavons, P.G., Cohen, D.A., Gyssens, M.: Closure properties of constraints. J. ACM 44(4), 527–548 (1997)
8. Jeavons, P.G., Cohen, D.A., Gyssens, M.: How to determine the expressive power of constraints. Constraints 4, 113–131 (1999)
9. Jonsson, P., Kuivinen, F., Nordh, G.: MAX ONES generalized to larger domains. SIAM J. Comput. 38(1), 329–365 (2008)
10. Jonsson, P., Kuivinen, F., Thapper, J.: Min CSP on four elements: Moving beyond submodularity. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 438–453. Springer, Heidelberg (2011)
11. Jonsson, P., Nordh, G.: Introduction to the Maximum Solution problem. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 255–282. Springer, Heidelberg (2008)
12. Jonsson, P., Nordh, G.: Approximability of clausal constraints. Theor. Comput. Syst. 46(2), 370–395 (2010)
13. Jonsson, P., Nordh, G., Thapper, J.: The maximum solution problem on graphs. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 228–239. Springer, Heidelberg (2007)
14. Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P.: The approximability of constraint satisfaction problems. SIAM J. Comput. 30(6), 1863–1920 (2001)
15. Kolmogorov, V., Živný, S.: The complexity of conservative valued CSPs. In: Proc. 23rd SODA, pp. 750–759 (2012); Full version: `arXiv:1110.2809 [cs.CC]`
16. Kuivinen, F.: Algorithms and Hardness Results for Some Valued CSPs. PhD thesis, Linköping University, The Institute of Technology (2009)
17. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Inc., New York (1986)
18. Takhanov, R.: A dichotomy theorem for the general minimum cost homomorphism problem. In: Proc. 27th STACS, pp. 657–668 (2010); Full version: `arXiv:0708.3226 [cs.LG]`
19. Takhanov, R.: Extensions of the minimum cost homomorphism problem. In: Thai, M.T., Sahni, S. (eds.) COCOON 2010. LNCS, vol. 6196, pp. 328–337. Springer, Heidelberg (2010); Full version: `arXiv:1210.2260 [cs.CC]`
20. Thapper, J., Živný, S.: The power of linear programming for valued CSPs. In: Proc. 53rd FOCS (2012) (to appear); Preprint: `arXiv:1204.1079 [cs.CC]`
21. Thapper, J., Živný, S.: The complexity of finite-valued CSPs. In: Proc. 45th STOC (to appear, 2013); Preprint: `arXiv:1210.2987 [cs.CC]`
22. Zhang, C.Q.: Finding critical independent sets and critical vertex subsets are polynomial problems. SIAM J. Discrete Math. 3(3), 431–438 (1990)

The Complexity of Infinitely Repeated Alternating Move Games*

Yaron Velner

The Blavatnik School of Computer Science, Tel Aviv University, Israel

Abstract. We consider infinite duration alternating move games. These games were previously studied by Roth, Balcan, Kalai and Mansour [10]. They presented an FPTAS for computing an approximate equilibrium, and conjectured that there is a polynomial algorithm for finding an exact equilibrium [9]. We extend their study in two directions: (1) We show that finding an exact equilibrium, even for two-player zero-sum games, is polynomial time equivalent to finding a winning strategy for a (two-player) mean-payoff game on graphs. The existence of a polynomial algorithm for the latter is a long standing open question in computer science. Our hardness result for two-player games suggests that two-player alternating move games are harder to solve than two-player simultaneous move games, while the work of Roth et al., suggests that for $k \geq 3$, k-player games are easier to analyze in the alternating move setting. (2) We show that optimal equilibria (with respect to the social welfare metric) can be obtained by pure strategies, and we present an FPTAS for computing a pure approximated equilibrium that is δ-optimal with respect to the social welfare metric. This result extends the previous work by presenting an FPTAS that finds a much more desirable approximated equilibrium. We also show that if there is a polynomial algorithm for mean-payoff games on graphs, then there is a polynomial algorithm that computes an optimal exact equilibrium, and hence, (two-player) mean-payoff games on graphs are inter-reducible with k-player alternating move games, for any $k \geq 2$.

1 Introduction

In this work, we investigate infinitely repeated games in which players *alternate* making moves. This framework can model, for example, five telecommunication providers competing for customers: each company can observe the price that is set by the others, and it can update the price at any time. In the short term, each company can benefit from undercutting its opponents price, but since the game is repeated indefinitely, in some settings, it might be better to coordinate prices with the other companies. Such examples motivate us to study equilibria in alternating move games.

In this work, we study infinitely repeated k-player n-action games. In such games, in every *round*, a player chooses an action, and the utility of each player

* A fuller version is available at `http://arxiv.org/abs/1212.6632`

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 816–827, 2013.

(for the current round) is determined according to the k-tuple of actions of the players. Each player goal is to maximize his own *long-run average* utility as the number of rounds tends to infinity.

These games were studied by Roth et al. in [10], and they showed an FPTAS for computing an ϵ-equilibrium. Their result provided a theoretical separation between the alternating move model and the simultaneous move model, since for the latter, it is known that an FPTAS for computing approximate equilibria does not exists for games with $k \geq 3$ players unless P=PPAD. Their result was obtained by a simple reduction to mean-payoff games on graphs. These games were presented in [5], and they play an important rule in automata theory and in economics. The computational complexity of finding an exact equilibrium for such games is a long standing open problem, and despite many efforts [1–3, 6, 7, 12], there is no known polynomial solution for this problem.

We extend the work in [10] by investigating the complexity of an exact equilibrium (which was stated as an open question in [10]), and by investigating the computational complexity of finding an *δ-optimal* approximated equilibrium with respect to the *social welfare* metric. Our main technical results are as follows:

- We show a reduction from mean-payoff games on graphs to two-player zero-sum alternating move games, and thus we prove that k-player alternating move games are computationally equivalent to mean-payoff games on graphs for any $k \geq 2$.
- We show that optimal equilibrium can be obtained by pure strategies, and we show an FPTAS for computing an δ-optimal ϵ-equilibrium. In addition, we show that computing an exact optimal equilibrium is polynomial time equivalent to solving mean-payoff games on graphs.

We note that the first result may suggest that two-player alternating move games are harder than two-player simultaneous move games, since a polynomial time algorithm to solve the latter is known [8]. Hence, along with the result of [10], we get that simultaneous move games are easier to solve with comparison to alternating move games for two-player games, and are harder to solve for $k \geq 3$ player games.

This paper is organized as following. In the next section we bring formal definitions for alternating move games and mean-payoff games on graphs. In Section 3, we show that alternating move games are at least as hard as mean-payoff games on graphs. In Section 4 we investigate the properties of optimal equilibria, and we present an FPTAS for computing an δ-optimal ϵ-equilibrium. Due to lack of space, some of the proofs were omitted.

2 Definitions

In this section we bring the formal definitions for alternating move repeated games and mean-payoff games on graphs. Alternating move games are presented in Subsection 2.1, and mean-payoff games are presented in Subsection 2.2.

2.1 Alternating Move Repeated Games

Actions, Plays and Utility Function. A k-player n-action game is defined by an action set A_i for every player i, and by k utility functions, one for each player, $u_i : A_1 \times \dots A_k \to [-1, 1]$. W.l.o.g we assume that the size of all action set is the same, and we denote it by n. We note that any game can be rescaled so its utilities are bounded in $[-1, 1]$, however, the FPTAS that was presented in [10], and our results in Subsection 4.4 crucially rely on the assumption that the utilities are in the interval $[-1, 1]$.

An alternating move game is played for infinitely many rounds. In round t player $j = 1 + (t \mod k)$ plays action a_j^t, and a *vector of actions* $a^t = (a_1, \dots, a_k)$ is produced, where $a_i \in A_i$ is the last action of player i. In every round t, player i receives a utility $u_i(a^t)$, which depends only in the last action of each of the k players (W.l.o.g the utility in the first k rounds is zero for all players). A sequence of infinite rounds forms a *play*, and we characterize a play either by an infinite sequence of actions or by the corresponding sequence of vectors of actions. The utility of player i in a play $a^1 a^2 \dots a^t a^{t+1} \dots$ is the *limit average payoff*, namely, $\lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} u_i(a^t)$. When this limit does not exist, we define the utility of the play for player i to be $\liminf_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} u_i(a^t)$. We note that in the frame work of [10], the utility of a play was undefined when the limit does not exist. The results we present in this paper for the lim inf metric holds also for the framework of [10]. On the other hand, if we would take the lim sup value instead, then the problem is much easier. An optimal equilibrium is obtained when all players join forces and maximize player $1 + (i \pmod k)$ utility for 2^i rounds (for $i = 1, 2, \dots, \infty$). Hence, we can easily produce a polynomial algorithm to solve these games.

Strategies. A *strategy* is a recipe for player's next action, based on the entire *history* of previous actions. Formally, a (mixed) strategy for player i is a function $\sigma_i : (A_1 \times \dots \times A_k)^* \times A_1 \times \dots \times A_{i-1} \to \Delta(A_i)$, where $\Delta(S)$ denotes the set probability distribution over any finite set S. We say that σ_i is a *pure strategy* if Δ is a degenerated distribution. A *strategy profile* is a vector $\sigma = (\sigma_1, \dots, \sigma_k)$ that defines a strategy for every player. A profile of pure strategies uniquely determines the action vector in every round and yields a utility vector for the players. A profile of mixed strategies determines, for every round t in the play, a distribution of sequences of action vectors, and the *average payoff* in round t is the expected average payoff over the distribution of action vectors. Formally, for a strategy profile σ we denote the average payoff of player i in round t by

$$P_{i,t}(\sigma) = E_{a^1 a^2 \dots a^t \sim \sigma}[\frac{u_i(a^1) + \dots + u_i(a^t)}{t}]$$

and the utility of player i is $\liminf_{t\to\infty} P_{i,t}$.

Equilibria, ϵ Equilibria and Optimal Equilibria. A strategy profile forms an *equilibrium* if none of the players can strictly improve his utility (that is induced by the profile) by unilaterally deviating from his strategy (that is defined

by the profile). For every $\epsilon > 0$, we say that a strategy profile forms an ϵ-*equilibrium* if none of the players can improve his utility by more than ϵ by unilaterally deviating from his strategy.

The *social welfare* of a strategy profile is the sum of the utilities of all players. An equilibrium (resp. an ϵ-equilibrium) is called an *optimal equilibrium* (optimal ϵ-equilibrium) if its social welfare is not smaller than the social welfare of any other equilibrium (ϵ-equilibrium). For $\delta > 0$, an equilibrium (resp. an ϵ-equilibrium) is called an δ-*optimal equilibrium* if its social welfare is not smaller by more than δ with comparison to the social welfare of any other equilibrium (ϵ-equilibrium).

2.2 Mean-Payoff Games on Graphs

Plays and Payoffs. A *mean-payoff game* on a graph is defined by a weighted directed bipartite graph $G = (V = V_1 \cup V_2, E, w : E \to \mathbb{Q})$ and an initial vertex $v_0 \in V$. The game consists of two players, namely, maximizer (who owns V_1) and minimizer (who owns V_2). Initially, a pebble is place on the initial vertex, and in every round, the player who owns the vertex in which the pebble resides, advance the pebble into an adjacent vertex. This process is repeated forever and forms a *play*. A play is characterized by a sequence of edges, and the *average payoff* of a play $\rho = e_1 \dots e_t$ up to round t is denoted by $P_t = \frac{1}{t} \sum_{i=1} w(e_i)$. The *value* of a play is the limit average payoff (mean-payoff), namely, $\liminf_{t\to\infty} P_t$. (We note that for games on graphs, the lim sup metric gives the same complexity results.) The objective of the maximizer is to maximize the mean-payoff of a play, and the minimizer aims to minimize the mean-payoff.

Strategies, Memoryless Strategies, Optimal Strategies and Winning Strategies. In this work, we consider only pure strategies for games on graphs, and it is well-known that randomization does not give better strategies for mean-payoff games. A *strategy* for maximizer is a function $\sigma : (V_1 \times V_2)^* \times V_1 \to E$ that decides the next move, and similarly, for the minimizer a strategy is a function $\tau : (V_1 \times V_2)^* \to E$. A strategy is called *memoryless* if it depends only on the current position of the pebble. Formally, a memoryless strategy for the maximizer is a function $\sigma : V_1 \to E$ and similarly a memoryless strategy for the minimizer is a function $\tau : V_2 \to E$.

A profile of strategies (σ, τ) uniquely determines the mean-payoff value of a game. We say that a play $\pi = e_1 e_2 \dots e_n \dots$ is *consistent* with a maximizer strategy σ if there exists a minimizer strategy τ such that π is formed by (σ, τ). We say that the *value* of a maximizer strategy is p if it can assure a value of at least p against any minimizer strategy. Analogously, we say that the value of a minimizer strategy is p if it can assure a value of at most p against any maximizer strategy.

We say that a maximizer strategy is *optimal* if its value is maximal (with respect to all possible maximizer strategies). Analogously, a minimizer strategy is optimal if its value is minimal. For a given threshold, we say that a maximizer strategy is a *winning strategy* if it assures mean-payoff value that is greater or equal to the given threshold, and a minimizer strategy is winning if it assures value that is strictly smaller than the given threshold.

One-Player Games, and Games According to Memoryless Strategies. A special (and easier) case of games on graphs is when the out-degree is one for all the vertices that are owned by a certain player. In this case, all the *choices* are done by one player. For a two-player game on graph G, and a player-1 strategy σ, we define the one-player game graph G^σ to be the game graph that is formed by removing, for every player-1 vertex v, the out-edges that are not equal to $\sigma(v)$.

Classical Results on Mean-Payoff Games. Mean-payoff games were introduced in '79 by Ehrenfeucht and Mycielski [5], and their main result was that optimal strategies (for both players) exist, and moreover, the optimal value can be obtained by a memoryless strategy. The decision problem for mean-payoff games is to determine if the maximizer has a winning strategy with respect to a given threshold. The existence of optimal memoryless strategies almost immediately proves that the decision problem for mean-payoff games is in NP∩coNP, and thus it is unlikely to be NP-hard (or coNP-hard). Zwick and Paterson [12] introduced the first pseudo-polynomial algorithm, which runs in polynomial time when the weights of the edges are encoded in unary. They also provided a polynomial algorithm for the special case of one-player mean-payoff games. A randomized sub-exponential algorithm for mean-payoff games is also known [2], but despite many efforts, the existence of a polynomial algorithm to solve mean-payoff games remains an open question, and it is one of the rare problems in computer science that is known to be in NP∩coNP but no polynomial algorithm is known.

We summarize the known results on mean-payoff games in the next theorems. The first theorem states that optimal strategies exist and moreover, there exist optimal strategies that are memoryless.

Theorem 1 ([5]). *For every mean-payoff game there exists a maximizer memoryless strategy σ and a minimizer memoryless strategy τ such that σ is optimal for the maximizer and τ is optimal for the minimizer.*

The next theorem shows that there is a polynomial algorithm that computes optimal strategies if and only if there is a polynomial algorithm for the mean-payoff games decision problem.

Theorem 2 ([12]). *The following problems are polynomial time inter-reducible: (i) Compute maximizer optimal memoryless strategy. (ii) Compute the optimal value that maximizer can assure. (iii) Determine whether maximizer optimal value is at least zero. (iv) Determine whether maximizer optimal value is greater than zero.*

3 Two-Player Zero-Sum (Alternating Move) Games Are Inter-reducible with Mean-Payoff Games

In this section we prove that there is a polynomial algorithm that computes an exact equilibrium for two-player zero-sum (alternating move) games if and only if there exists a polynomial algorithm that solves mean-payoff games.

The reduction from two-player zero-sum games to mean-payoff games is trivial, for a two-player zero-sum game with actions A_1, A_2 and utility functions $u_1 : A_1 \times A_2 \to \mathbb{Q}$ and $u_2 = -u_1$, we construct a complete bipartite game graph $G = (V = A_1 \cup A_2, E = (A_1 \times A_2) \cup (A_2 \times A_1), w : E \to \mathbb{Q})$ such that the weight of the transition from $a_1 \in A_1$ to $a_2 \in A_2$ is simply $u_1(a_1, a_2)$, and the weight of the transition from a_2 to a_1 is also $u_1(a_1, a_2)$. It is a simple observation that a pair of optimal strategies (for the maximizer and minimizer) in the mean-payoff game induces an equilibrium strategy profile in the two-player zero-sum game and vice versa.

The reduction for the converse direction is more complicated. For this purpose we bring the notion of *undirected game graph*. A mean-payoff game graph is said to be *undirected* if its edge relation is symmetric, and $w(v_1, v_2) = w(v_2, v_1)$ for every edge (v_1, v_2). (Basically, it is a game on an undirected graph.) The next simple lemma shows a reduction from mean-payoff games on a complete bipartite undirected graphs to two-player zero-sum game.

Lemma 1. *There is a polynomial reduction from mean-payoff games on complete bipartite undirected graphs to two-player zero-sum games.*

Proof. The proof is straight forward. Let V_1 and V_2 be the maximizer and minimizer (resp.) vertices in the mean-payoff game. We construct a two-player zero-sum game in the following way. The set of action of player 1 is $A_1 = V_2$ and the set of action for player 2 is $A_2 = V_1$. We denote by W the least value for which all the weights in the undirected graphs are in $[-W, +W]$, and the utility function of player 1 is $u_1(a_1, a_2) = \frac{w(a_1,a_2)}{W} = \frac{w(a_2,a_1)}{W}$, and $u_2 = -u_1$. It is trivial to observe that an equilibrium profile induces a pair of optimal strategies for the mean-payoff game, and the proof follows. □

Due to Lemma 1, all that is left is to prove that mean-payoff games on complete bipartite undirected graphs are equivalent to mean-payoff games. A recent result by Chatterjee, Henzinger, Krinninger and Nanongkai [4] gives us the first step towards such proof.

Theorem 3 (Corollary 24 in [4]). *Solving mean-payoff games on complete bipartite (directed) graphs is as hard as solving mean-payoff games on arbitrary graphs.*

We use the above result as a black box and extend it to complete bipartite undirected graphs. We note that the main difference between directed and undirected graphs is that for undirected graphs the weight function is symmetric. In the rest of this section we will describe a process that for a given complete bipartite directed graph, generates a suitable symmetric weight function, and the winner in the generated graph is the same as in the original graph.

We say that a directed game graph has a *normalized weight function* if it assigns a positive weight to every out-edge of maximizer vertex, and a negative weight for every out-edge of minimizer vertex. The next lemma shows that we may assume w.l.o.g that a directed game graph has a normalized weight function.

Lemma 2. *Solving mean-payoff games on (directed) bipartite graphs is polynomial time inter-reducible to solving mean-payoff games on (directed) bipartite graphs with normalized weights.*

Proof. Let G be a non-normalized graph and let us denote by W the heaviest weight (in absolute value) that is assigned by its weight function w. We construct a normalized graph G' from G by defining a weight function w' as:

$$w'(u,v) = \begin{cases} w(u,v) + (W+1) & \text{if } u \text{ is owned by maximizer} \\ w(u,v) - (W+1) & \text{otherwise (if } u \text{ is owned by the minimizer)} \end{cases}$$

Clearly, G' is a normalized graph, and since G' and G are bipartite, it is straight forward to observe that for any finite path in π we have that $|w(\pi) - w'(\pi)| \leq W+1$, and thus, for every infinite path ρ, we have that that mean-payoff value of ρ according to w is identical to the mean-payoff of ρ according to w'. Therefore, a maximizer winning (resp. optimal) strategy in graph G is a winning (optimal) strategy also in G' and vice versa, and the proof of the lemma follows. □

In the next lemma we show that mean-payoff games on direct normalized bipartite complete graphs are as hard as mean-payoff games on undirected normalized bipartite complete graphs.

Lemma 3. *The problem of determining whether maximizer has a winning strategy for a threshold* 0 *for a mean-payoff games on a directed normalized bipartite complete graph is as hard as the corresponding problem for mean-payoff games on an undirected normalized bipartite complete graph*

To conclude, by Lemma 1 we get that a polynomial algorithm for alternating move two-player zero-sum games exists if and only if there exists a polynomial algorithm for solving mean-payoff games on a complete bipartite undirected graph, and by Lemmas 2, 1 and 3 and by Theorem 3 we get that the latter exists if and only if there exists a polynomial algorithm for solving mean-payoff games on arbitrary (directed) graphs. Hence, the main result of this section follows.

Theorem 4. *There exists a polynomial time algorithm for computing exact equilibrium for two-player zero-sum (alternating move) games if and only if there exists a polynomial time algorithm for solving mean-payoff games on graphs.*

4 Complexity of Computing Optimal Equilibrium

In this section, we investigate the complexity of computing an optimal equilibrium. Our main results are summarized in the next theorem:

Theorem 5. *1. Optimal equilibrium can be obtained by a profile of pure strategies.*
2. If mean-payoff games are in P, then there is a polynomial algorithm for computing an exact optimal equilibrium.

3. *If mean-payoff games are not in P, then there is no FPTAS that approximate the social welfare of the optimal equilibrium.*
4. *There is an FPTAS to compute an ϵ-equilibrium that is δ-optimal. (Note that it does not necessarily approximate the value of an exact optimal equilibrium.)*

We will prove Theorem 5 in the next four subsections: In Subsection 4.1 we show the naive algorithm for computing an equilibrium that is based on Folk Theorem, and we prove basic properties of equilibria in alternating move games. In Subsection 4.2 we prove Theorem 5(1). In Subsection 4.3 we investigate the complexity of computing the social welfare of the optimal equilibrium, and prove Theorem 5(2) and Theorem 5(3). Finally, in Subsection 4.4 we prove Theorem 5(4) which is the main result of this section.

In this section, we will model n-action k-player alternating move games by a multi-weighted graph, according to the following conventions: The vertices of the graph are the vertices in the set $V = (A_1 \times A_2 \times \cdots \times A_k) \times \{1, \ldots, k\}$, and we say that player i owns the vertex set $V_i = (A_1 \times A_2 \times \cdots \times A_k) \times \{i\}$. Intuitively, a vertex is characterized by an action vector and by a player that owns it. The pair (u, v) is in the edge relation if u is owned by player i, v is owned by player $i + 1$ (where player $k + 1$ is player 1), and there is at most one difference in the action vector of u and v and it is in position i. The weight of every edge is a vector of size k that corresponds to the utility vector of the actions. Formally, if $u = (\overline{a_1}, i)$ and $v = (\overline{a_2}, i + 1)$ then $w(u, v) = (u_1(\overline{a_2}), u_2(\overline{a_2}), \ldots, u_k(\overline{a_2}))$.

For an infinite path in the multi-weighted graph we define the dimension i of *mean-payoff vector* of the path to be the mean-payoff value of the path according to dimension i. It is an easy observation that every infinite path in the graph corresponds to a play and its mean-payoff vector corresponds to the utility vector of the play. We note that the size of the graph is $k^2 \cdot n^k$ which is polynomial in the size of the encoding of the utility functions (which is $k \cdot n^k$), hence this graph can be constructed in polynomial time.

4.1 Basic Properties of Equilibria

The Folk Theorem gives a conceptually simple (but inefficient) technique to construct an equilibrium. Intuitively, an equilibrium is obtained when each of the players play as if the goal of all the other players is to minimize its utility, and if one of the players deviates from this strategy, then all the other players switch to playing according to a strategy that will minimize the utility of the rebellious player. Formally, let G be the corresponding k-player game graph that models the alternating move game. For every player i, we consider a zero-sum two-player mean-payoff game graph G^i in which the maximizer owns player i vertices and the minimizer owns the other vertices. Let σ_i be an arbitrary optimal strategy for the maximizer in graph G^i, let $\overline{\sigma_i}$ be an arbitrary optimal strategy for the minimizer in G^i, and let ν_i be the value that is obtained by the strategy profile $(\sigma_i, \overline{\sigma_i})$. Then if every player i plays according to the strategy:

If player $j \neq i$ deviated from σ_j, then play according to $\overline{\sigma_j}$ forever, and otherwise play according to σ_i

an equilibrium is formed (since by definition, playing according to σ_i assures utility at least ν_i, and deviating from σ_i assures utility at most ν_i).

In the next lemma, we extend the basic principle of Folk Theorem and get a characterization of all the equilibria that are obtained by a profile of pure strategies.

Lemma 4. *Let $(t_1, t_2, \ldots, t_k)$ be a utility vector such that $t_i \geq \nu_i$ (for every player i), then there exists a pure equilibrium with utility exactly t_i for every player i if and only if there exists an infinite path π in the graph G with mean-payoff vector $(t_1, t_2, \ldots, t_k)$.*

4.2 Optimal Equilibrium Can Be Obtained by Pure Strategies

In this subsection, we extend Lemma 4 also for the case of mixed strategies, and as a consequence we get that optimal equilibrium can be obtained by pure strategies. Intuitively, we wish to show that if a profile of (mixed) strategies yields a utility vector $(t_1, \ldots, t_k)$, then there exists an infinite path in the graph with mean-payoff vector that is greater or equal (in every dimension) to $(t_1, \ldots, t_k)$. Then we get that if a utility vector is obtained by a profile of mixed strategies, and then by Lemma 4 it is also obtained by a profile of pure strategies.

We formally prove the above by the next two lemmas.

Lemma 5. *Let G be a multi-weighted graph that is strongly connected, and let $(t_1, \ldots, t_k)$ be a vector. Then if for every $\alpha > 0$ there exists a (finite) cyclic path with average weight at least $t_i - \alpha$ in every dimension, then there exists an infinite path with mean-payoff vector at least $(t_1, \ldots, t_k)$.*

Lemma 6. *Let σ be a profile of (mixed) strategies with utility vector $(t_1, \ldots, t_k)$. Then for every $\alpha > 0$ there exists a cyclic path in the game graph with average weight at least $t_i - \alpha$ in every dimension.*

We are now ready to prove that the utility vector of a mixed equilibrium can be obtained by a pure equilibrium.

Proposition 1. *Let σ be a profile of mixed strategies that induces a utility vector $(t_1, \ldots, t_k)$. Then there exists a profile σ' of pure strategies that induces exactly the same utility vector. Moreover, if σ is an equilibrium, then so is σ'.*

Proof. By Lemma 6 we get that for every $\alpha > 0$ there is a cyclic path with average weight at least $t_i - \alpha$ in every dimension. Therefore, by Lemma 5 we get that there is an infinite path in G with mean-payoff vector at least $(t_1, \ldots, t_k)$, and by Lemma 4 we get that there is a profile of pure strategies that has utility at least $(t_1, \ldots, t_k)$. If σ is an equilibrium we get that $t_i \geq \nu_i$ (since otherwise, player i would deviate to strategy σ_i), and thus, by Lemma 4, we get that there is a pure equilibrium that gives the same utility vector. □

The next corollary immediately follows from Proposition 1.

Corollary 1 (Theorem 5(1)). *An optimal equilibrium can be obtained by a profile of pure strategies.*

4.3 The Complexity of Computing the Social Welfare of the Optimal (Exact) Equilibrium

In this section we show that if there is a polynomial algorithm for mean-payoff games, then there is a polynomial algorithm to compute an optimal equilibrium in a k-player alternating move games. We also prove the converse direction, that is, we show that if there is a polynomial algorithm that computes the social welfare of the optimal equilibrium, then there is a polynomial algorithm that solves mean-payoff games. We prove these two assertions in the next two lemmas.

Lemma 7. *Suppose that mean-payoff games are in P, then there is a polynomial algorithm that computes the social welfare of an optimal equilibrium.*

Proof. Due to Corollary 1, it is enough to consider only pure strategies, and due to Lemma 4 a vector of utilities $(t_1, \ldots, t_k)$ is obtained by a pure equilibrium if and only if $t_i \geq \nu_i$ (for $i = 1, \ldots, k$) and there is an infinite path in the game graph with mean-payoff $(t_1, \ldots, t_k)$. Since we assume that there is a polynomial algorithm for computing ν_i, our problem boils down to

Find the maximal value of $\sum_{i=1}^{k} t_i$ subject to
- $t_i \geq \nu_i$; and
- there exists an infinite path with mean-payoff vector at least $(t_1, \ldots, t_k)$

It was shown in [11] (in the proof of Theorem 18) that the problem of deciding whether there exists an infinite path with mean-payoff vector at least $(t_1, \ldots, t_k)$ can be reduced (in polynomial time) to a set of linear constraints. Moreover, the generated set of constraints remain linear even when t_i is a variable. Hence, we can find a feasible threshold vector $(t_1, \ldots, t_k)$ (that is, a vector that is realizable by an infinite path in the graph) that maximizes $\sum_{i=1}^{k} t_i$ by linear programming. Therefore, if we have a polynomial algorithm that computes ν_i, then we can find the social welfare of the optimal equilibrium in polynomial time. □

Lemma 7 proves Theorem 5(2) and gives an upper bound to the complexity of computing optimal equilibrium. In the next lemma we show that this bound is tight, and that the social welfare of the optimal equilibrium cannot be approximated, unless mean-payoff games are in P.

Lemma 8 (Theorem 5(3)). *There is no FPTAS that approximates the social welfare of an optimal equilibrium, unless mean-payoff games are in P.*

4.4 An FPTAS to Compute an ϵ-Equilibrium That Is δ-Optimal

In this subsection, we assume that the utilities of the players are scaled to rationals in $[-1, 1]$, and we will describe an algorithm that computes an ϵ-equilibrium

that is δ-optimal (with respect to all ϵ-equilibria) and runs in time complexity that is polynomial in the input size and in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Subsection 4.3 suggests that in order to compute a δ-optimal ϵ-equilibrium we should approximate (by some value) the values of $\nu_1, \ldots, \nu_k$ and then compute the optimal infinite path (with respect to the sum of utilities) that has utility for player i that is greater than the approximation of ν_i. However, this approach would not work, since the optimal social welfare is not a continuous function with respect to the values $\nu_1, \ldots, \nu_k$.

We denote by OPT_ϵ the social welfare of the optimal ϵ-equilibrium. We base our solution on the next lemma, which gives two key properties of OPT_ϵ.

Lemma 9. *1. If $\epsilon_1 \geq \epsilon_2$, then $OPT_{\epsilon_1} \geq OPT_{\epsilon_2}$.*
2. For every $\alpha \in [0,1]$ and $\epsilon_1, \epsilon_2 > 0$, let $\epsilon = \alpha\epsilon_1 + (1-\alpha)\epsilon_2$, then there exists an ϵ-equilibrium with social welfare $\alpha OPT_{\epsilon_1} + (1-\alpha)OPT_{\epsilon_2}$.

Proof. The first item of the lemma is a trivial observation. In order to prove the second item, we observe that by Lemma 4 (and since by Proposition 1 it is enough to consider only pure equilibria) it is enough to prove that there is an infinite path π with utility at least $\nu_i - \epsilon$ in every dimension and with social welfare $\alpha OPT_{\epsilon_1} + (1-\alpha)OPT_{\epsilon_2}$. By Proposition 1, it is enough to show that there is a profile σ of mixed strategies (that need not be an equilibrium) that has a utility at least $\nu - \epsilon$ in every dimension and has a social welfare at least $\alpha OPT_{\epsilon_1} + (1-\alpha)OPT_{\epsilon_2}$. The construction of σ is trivial. For $i = 1, 2$, let σ_{ϵ_i} be a profile of strategies that induces an ϵ_i-optimal equilibrium, then we construct σ by playing according to σ_{ϵ_1} with probability α and playing according to σ_{ϵ_2} with probability $1-\alpha$. □

Corollary 2. *For every $\zeta \leq \frac{\epsilon\delta}{4k}$ we have $OPT_{\epsilon+\zeta} - \frac{\delta}{2} \leq OPT_\epsilon \leq OPT_{\epsilon+\zeta}$*

By the above corollary, to approximate OPT_ϵ, it is enough to approximate by $\frac{\delta}{2}$ the value of $OPT_{\epsilon+\zeta}$ for some $\zeta \leq \frac{\epsilon\delta}{4k}$. For this purpose, we extend the notion of ϵ-equilibrium also for k-dimensional vectors, and we say that a profile of strategies is a $\overline{\beta}$-equilibrium if player i cannot improve its utility by at least β_i. Let us denote by $\min \overline{\beta}$ and by $\max \overline{\beta}$ the minimal and maximal element of β (respectively). Then by definition, $OPT_{\min \overline{\beta}} \leq OPT_{\overline{\beta}} \leq OPT_{\max \overline{\beta}}$, and by Corollary 2 we get that $OPT_{\overline{\beta}} \leq OPT_{\max \overline{\beta}} \leq OPT_{\overline{\beta}} + (\max \overline{\beta} - \min \overline{\beta}) \cdot \frac{4k}{\epsilon}$.

We are now ready to present an FPTAS that computes a δ-approximation for OPT_ϵ: (1) Set $\zeta = \frac{\epsilon\delta}{4k}$, and compute a ζ approximation of ν_i for every player i, and denote it by r_i. (2) Compute the optimal path π (with respect to social welfare) that has utility at least $r_i - (\epsilon - \zeta)$ for every player, and return its social welfare.

We note that we can execute the first step of the algorithm in polynomial time due to [10](Observation 3.1), and we can execute the second step in polynomial time by solving the linear programming problem that we described in the proof of Lemma 7. The next lemma proves the correctness of our approximation algorithm.

Lemma 10. *Let $S(\pi)$ be the social welfare of π. Then $S(\pi) - \delta \leq OPT_\epsilon \leq S(\pi)$*

Lemma 10 along with the complexity analysis that we provided, proves that there is an FPTAS to compute an ϵ-equilibrium that is δ-optimal, and Theorem 5(4) follows. We also note that our proof for Theorem 5(4) gives a constructive (and polynomial) algorithm that computes a description of an actual ϵ-equilibrium that is δ-optimal.

Acknowledgement. This work was carried out in partial fulfillment of the requirements for the course Computational Game Theory that was given by Prof. Amos Fiat in Tel Aviv University.

References

1. Andrews, R.L., Manrai, A.K., Pisaruk, N.: Mean cost cyclical game. Math. Oper. Res. 24(4), 817–828 (1999)
2. Björklund, H., Vorobyov, S.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Applied Mathematics 155(2), 210–229 (2007)
3. Brim, L., Chaloupka, J., Doyen, L., Gentilini, R., Raskin, J.-F.: Faster algorithms for mean-payoff games. Formal Methods in System Design 38(2), 97–118 (2011)
4. Chatterjee, K., Henzinger, M., Krinninger, S., Nanongkai, D.: Polynomial-time algorithms for energy games with special weight structures. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 301–312. Springer, Heidelberg (2012)
5. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. Int. Journal of Game Theory 8(2), 109–113 (1979)
6. Gurvich, V.A., Karzanov, A.V., Khachivan, L.G.: Cyclic games and an algorithm to find minimax cycle means in directed graphs. USSR Computational Mathematics and Mathematical Physics 28(5), 85–91 (1988)
7. Lifshits, Y.M., Pavlov, D.S.: Potential theory for mean payoff games. Journal of Mathematical Sciences 145, 4967–4974 (2007)
8. Littman, M.L., Stone, P.: A polynomial-time nash equilibrium algorithm for repeated games. In: Proceedings of the ACM Conference on Electronic Commerce (ACM-EC), pp. 48–54 (2004)
9. Roth, A.: Personal website, `http://www.cs.cmu.edu/~alroth/alternatingmoves.html`
10. Roth, A., Balcan, M.F., Kalai, A., Mansour, Y.: On the equilibria of alternating move games. In: SODA (2010)
11. Ummels, M., Wojtczak, D.: The complexity of nash equilibria in limit-average games. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 482–496. Springer, Heidelberg (2011)
12. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. 158(1&2), 343–359 (1996)

Approximating the Diameter of Planar Graphs in Near Linear Time*

Oren Weimann and Raphael Yuster

University of Haifa, Israel
oren@cs.haifa.ac.il, raphy@math.haifa.ac.il

Abstract. We present a $(1+\varepsilon)$-approximation algorithm running in $O(f(\varepsilon) \cdot n \log^4 n)$ time for finding the diameter of an undirected planar graph with n vertices and with non-negative edge lengths.

1 Introduction

The diameter of a graph is the largest distance between two vertices. Computing it is among the most fundamental algorithmic graph problems.

In general weighted graphs, as well as in planar graphs, the only known way to compute the diameter is to essentially solve the (more general) All-Pairs Shortest Paths (APSP) problem and then take the pair of vertices with the largest distance.

In general weighted graphs with n vertices and m edges, solving APSP (thus diameter) currently requires $\tilde{O}(n^3)$ time. The fastest algorithm to date is $O(n^3(\log\log n)/\log^2 n)$ by Han and Takaoka [10], or for sparse graphs $O(mn + n^2 \log n)$ by Johnson [13], with a small improvement to $O(mn + n^2 \log\log n)$ [18].

In weighted *planar* graphs, solving APSP can be done in $O(n^2)$ time by Frederickson [9]. While this is optimal for APSP, it is not clear that it is optimal for diameter. Currently, only a logarithmic factor improvement by Wulff-Nilsen [20] is known for the diameter, running in $O(n^2(\log\log n)^4/\log n)$ time. A long standing open problem [5] is to find the diameter in truly subquadratic $O(n^{2-\varepsilon})$ time. Eppstein [7] has shown that if the diameter in a planar graph is bounded by a fixed constant, then it can be found in $O(n)$ time. Fast algorithms are also known for some simpler classes of graphs like outer-planar graphs [8], interval graphs [17], and others [4,6].

In lack of truly subcubic-time algorithms for general graphs and truly subquadratic time algorithms for planar graphs it is natural to seek faster algorithms that *approximate* the diameter. It is easy to approximate the diameter within a factor of 2 by simply computing a Single-Source Shortest Path (SSSP) tree from any vertex in the graph and returning twice the depth of the deepest node in the tree. This requires $O(m + n \log n)$ time for general graphs and $O(n)$ time for planar graphs [12]. For general graphs, Aingworth et al. [1] improved the approximation factor from 2 to 3/2 at the cost of $\tilde{O}(m\sqrt{n} + n^2)$ running time,

* A full version of this paper can be found in Arxiv at http://arxiv.org/abs/1112.1116

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 828–839, 2013.

and Boitmanis et al. [3] gave an additive approximation factor of $O(\sqrt{n})$ with $\tilde{O}(m\sqrt{n})$ running time. For planar graphs, the current best approximation is a 3/2-approximation by Berman and Kasiviswanathan running in $O(n^{3/2})$ time [2]. We improve this to a $(1+\varepsilon)$-approximation running in $\tilde{O}(n)$ time for any fixed $0 < \varepsilon < 1$. More precisely, we prove the following theorem:

Theorem 1. *Given an undirected planar graph with n vertices, non-negative edge lengths, and diameter d. For any $\varepsilon > 0$ we can compute an approximate diameter d' (where $d \leq d' \leq (1+\varepsilon) \cdot d$) in time $O(n \log^4 n/\varepsilon^4 + n \cdot 2^{O(1/\varepsilon)})$.*

Summary of the Algorithm. A lemma of Lipton and Tarjan [16] states that, for any SSSP tree T in a planar graph G, there is a non-tree edge e (where e might possibly be a non-edge of the planar graph) such that the strict interior and strict exterior of the unique simple cycle C in $T \cup \{e\}$ each contains at most $2/3 \cdot n$ vertices. The vertices of C therefore form a *separator* consisting of two shortest paths with the same common starting vertex.

Let G_{in} (resp. G_{out}) be the subgraph of G induced by C and all interior (resp. exterior) vertices to C. Let $d(G_{in}, G_{out}, G)$ denote the largest distance in the graph G between a *marked* vertex in $V(G_{in})$ and a *marked* vertex in $V(G_{out})$. In the beginning, all vertices of G are marked and we seek the diameter which is $d(G, G, G)$. We use a divide and conquer algorithm that first approximates $d(G_{in}, G_{out}, G)$, then unmarks all vertices of C, and then recursively approximates $d(G_{in}, G_{in}, G)$ and $d(G_{out}, G_{out}, G)$ and takes the maximum of all three. We outline this algorithm below. Before running it, we compute an SSSP tree from any vertex using the linear-time SSSP algorithm of Henzinger et al. [12]. The depth of the deepest node in this tree already gives a 2-approximation to the diameter $d(G, G, G)$. Let x be the obtained value such that $x \leq d(G, G, G) \leq 2x$.

Reduce $d(G_{in}, G_{out}, G)$ to $d(G_{in}, G_{out}, G_t)$ in a tripartite graph G_t: The separator C is composed of two shortest paths P and Q emanating from the same vertex, but that are otherwise disjoint. We carefully choose a subset of $16/\varepsilon$ vertices from C called *portals*. The first (resp. last) $8/\varepsilon$ portals are all part of the prefix of P (resp. Q) that is of length $8x$. The purpose of the portals is to approximate a shortest u-to-v path for $u \in G_{in}$ and $v \in G_{out}$ by forcing it to go through a portal. Formally, we construct a tripartite graph G_t with vertices $(V(G_{in}), \mathit{portals}, V(G_{out}))$. The length of edge $(u \in V(G_{in}), v \in \mathit{portals})$ or $(u \in \mathit{portals}, v \in V(G_{out}))$ in G_t is the u-to-v distance in G. This distance is computed by running the SSSP algorithm of [12] from each of the $16/\varepsilon$ portals. By the choice of portals, we show that $d(G_{in}, G_{out}, G_t)$ is a $(1+2\varepsilon)$-approximation of $d(G_{in}, G_{out}, G)$.

Approximate $d(G_{in}, G_{out}, G_t)$: If ℓ is the maximum edge-length of G_t, then note that $d(G_{in}, G_{out}, G_t)$ is between ℓ and 2ℓ. This fact makes it possible to round the edge-lengths of G_t to be in $\{1, 2, \ldots, 1/\varepsilon\}$ so that $\varepsilon\ell \cdot d(G_{in}, G_{out}, G_t)$ after rounding is a $(1+2\varepsilon)$-approximation to $d(G_{in}, G_{out}, G_t)$ before rounding. For any fixed ε we can assume without loss of generality that $1/\varepsilon$ is an integer. This means that after rounding $d(G_{in}, G_{out}, G_t)$ is bounded by some fixed integer. We give a linear-time algorithm to compute it exactly, thus approximating

$d(G_{in}, G_{out}, G)$. We then unmark all vertices of C and move on to recursively approximate $d(G_{in}, G_{in}, G)$ (the case of $d(G_{out}, G_{out}, G)$ is symmetric).

Reduce $d(G_{in}, G_{in}, G)$ to $d(G_{in}, G_{in}, G^+_{in})$ in a planar graph G^+_{in} of size at most $2/3 \cdot n$: In order to apply recursion, we construct planar graphs G^+_{in} and G^+_{out} (that is constructed similarly to G^+_{in}). The size of each of these graphs will be at most $2/3 \cdot n$ and their total size $n + o(n)$. We would like G^+_{in} to be such that $d(G_{in}, G_{in}, G^+_{in})$ is a $(1 + \varepsilon/(2 \log n))$-approximation[1] to $d(G_{in}, G_{in}, G)$.

To construct G^+_{in}, we first choose a subset of $256 \log n/\varepsilon$ vertices from C called *dense portals*. We then compute all $O((256 \log n/\varepsilon)^2)$ shortest paths in G_{out} between dense portals. The graph B' obtained by the union of all these paths has at most $O((256 \log n/\varepsilon)^4)$ vertices of degree > 2. We contract vertices of degree $= 2$ so that the number of vertices in B' decreases to $O((256 \log n/\varepsilon)^4)$. Appending this small graph B' (after unmarking all of its vertices) as an exterior to G_{in} results in a graph G^+_{in} that has $|G_{in}| + O((256 \log n/\varepsilon)^4)$ vertices and $d(G_{in}, G_{in}, G^+_{in})$ is a $(1 + \varepsilon/(2 \log n))$-approximation of $d(G_{in}, G_{in}, G)$.

The problem is still that the size of G^+_{in} is not necessarily bounded by $2/3 \cdot n$. This is because C (that is part of G^+_{in}) can be as large as n. We show how to shrink G^+_{in} to size roughly $2/3 \cdot n$ while $d(G_{in}, G_{in}, G^+_{in})$ remains a $(1 + \varepsilon/(2 \log n))$-approximation of $d(G_{in}, G_{in}, G)$. To achieve this, we shrink the C part of G^+_{in} so that it only includes the dense portals without changing $d(G_{in}, G_{in}, G^+_{in})$.

Approximate $d(G_{in}, G_{in}, G^+_{in})$: Finally, once $|G^+_{in}| \leq 2/3 \cdot n$ we apply recursion to $d(G_{in}, G_{in}, G^+_{in})$. In the halting condition, when $|G^+_{in}| \leq (256 \log n/\varepsilon)^4$, we naively compute $d(G_{in}, G_{in}, G^+_{in})$ using APSP.

Related Work. The use of shortest-path separators and portals to approximate distances in planar graphs was first suggested in the context of *approximate distance oracles*. These are data structures that upon query u, v return a $(1 + \varepsilon)$-approximation of the u-to-v distance. Thorup [19] presented an $O(1/\varepsilon \cdot n \log n)$-space oracle answering queries in $O(1/\varepsilon)$ time on directed weighted planar graphs. Independently, Klein [15] achieved these same bounds for undirected planar graphs.

In distance oracles, we need distances between every pair of vertices and each vertex is associated with a possibly different set of portals. In our diameter case however, since we know the diameter is between x and $2x$, it is possible to associate all vertices with the exact same set of portals. This fact is crucial in our algorithm, both for its running time and for its use of rounding. Another important distinction between our algorithm and distance oracles is that distance oracles upon query (u, v) can inspect all recursive subgraphs that include both u and v. We on the other hand must have that, for every (u, v), the shortest u-to-v path exists (approximately) in the unique subgraph where u and v are separated by C. This fact necessitated our construction of G^+_{in} and G^+_{out}.

[1] $\log n = \log_2 n$ throughout the paper.

2 The Algorithm

In this section we give a detailed description of an algorithm that approximates the diameter of an undirected weighted planar graph $\mathcal{G} = (V, E)$ in the bounds of Theorem 1. The algorithm computes a $(1+\varepsilon)$-approximation of the diameter $d = d(G, G, G)$ for $G = \mathcal{G}$. This means it returns a value d' where $d \leq d' \leq (1+\varepsilon) \cdot d$ (recall that, before running the algorithm, we compute a value x such that $x \leq d \leq 2x$ by computing a single-source shortest-path tree from an arbitrary vertex in $\mathcal{G}$). We focus on approximating the value of the diameter. An actual path of length d' can be found in the same time bounds. For simplicity we will assume that shortest paths are unique. This can always be achieved by adding random infinitesimal weights to each edge, and can also be achieved deterministically using lexicographic-shortest paths (see, e.g., [11]). Also, to simplify the presentation, we assume that $\varepsilon \leq 0.1$ and we describe a $(1+7\varepsilon)$-approximation. (then just take $\varepsilon' = \varepsilon/7$).

The algorithm is recursive and actually solves the more general problem of finding the largest distance only between all pairs of *marked vertices*. In the beginning, we mark all $n = |V(G)|$ vertices of $G = \mathcal{G}$ and set out to approximate $d(G, G, G)$ (the largest distance in G between marked vertices in $V(G)$). Each recursive call approximates the largest distance in a specific subset of marked vertices, and then unmarks some vertices before the next recursive call. We make sure that whatever the endpoints of the actual diameter are, their distance is approximated in some recursive call. Finally, throughout the recursive calls, we maintain the invariant that the distance between any two *marked* vertices in the graph G of the recursive call is a $(1+\varepsilon)$-approximation of their distance in the original graph $\mathcal{G}$ (there is no guarantee on the marked-to-unmarked or the unmarked-to-unmarked distances). We denote by $\delta_{\mathcal{G}}(u, v)$ the u-to-v distance in the original graph $\mathcal{G}$.

The recursion is applied according to a variant of the shortest-path separator decomposition for planar graphs by Lipton and Tarjan [16]: We first pick any *marked* vertex v_1 and compute in linear time the SSSP tree from v_1 in G. In this tree, we can find in linear time two shortest paths P and Q (both emanating from v_1) such that removing the vertices of $C = P \cup Q$ from G results in two disjoint planar subgraphs A and B (i.e., there are no edges in $V(A) \times V(B)$). The number of vertices of C can be as large as n but it is guaranteed that $|V(A)| \leq 2/3 \cdot n$ and $|V(B)| \leq 2/3 \cdot n$. Notice that the paths P and Q might share a common prefix. It is common to not include this shared prefix in C. However, in our case, we must have the property that P and Q start at a *marked* vertex. So we include in C the shared prefix as well. See Fig. 1 (left).

Let G_{in} (resp. G_{out}) be the subgraph of G induced by $V(C) \cup V(A)$ (resp. $V(C) \cup V(B)$). In order to approximate $d(G, G, G)$, we first compute a $(1+5\varepsilon)$-approximation d_1 of $d(G_{in}, G_{out}, G)$ (the largest distance in G between the marked vertices of $V(G_{in})$ and the marked vertices of $V(G_{out})$). In particular, d_1 takes into account all $V(C) \times V(G)$ distances. We can therefore unmark all the vertices of C and move on to approximate $d_2 = d(G_{in}, G_{in}, G)$ (approximating $d_3 = d(G_{out}, G_{out}, G)$ is done similarly). We approximate $d(G_{in}, G_{in}, G)$ by applying recursion on $d(G_{in}, G_{in}, G_{in}^+)$ where $|V(G_{in}^+)| \leq 2/3 \cdot n$. The marked

vertices in G^+_{in} and in G_{in} are the same and $d(G_{in}, G_{in}, G^+_{in})$ is a $(1+\varepsilon/(2\log n))$-approximation of $d(G_{in}, G_{in}, G)$. This way, the diameter grows by a multiplicative factor of $(1+\varepsilon/(2\log n))$ in each recursive call. Since the recursive depth is $O(\log n)$ (actually, it is never more than $1.8\log n$) we get a $(1+5\varepsilon)\cdot(1+\varepsilon) \leq (1+7\varepsilon)$-approximation d_2 to $d(G_{in}, G_{in}, G)$. Finally, we return $\max\{d_1, d_2, d_3\}$.

2.1 Reduce $d(G_{in}, G_{out}, G)$ to $d(G_{in}, G_{out}, G_t)$

Our goal is now to approximate $d(G_{in}, G_{out}, G)$. For $u \in G_{in}$ and $v \in G_{out}$, we approximate a shortest u-to-v path in G by forcing it to go through a *portal*. In other words, consider a shortest u-to-v path. It is is obviously composed of a shortest u-to-c path in G concatenated with a shortest c-to-v path in G for some vertex $c \in C$. We approximate the shortest u-to-v path by insisting that c is a portal. The fact that we only need to consider u-to-v paths that are of length between x and $2x$ makes it possible to choose the same portals for all vertices.

We now describe how to choose the portals in linear time. Recall that the separator C is composed of two shortest paths P and Q emanating from the same *marked* vertex v_1. The vertex v_1 is chosen as the first portal. Then, for $i = 2, 3, \ldots$ we start from v_{i-1} and walk on P until we reach the first vertex v whose distance from v_{i-1} via P is greater than εx. We designate v as the portal v_i and continue to $i+1$. We stop the process when we encounter a vertex v whose distance from v_1 is greater than $8x$. This guarantees that at most $8/\varepsilon$ portals are chosen from the shortest path P and they are all in a prefix of P of length at most $8x$. This might seem counterintuitive as we know that any shortest path P in the original graph $\mathcal{G}$ is of length at most $2x$. However, since one endpoint of P is not necessarily marked, it is possible that P is a shortest path in G but not even an approximate shortest path in the original graph $\mathcal{G}$. We do the same for Q, and we get a total of $16/\varepsilon$ portals. See Fig. 1 (right).

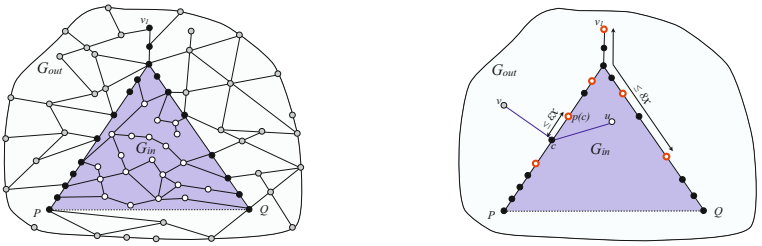


Fig. 1. Two illustrations of a weighted undirected planar graph G. On the left: The black nodes constitute the shortest path separator C composed of two shortest paths P and Q emanating from the same vertex v_1. The subgraph of G induced by the white (resp. gray) nodes is denoted A (resp. B). The graph G_{in} (resp. G_{out}) is the subgraph induced by $A \cup C$ (resp. $B \cup C$). On the right: The six circled vertices are the $16/\varepsilon$ portals in the $8x$ prefixes of P and Q. The shortest path between u and v goes through the separator vertex c and is approximated by the u-to-$\lambda(c)$ and the $\lambda(c)$-to-v shortest paths where $\lambda(c)$ is the closest portal to c. The distance from c to $\lambda(c)$ is at most εx.

Once we have chosen the portals, we move on to construct a tripartite graph G_t whose three vertex sets (or columns) are $(V(G_{in}), portals, V(G_{out}))$. The length of edge $(u \in V(G_{in}), v \in portals)$ or $(u \in portals, v \in V(G_{out}))$ is the u-to-v distance in G. This distance is computed by running the linear-time SSSP algorithm of Henzinger et al. [12] in G from each of the $16/\varepsilon$ portals in total $O(1/\varepsilon \cdot |V(G)|)$ time. The following lemma states that our choice of portals implies that $d(G_{in}, G_{out}, G_t)$ is a good approximation of $d(G_{in}, G_{out}, G)$.

Lemma 1. *If $d(G_{in}, G_{out}, G) \geq x$, then $d(G_{in}, G_{out}, G_t)$ is a $(1 + 2\varepsilon)$-approximation of $d(G_{in}, G_{out}, G)$. Otherwise, $d(G_{in}, G_{out}, G_t) \leq (1 + 2\varepsilon)x$.*

Proof. The first thing to notice is that $d(G_{in}, G_{out}, G_t) \geq d(G_{in}, G_{out}, G)$. This is because every shortest u-to-v path in G_t between a marked vertex $u \in V(G_{in})$ of the first column and a marked vertex $v \in V(G_{out})$ of the third column corresponds to an actual u-to-v path in G.

We now show that $d(G_{in}, G_{out}, G_t) \leq (1+2\varepsilon) \cdot d(G_{in}, G_{out}, G)$. We begin with some notation. Let P_t denote the shortest path in G_t realizing $d(G_{in}, G_{out}, G_t)$. The path P_t is a shortest u-to-v path for some *marked* vertices $u \in G_{in}$ and $v \in G_{out}$. The length of the path P_t is $\delta_{G_t}(u, v)$. Let P_G denote the shortest u-to-v path in G that is of length $\delta_G(u, v)$ and let $P_{\mathcal{G}}$ denote the shortest u-to-v path in the original graph $\mathcal{G}$ that is of length $\delta_{\mathcal{G}}(u, v)$. Recall that we have the invariant that in every recursive level for every pair of marked vertices $\delta_G(u, v) \leq (1 + \varepsilon) \cdot \delta_{\mathcal{G}}(u, v)$. We also have that $\delta_{\mathcal{G}}(u, v) \leq 2x$ and so $\delta_G(u, v) \leq 2x \cdot (1 + \varepsilon)$. For the same reason, since v_1 (the first vertex of both P and Q) is also marked, we know that $\delta_G(v_1, u)$ is of length at most $2x \cdot (1 + \varepsilon)$.

The path P_G must include at least one vertex $c \in C$. Assume without loss of generality that $c \in P$. We claim that c must be a vertex in the prefix of P of length $8x$. Assume the converse, then the v_1-to-c prefix of P is of length at least $8x$. Since P is a shortest path in G, this means that $\delta_G(v_1, c)$ is at least $8x$. However, consider the v_1-to-c path composed of the v_1-to-u shortest path (of length $\delta_G(v_1, u) \leq 2x \cdot (1 + \varepsilon)$) concatenated with the u-to-c shortest path (of length $\delta_G(u, c) \leq \delta_G(u, v) \leq 2x \cdot (1 + \varepsilon)$). Their total length is $4x \cdot (1 + \varepsilon)$ which is less than $8x$ (since $\varepsilon < 1$) thus contradicting our assumption.

After establishing that c is somewhere in the $8x$ prefix of P, we now want to show that $\delta_{G_t}(u, v) \leq (1 + 2\varepsilon) \cdot \delta_G(u, v)$. Let $\lambda(c)$ denote a closest portal to c on the path P. Notice that by our choice of portals and since c is in the $8x$ prefix of P we have that $\delta_G(c, \lambda(c)) \leq \varepsilon x$. By the triangle inequality we know that $\delta_G(u, \lambda(c)) \leq \delta_G(u, c) + \delta_G(c, \lambda(c)) \leq \delta_G(u, c) + \varepsilon x$ and similarly $\delta_G(\lambda(c), v) \leq \delta_G(c, v) + \varepsilon x$. This means that $d(G_{in}, G_{out}, G_t) = \delta_{G_t}(u, v) \leq \delta_G(u, \lambda(c)) + \delta_G(\lambda(c), v) \leq \delta_G(u, c) + \delta_G(c, v) + 2\varepsilon x = \delta_G(u, v) + 2\varepsilon x \leq d(G_{in}, G_{out}, G) + 2\varepsilon x \leq (1 + 2\varepsilon) \cdot d(G_{in}, G_{out}, G)$, where in the last inequality we assumed that $d(G_{in}, G_{out}, G) \geq x$. Note that if $d(G_{in}, G_{out}, G) < x$, then $d(G_{in}, G_{out}, G_t) \leq (1 + 2\varepsilon) \cdot x$. The lemma follows. □

By Lemma 1, approximating $d(G_{in}, G_{out}, G)$ when $d(G_{in}, G_{out}, G) \geq x$ reduces to approximating $d(G_{in}, G_{out}, G_t)$. The case of $d(G_{in}, G_{out}, G) < x$ means that the diameter d of the original graph $\mathcal{G}$ is *not* a $(u \in G_{in})$-to-$(v \in G_{out})$ path. This is

because $d \geq x > d(G_{in}, G_{out}, G) \geq d(G_{in}, G_{out}, \mathcal{G})$. So d will be approximated in a different recursive call (when the separator separates the endpoints of the diameter). In the meanwhile, we will get that $d(G_{in}, G_{out}, G_t)$ is at most $(1+2\varepsilon) \cdot x$ and so it will not compete with the correct recursive call when taking the maximum.

2.2 Approximate $d(G_{in}, G_{out}, G_t)$

In this subsection, we show how to approximate the diameter in the tripartite graph G_t. We give a $(1+2\varepsilon)$-approximation for $d(G_{in}, G_{out}, G_t)$. By the previous subsection, this means we have a $(1+2\varepsilon)(1+2\varepsilon) < (1+5\varepsilon)$-approximation for $d(G_{in}, G_{out}, G)$. From the invariant that distances in G between marked vertices are a $(1+\varepsilon)$-approximation of these distances in the original graph $\mathcal{G}$, we get a $(1+5\varepsilon)(1+\varepsilon) < (1+7\varepsilon)$-approximation for $d(G_{in}, G_{out}, \mathcal{G})$ in $\mathcal{G}$.

We now present our $(1+2\varepsilon)$-approximation for $d(G_{in}, G_{out}, G_t)$ in the tripartite graph G_t. Recall that P_t denotes the shortest path in G_t that realizes $d(G_{in}, G_{out}, G_t)$. By the definition of G_t, we know that the path P_t is composed of only two edges: (1) edge (u, p) between a marked vertex u of the first column (i.e., $u \in V(G_{in})$) and a vertex p of the second column (i.e., p corresponds to some portal in G). (2) edge (p, v) between p and a marked vertex v of the third column (i.e., $v \in V(G_{out})$).

Let X (resp. Y) denote the set of all edges in G_t adjacent to marked vertices of the first (resp. third) column. Let ℓ denote the maximum edge-length over all edges in $X \cup Y$. Notice that $\ell \leq d(G_{in}, G_{out}, G_t) \leq 2\ell$. We round up the lengths of all edges in $X \cup Y$ to the closest multiple of $\varepsilon\ell$. The rounded edge-lengths are thus all in $\{\varepsilon\ell, 2\varepsilon\ell, 3\varepsilon\ell, \ldots, \ell\}$. We denote G_t after rounding as G'_t. Notice that $d(G_{in}, G_{out}, G'_t)$ is a $(1+2\varepsilon)$-approximation of $d(G_{in}, G_{out}, G_t)$. This is because the path P_t is of length at least ℓ and is composed of two edges, each one of them has increased its length by at most $\varepsilon\ell$.

We now show how to compute $d(G_{in}, G_{out}, G'_t)$ exactly in linear time. We first divide all the edge-lengths of G'_t by $\varepsilon\ell$ and get that G'_t has edge-lengths in $\{1, 2, 3, \ldots, 1/\varepsilon\}$. After finding $d(G_{in}, G_{out}, G'_t)$ (which is now a constant) we simply multiply the result by $\varepsilon\ell$. The following lemma states that when the diameter is constant it is possible to compute it exactly in linear time. Note that we can't just use Eppstein's [7] linear-time algorithm because it works only on planar graphs and in our case we get a *non-planar* tripartite graph G'_t.

Lemma 2. *$d(G_{in}, G_{out}, G'_t)$ can be computed in time $O(|V(G)|/\varepsilon + 2^{O(1/\varepsilon)})$.*

Proof. Recall that in G'_t we denote the set of all edges adjacent to marked vertices of the first and third column as X and Y. The length of each edge in $X \cup Y$ is in $\{1, 2, \ldots, k\}$ where $k = 1/\varepsilon$. The number of edges in X (and similarly in Y) is at most $16k \cdot |V(G)|$. This is because the first column contains $|G_{in}| \leq |V(G)|$ vertices and the second column contains $j \leq 16k$ vertices $v_1, v_2, \ldots, v_j$ (the portals). For every marked vertex v in the first (resp. third) column, we store a j-tuple v_X (resp. v_Y) containing the edge lengths from v to all vertices of the second column. In other words, the j-tuple $v_X = \langle \delta(v, v_1), \delta(v, v_2), \ldots, \delta(v, v_j) \rangle$ where every $\delta(v, v_i) \in \{1, 2, \ldots, k\}$ is the length of the edge (v, v_i). The total

number of tuples is $O(k \cdot |V(G)|)$ but the total number of *different* tuples is only $t = k^{O(k)}$ since each tuple has $O(k)$ entries and each entry is in $\{1, 2, \ldots, k\}$.

We create two binary vectors V_X and V_Y each of length t. The i'th bit of V_X (resp. V_Y) is 1 iff the i'th possible tuple exists as some v_X (reps. v_Y). Creating these vectors takes $O(k \cdot |V(G)|) = O(|V(G)|/\varepsilon)$ time. Then, for every 1 bit in V_X (corresponding to a tuple of vertex u in the first column) and every 1 bit in V_Y (corresponding to a tuple of vertex v in the third column) we compute the u-to-v distance in G'_t using the two tuples in time $O(16k)$. We then return the maximum of all such (u, v) pairs. Notice that a 1 bit can correspond to several vertices that have the exact same tuple. We arbitrarily choose any one of these. There are t entries in V_X and t entries in V_Y so there are $O(t^2)$ pairs of 1 bits. Each pair is examined in $O(16k)$ time for a total of $O(kt^2) = k^{O(k)}$ time.

To complete the proof we now show that this last term $O(kt^2)$ is not only $k^{O(k)}$ but actually $2^{O(k)}$. For that we claim that the total number of different tuples is $t = 2^{O(k)}$. We assume for simplicity (and w.l.o.g.) that all portals $v_1, \ldots, v_j$ are on the separator P. We encode a j-tuple $v_X = \langle \delta(v, v_1), \ldots, \delta(v, v_j) \rangle$ by a $(2j-1)$-tuple v'_X: The first entry of v'_X is $\delta(v, v_1)$. The next $j-1$ entries are $|\delta(v, v_{i+1}) - \delta(v, v_i)|$ for $i = 1, \ldots, j-1$. Finally, the last $j-1$ entries are single bits where the i'th bit is 1 if $\delta(v, v_{i+1}) - \delta(v, v_i) \geq 0$ and 0 if $\delta(v, v_{i+1}) - \delta(v, v_i) < 0$.

We will show that the number of different $(2j-1)$-tuples v'_X is $2^{O(k)}$. There are k options for the first entry of v'_X and two options (0 or 1) for each of the last $j-1$ entries. We therefore only need to show that there are at most $2^{O(k)}$ possible $(j-1)$-tuples $\langle a_1, a_2, \ldots, a_{j-1} \rangle$ where $a_i = |\delta(v, v_{i+1}) - \delta(v, v_i)|$. First notice that since $\delta(v, v_{i+1})$ and $\delta(v, v_i)$ correspond to distances, by the triangle inequality we have $a_i = |\delta(v, v_{i+1}) - \delta(v, v_i)| \leq \delta(v_i, v_{i+1})$. We also know that $\delta(v_1, v_j) \leq 8x/\varepsilon\ell$ since all portals lie on a prefix of P of length at most $8x$ and we scaled the lengths by dividing by $\varepsilon\ell$. We get that $\sum_{i=1}^{j-1} a_i \leq 8x/\varepsilon\ell \leq 16k$. In the last inequality we used the fact that $x \leq 2\ell$, if $x > 2\ell$, then we ignore this recursive call altogether (the diameter will be found in another recursive call). To conclude, observe that the number of possible vectors $\langle a_1, a_2, \ldots, a_{j-1} \rangle$ where every a_i is non-negative and $\sum a_i \leq 16k$ is at most $2^{O(k)}$. □

To conclude, we have so far seen how to obtain a $(1 + 5\varepsilon)$-approximation for $d(G_{in}, G_{out}, G)$ implying a $(1 + 7\varepsilon)$-approximation for $d(G_{in}, G_{out}, \mathcal{G})$ in the original graph $\mathcal{G}$. The next step is to unmark all vertices of C and move on to recursively approximate $d(G_{in}, G_{in}, G)$ (and similarly $d(G_{out}, G_{out}, G)$).

2.3 Reduce $d(G_{in}, G_{in}, G)$ to $d(G_{in}, G_{in}, G^+_{in})$

In this subsection we show how to recursively obtain a $(1 + 5\varepsilon)$-approximation of $d(G_{in}, G_{in}, G)$ and recall that this implies a $(1 + 7\varepsilon)$-approximation of $d(G_{in}, G_{in}, \mathcal{G})$ in the original graph $\mathcal{G}$ since we will make sure to maintain our invariant that, at any point of the recursion, distances between marked vertices are a $(1 + \varepsilon)$-approximation of these distances in the original graph $\mathcal{G}$.

It is important to note that our desired construction can be obtained with similar guarantees using the construction of Thorup [19] for distance oracles.

However, we present here a simpler construction than [19] since, as apposed to distance oracles that require *all-pairs* distances, we can afford to only consider distances that are between x and $2x$.

There are two problems with applying recursion to solve $d(G_{in}, G_{in}, G)$. The first is that $|V(G_{in})|$ can be as large as $|V(G)|$ and we need it to be at most $2/3 \cdot |V(G)|$. We do know however that the number of *marked* vertices in $V(G_{in})$ is at most $2/3 \cdot |V(G)|$. The second problem is that it is possible that the u-to-v shortest path in G for $u, v \in G_{in}$ includes vertices of G_{out}. This only happens if the u-to-v shortest path in G is composed of a shortest u-to-p path ($p \in P$) in G_{in}, a shortest p-to-q path ($q \in Q$) in G_{out}, and a shortest q-to-v path in G_{in}. To overcome these two problems, we construct a planar graph G^+_{in} that has at most $2/3 \cdot |V(G)|$ vertices and $d(G_{in}, G_{in}, G^+_{in})$ is a $(1+\varepsilon/(2\log n))$-approximation to $d(G_{in}, G_{in}, G)$.

Recall that the subgraph B of G induced by all vertices in the strict exterior of the separator C is such that $|B| \leq 2/3 \cdot |V(G)|$ and $G_{out} = B \cup C$. The construction of G^+_{in} is done in two phases. In the first phase, we replace the B part of G with a graph B' of polylogarithmic size. In the second phase, we contract the C part of G to polylogarithmic size.

Phase I: Replacing B with B'. To construct G^+_{in}, we first choose a subset of $256 \log n/\varepsilon$ vertices from C called *dense portals*. The dense portals are chosen similarly to the regular portals but there are more of them. The marked vertex v_1 (the first vertex of both P and Q) is chosen as the first dense portal. Then, for $i = 2, \ldots, 128 \log n/\varepsilon$ we start from v_{i-1} and walk on P until we reach the first vertex whose distance from v_{i-1} via P is greater than $\varepsilon x/(16 \log n)$. We set this vertex as the dense portal v_i and continue to $i+1$. We do the same for Q, for a total of $256 \log n/\varepsilon$ dense portals.

After choosing the dense portals, we compute all $O((256 \log n/\varepsilon)^2)$ shortest paths in G_{out} between dense portals. This can be done using SSSP from each portal in total $O(|V(G_{out})| \cdot \log n/\varepsilon)$ time. It can also be done using the Multiple Source Shortest Paths (MSSP) algorithm of Klein [14] in total $O(|V(G_{out})| \cdot \log n + \log^2 n/\varepsilon^2)$ time.

Let B' denote the graph obtained by the union of all these dense portal to dense portal paths in G_{out}. Notice that since these are shortest paths, and since we assumed shortest paths are unique, then every two paths can share at most one consecutive subpath. The endpoints of this subpath are of degree > 2. There are only $O((\log n/\varepsilon)^2)$ paths so this implies that the graph B' has at most $O((\log n/\varepsilon)^4)$ vertices of degree > 2. We can therefore contract vertices of degree $= 2$. The number of vertices of B' then decreases to $O((\log n/\varepsilon)^4)$, it remains a planar graph, and its edge lengths correspond to subpath lengths.

We then unmark all vertices of B' and append B' to the infinite face of G_{in}. In other words, we take the disjoint union of G_{in} and B' and identify the dense portals of G_{in} with the dense portals of B'. This results in a graph G^+_{in} that has $|V(G_{in})| + O((\log n/\varepsilon)^4)$ vertices. In Lemma 3 we will show that $d(G_{in}, G_{in}, G^+_{in})$ can serve as a $(1+\varepsilon/(2\log n))$-approximation to $d(G_{in}, G_{in}, G)$. But first we will shrink G^+_{in} so that the number of its vertices is bounded by $2/3 \cdot |V(G)|$.

Fig. 2. On the left: The graph G^+_{in} before shrinking. The white vertices are the vertices of A, the black vertices are the vertices of C that are not dense portals, the six red circled vertices are the dense portals, and the gray vertices are the vertices of $B' \setminus C$ with degree > 2. On the right: The graph G^+_{in} after shrinking. The edges adjacent to vertices of C that are not dense portals are replaced with edges to dense portals.

Phase II: Shrinking G^+_{in}. The problem with the current G^+_{in} is still that the size of $V(G^+_{in})$ is not necessarily bounded by $2/3 \cdot |V(G)|$. This is because C (that is part of $V(G^+_{in})$) can be as large as n. We now show how to shrink $V(G^+_{in})$ to size $2/3 \cdot |V(G)|$ while $d(G_{in}, G_{in}, G^+_{in})$ remains a $(1 + \varepsilon/(2 \log n))$-approximation of $d(G_{in}, G_{in}, G)$. To achieve this, we shrink the C part of $V(G^+_{in})$ so that it only includes the dense portals. We show how to shrink P, shrinking Q is done similarly.

Consider two dense portals v_i and v_{i+1} on P (i.e., v_i is the closest portal to v_{i+1} on the path P towards v_1). We want to eliminate all vertices of P between v_i and v_{i+1}. Denote these vertices by $p_1, \ldots, p_k$. If v_i is the last portal of P (i.e., $i = 128 \log n$), then $p_1, \ldots, p_k$ are all the vertices between v_i and the end of P. Recall that A is the subgraph of G induced by all vertices in the strict interior of the separator C. Fix a planar embedding of G^+_{in}. We perform the following process as long as there is some vertex u in $Q \cup A$ which is a neighbor of some p_j, and which is on some face of the embedding that also contains v_i. We want to "force" any shortest path that goes through an edge (u, p_j) to also go through the dense portal v_i. To this end, we delete all such edges (u, p_j), and instead insert a single edge (u, v_i) of length $\min_j\{\ell(u, p_j) + \delta_G(p_j, v_i)\}$. Here, $\ell(u, p_j)$ denotes the length of the edge (u, p_j) (it may be that $\ell(u, p_j) = \infty$ if (u, p_j) is not an edge) and $\delta_G(p_j, v_i)$ denotes the length of the p_j-to-v_i subpath of P. It is important to observe that the new edge (u, v_i) can be embedded while maintaining the planarity since we have chosen u to be on the same face as v_i. Observe that once the process ends, the vertices p_j have no neighbors in $Q \cup A$.

Finally, we replace the entire v_{i+1}-to-v_i subpath of P with a single edge (v_{i+1}, v_i) whose length is equal to the entire subpath length. If v_i is the last dense portal in P, then we simply delete the entire subpath between v_i and the end of P. The entire shrinking process takes only linear time in the size of $|V(G)|$ since it is linear in the number of edges of G^+_{in} (which is a planar graph).

The following Lemma asserts that after the shrinking phase $d(G_{in}, G_{in}, G^+_{in})$ can serve as a $(1 + \varepsilon/(2 \log n))$-approximation to $d(G_{in}, G_{in}, G)$. The proof is given in the full version of this paper.

Lemma 3. $d(G_{in}, G_{in}, G) \leq d(G_{in}, G_{in}, G^+_{in}) \leq d(G_{in}, G_{in}, G) + \varepsilon x/(2\log n)$

Corollary 1. *If* $d(G_{in}, G_{in}, G) \geq x$*, then* $d(G_{in}, G_{in}, G^+_{in})$ *is a* $(1+\varepsilon/(2\log n))$*-approximation of* $d(G_{in}, G_{in}, G)$*. If* $d(G_{in}, G_{in}, G) < x$*, then* $d(G_{in}, G_{in}, G^+_{in}) \leq (1+\varepsilon/(2\log n)) \cdot x$*.*

By the above corollary, approximating $d(G_{in}, G_{in}, G)$ when $d(G_{in}, G_{in}, G) \geq x$ reduces to approximating $d(G_{in}, G_{in}, G^+_{in})$. When $d(G_{in}, G_{in}, G) < x$ it means that the diameter of the original graph $\mathcal{G}$ is *not* a $(u \in G_{in})$-to-$(v \in G_{in})$ path and will thus be approximated in a different recursive call.

Finally, notice that indeed we maintain the invariant that the distance between any two *marked* vertices in the recursive call to G^+_{in} is a $(1+\varepsilon)$-approximation of the distance in the original graph $\mathcal{G}$. This is because, by the above corollary, every recursive call adds a $1+\varepsilon/(2\log n)$ factor to the approximation. Each recursive call decreases the input size by a factor of $(2/3+o(1))^{-1}$. Hence, the overall depth of the recursion is at most $\log_{1.5-o(1)} n < 1.8\log n$. Since $(1+\varepsilon/(2\log n))^{1.8\log n} < e^{0.9\varepsilon} < 1+\varepsilon$, the invariant follows (we assume in the last inequality that $\varepsilon \leq 0.1$). Together with the $(1+5\varepsilon)$-approximation for $d(G_{in}, G_{out}, \mathcal{G})$ in the original graph $\mathcal{G}$, we get a $(1+5\varepsilon)\cdot(1+\varepsilon) \leq (1+7\varepsilon)$-approximation of $d(G_{in}, G_{in}, \mathcal{G})$ in $\mathcal{G}$, once we apply recursion to $d(G_{in}, G_{in}, G^+_{in})$.

We note that our recursion halts once $|G^+_{in}| \leq (256\log n/\varepsilon)^4$ in which case we naively compute $d(G_{in}, G_{in}, G^+_{in})$ using APSP in time $O(|G^+_{in}|^2)$. Even at this final point, the distances between marked vertices still obey the invariant.

2.4 Running Time

We now examine the total running time of our algorithm. Let n denote the number of vertices in our original graph $\mathcal{G}$ and let $V(G)$ denote the vertex set of the graph G in the current invocation of the recursive algorithm. The current invocation approximates $d(G_{in}, G_{out}, G_t)$ as shown in subsection 2.2 in time $O(|V(G)|/\varepsilon + 2^{O(1/\varepsilon)})$. It then constructs the subgraphs G^+_{in} and G^+_{out} as shown in subsection 2.3, where we have that after shrinking, $|V(G^+_{in})| = \alpha|V(G)| + O(\log^4 n/\varepsilon^4)$ and $|V(G^+_{out})| = \beta|V(G)| + O(\log^4 n/\varepsilon^4)$, where $\alpha, \beta \leq 2/3$ and $\alpha+\beta \leq 1$. The time to construct $|V(G^+_{in})|$ and $|V(G^+_{out})|$ is dominated by the time required to compute SSSP for each dense portal, which requires $O(|V(G)|\cdot \log n/\varepsilon)$. We then continue recursively to G^+_{in} and to G^+_{out}. Hence, if $T(|V(G)|)$ denotes the running time for G, then we get that $T(|V(G)|) = O\big(|V(G))|\cdot \log n/\varepsilon + 2^{O(1/\varepsilon)}\big) + T\big(\alpha|V(G)| + O(\log^4 n/\varepsilon^4)\big) + T\big(\beta|V(G)| + O(\log^4 n/\varepsilon^4)\big)$. In the recursion's halting condition, once we get to components of size $|V(G)| = (256\log n/\varepsilon)^4$, we naively run APSP. This takes $O(|V(G)|^2)$ time for each such component, and there are $O(n/|V(G)|)$ such components, so the total time is $O(n\cdot|V(G)|) = O(n\log^4 n/\varepsilon^4)$. It follows that $T(n) = O(n\log^4 n/\varepsilon^4 + n\cdot 2^{O(1/\varepsilon)})$.

References

1. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM Journal on Computing 28(4), 1167–1181 (1999)

2. Berman, P., Kasiviswanathan, S.P.: Faster approximation of distances in graphs. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 541–552. Springer, Heidelberg (2007)
3. Boitmanis, K., Freivalds, K., Lediņš, P., Opmanis, R.: Fast and simple approximation of the diameter and radius of a graph. In: Àlvarez, C., Serna, M. (eds.) WEA 2006. LNCS, vol. 4007, pp. 98–108. Springer, Heidelberg (2006)
4. Chepoi, V., Dragan, F.F.: A linear-time algorithm for finding a central vertex of a chordal graph. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 159–170. Springer, Heidelberg (1994)
5. Chung, F.R.K.: Diameters of graphs: Old problems and new results. Congressus Numerantium 60, 295–317 (1987)
6. Dragan, F.F., Nicolai, F., Brandstadt, A.: LexBFS-orderings and powers of graphs. In: D'Amore, F., Marchetti-Spaccamela, A., Franciosa, P.G. (eds.) WG 1996. LNCS, vol. 1197, pp. 166–180. Springer, Heidelberg (1997)
7. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. In: Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 632–640 (1995)
8. Farley, A.M., Proskurowski, A.: Computation of the center and diameter of outerplanar graphs. Discrete Applied Mathematics 2, 185–191 (1980)
9. Frederickson, G.N.: Fast algorithms for shortest paths in planar graphs. SIAM Journal on Computing 16, 1004–1022 (1987)
10. Han, Y., Takaoka, T.: An $O(n^3 \log\log n / \log^2 n)$ time algorithm for all pairs shortest paths. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 131–141. Springer, Heidelberg (2012)
11. Hartvigsen, D., Mardon, R.: The all-pairs min cut problem and the minimum cycle basis problem on planar graphs. Journal of Discrete Mathematics 7(3), 403–418 (1994)
12. Henzinger, M.R., Klein, P.N., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. Journal of Computer and System Sciences 55(1), 3–23 (1997)
13. Johnson, D.B.: Efficient algorithms for shortest paths in sparse graphs. Journal of the ACM 24, 1–13 (1977)
14. Klein, P.N.: Multiple-source shortest paths in planar graphs. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Mathematics (SODA), pp. 146–155 (2005)
15. Klein, P.N.: Preprocessing an undirected planar network to enable fast approximate distance queries. In: Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Mathematics (SODA), pp. 820–827 (2002)
16. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Math. 36, 177–189 (1979)
17. Olariu, S.: A simple linear-time algorithm for computing the center of an interval graph. International Journal of Computer Mathematics 34, 121–128 (1990)
18. Pettie, S.: A faster all-pairs shortest path algorithm for real-weighted sparse graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 85–97. Springer, Heidelberg (2002)
19. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. Journal of the ACM 51(6), 993–1024 (2004); Announced at FOCS 2001
20. Wulff-Nilsen, C.: Wiener index, diameter, and stretch factor of a weighted planar graph in subquadratic time. Technical report, University of Copenhagen (2008)

Testing Linear-Invariant Function Isomorphism

Karl Wimmer[1,⋆] and Yuichi Yoshida[2,⋆⋆]

[1] Duquesne University
[2] National Institute of Informatics and Preferred Infrastructure, Inc.
wimmerk@duq.edu, yyoshida@nii.ac.jp

Abstract. A function $f : \mathbb{F}_2^n \to \{-1, 1\}$ is called linear-isomorphic to g if $f = g \circ A$ for some non-singular matrix A. In the g-isomorphism problem, we want a randomized algorithm that distinguishes whether an input function f is linear-isomorphic to g or far from being so.

We show that the query complexity to test g-isomorphism is essentially determined by the spectral norm of g. That is, if g is close to having spectral norm s, then we can test g-isomorphism with $\mathrm{poly}(s)$ queries, and if g is far from having spectral norm s, then we cannot test g-isomorphism with $o(\log s)$ queries. The upper bound is almost tight since there is indeed a function g close to having spectral norm s whereas testing g-isomorphism requires $\Omega(s)$ queries. As far as we know, our result is the first characterization of this type for functions. Our upper bound is essentially the Kushilevitz-Mansour learning algorithm, modified for use in the implicit setting.

Exploiting our upper bound, we show that any property is testable if it can be well-approximated by functions with small spectral norm. We also extend our algorithm to the setting where A is allowed to be singular.

1 Introduction

In this paper, we are concerned with property testing of Boolean functions. We say that two Boolean functions f and $f' : \mathbb{F}_2^n \to \{-1, 1\}$ are ϵ-*far* if the distance $\mathrm{dist}(f, f') := |\{x \in \mathbb{F}_2^n \mid f(x) \neq f'(x)\}|/2^n$ between f and f' is at least ϵ, and we call them ϵ-*close* otherwise. We say that a function $f : \mathbb{F}_2^n \to \{-1, 1\}$ is ϵ-*far* from a property P if f is ϵ-far from every function f' satisfying P and ϵ-*close* otherwise. A randomized query algorithm A with oracle access to an unknown function $f : \mathbb{F}_2^n \to \{-1, 1\}$ is called an ϵ-*tester* for P if it accepts with probability at least $\frac{2}{3}$ when f satisfies P and rejects with probability at least $\frac{2}{3}$ when f is ϵ-far from P. The efficiency of a tester is measured by its *query complexity*, that is, the number of queries made to the oracle. Here, we are chiefly interested in testers that make a number of queries independent of n (although the number

⋆ Supported by NSF award CCF-1117079.
⋆⋆ Supported by JSPS Grant-in-Aid for Research Activity Start-up (24800082), MEXT Grant-in-Aid for Scientific Research on Innovative Areas (24106001), and JST, ERATO, Kawarabayashi Large Graph Project.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 840–850, 2013.

of queries can depend on some complexity measure); we refer to such testers as making a constant number of queries.

Property testing was first introduced by Rubinfeld and Sudan [1] to test algebraic properties of a function. Goldreich, Goldwasser, and Ron [2] extended the scope of this definition to graphs and other combinatorial objects. Since then, the field of property testing has been very active. For an overview of recent developments, we refer the reader to the surveys [3,4] and the book [5].

A notable achievement in the field of property testing is the complete characterization of graph properties that are testable with a constant number of queries [6]. An ambitious open problem is obtaining a similar characterization for properties of Boolean functions. Recently there has been a lot of progress on the restriction of this question to linear-invariant properties [7,8,9]. A property P of Boolean functions $\mathbb{F}_2^n \to \{-1,1\}$ is called *linear-invariant* if a function f satisfies P, then $f \circ A$ is also in P for any square matrix A, where $(f \circ A)(x) := f(Ax)$ for any $x \in \mathbb{F}_2^n$. For a thorough discussion of linear-invariant properties, we refer the reader to Sudan's survey on the subject [10].

Despite much effort, we are still far from reaching a characterization of constant-query testable linear-invariant properties with two-sided error. In this paper, we consider function isomorphism testing, which in some sense is the simplest possible linear-invariant property. Given a Boolean function g, the *g-isomorphism testing problem* is to determine whether a function f is isomorphic to g, that is, whether $f = g \circ A$ for some non-singular[1] matrix A or far from being so. A natural goal, and the focus of this paper, is to characterize the set of functions for which isomorphism testing can be done with a constant number of queries.

Our first contribution is revealing a parameter of functions that makes isomorphism testing easy. To state our result precisely, let us introduce several definitions about Fourier analysis. Using characteristic functions $\chi_\alpha(x) := (-1)^{\langle \alpha, x \rangle} = (-1)^{\sum_{i=1}^n \alpha_i x_i}$, every Boolean function $f : \mathbb{F}_2^n \to \{-1,1\}$ has a unique representation as $f(x) = \sum_{\alpha \in \mathbb{F}_2^n} \widehat{f}(\alpha)\chi_\alpha(x)$. The coefficients $\widehat{f}(\alpha)$ are called the *Fourier coefficients* of f. Then, the *spectral norm* of f is defined as $\widehat{\|f\|}_1 := \sum_{\alpha \in \mathbb{F}_2^n} |\widehat{f}(\alpha)|$. We show that $\widehat{\|g\|}_1$ essentially determines the hardness of testing g-isomorphism:

Theorem 1 (Main theorem). *Suppose that $g : \mathbb{F}_2^n \to \{-1,1\}$ is $\frac{\epsilon}{3}$-close to having $\widehat{\|g\|}_1 \leq s$. Then, we can ϵ-test g-isomorphism with query complexity* $\mathrm{poly}(s, \frac{1}{\epsilon})$ *queries.*

Indeed, Theorem 1 is proved by giving a *tolerant tester* for the property of having spectral norm at most s. That is, it accepts every function $\frac{\epsilon}{2}$-close to having spectral norm at most s with probability at least $\frac{2}{3}$ whereas it rejects every function ϵ-far from having spectral norm at most s.

[1] The definition of linear invariance allows for using matrices A that could be singular. However, this definition does not induce an equivalence relation on the set of all Boolean functions. Thus, we restrict A to be non-singular to focus on isomorphism. As a corollary of our isomorphism work, we can handle the full linear-invariant case.

In contrast to Theorem 1, if a function g is far from having small spectral norm, then g-isomorphism becomes hard to test:

Theorem 2. *Suppose that $g : \mathbb{F}_2^n \to \{-1, 1\}$ is ϵ-far from having $\widehat{\|g\|}_1 \leq s$. Then, ϵ-testing g-isomorphism requires $\Omega(\log s)$ queries.*

We note that Fischer obtained a similar result for graph isomorphism testing in the dense graph model [11] using complexity of graphs instead of spectral norm. Here, the *complexity* of a graph is the size of a partition needed to express the graph, that is, an edge exists between two parts in the partition if and only if all vertices between the two parts are adjacent. His result has been used as an important tool to obtain the complete characterization of constant-query testable graph properties [6,12]. Thus, we believe that our result will be also useful to obtain a characterization for function properties.

Theorem 1 is close to tight as is shown in the following theorem.

Theorem 3. *For any $s > 0$, there exists a function g with $\widehat{\|g\|}_1 \leq s$ such that testing g-isomorphism requires $\Omega(s)$ queries.*

For a function $f : \mathbb{F}_2^n \to \{-1, 1\}$, $\mathrm{Spec}(f) := \{\alpha \in \mathbb{F}_2^n \mid \widehat{f}(\alpha) \neq 0\}$ denotes the *spectrum* of f. A function f is called *r-sparse* if $|\mathrm{Spec}(f)| \leq r$ and having *Fourier dimension* k if $\mathrm{Spec}(f)$ lies in an k-dimensional subspace of $\mathbb{F}_2^n$. It is not hard to show that an r-sparse function has spectral norm at most $\sqrt{r}$ and an k-dimensional function has spectral norm at most $2^{k/2}$. Thus, as corollaries of Theorem 1, we can test g-isomorphism with a constant number of queries when g is close to being r-sparse for constant r or k-dimensional for constant k. More generally, we can test any property that can be approximated by a set of functions such that each function in the set has small spectral norm.

Theorem 4. *Let P be a linear-invariant property. Suppose that every function $g \in P$ is $\frac{\epsilon}{3}$-close to having $\widehat{\|g\|}_1 \leq s$. Then, we can ϵ-test P with query complexity* $\mathrm{poly}(s, \frac{1}{\epsilon})$.

For example, with this theorem, we can test whether an input function is a small Boolean circuit in which each input is made by the parity of variables. It is shown that r-sparsity and k-dimensionality are constant-query testable in [13]. As a corollary of Theorem 4, these properties are tolerantly constant-query testable.

Further, we can extend Theorem 1 to the case where we alter the definition of linear isomorphism to allow for singular linear transformations.

Proof Sketch of Theorem 1.1. We prove our main theorem in a similar manner to the implicit learning method used in [14] and [13]. Similarly to [13], our algorithm finds all the large Fourier coefficients of the unknown function f using an implicit version of the Goldreich-Levin algorithm [15]; we call our algorithm Implicit Sieve. Given a set of vectors $\mathcal{M}$, it outputs $\{\langle \chi_{\alpha_1}(x), \ldots, \chi_{\alpha_k}(x), f(x)\rangle \mid x \in \mathcal{M}\}$, where α_i's are (implicitly) chosen to cover large Fourier coefficients. The approach in [13] works when f is k-dimensional; our algorithm Implicit Sieve gives correct output with no assumptions about f.

Once Implicit Sieve is run, rather than checking consistency as in [14] and [13], our algorithm produces a sparse polynomial that approximates f. This can be done by estimating $\widehat{f}(\alpha_i)$ from the output of Implicit Sieve. To check isomorphism to a given function g (or even multiple functions), the algorithm tries to "fit" the Fourier mass of f into the Fourier mass of g. We note that checking this "fit" requires no further queries. If the spectral norm of g is small, then this algorithm will succeed, and in fact does so in a tolerant manner. We note the strong connection of learning functions with small spectral norm due to Kushilevitz and Mansour [16].

Finally, we show that tolerant isomorphism testing implies that we can test properties that are well-approximated by functions with small spectral norm.

Related Work. Another natural setting of function isomorphism is that we regard two functions $f, f' : \mathbb{F}_2^n \to \{-1, 1\}$ are isomorphic if they are the same up to relabeling of bits. In this setting, it is known that g-isomorphism is constant-query testable if g is a k-junta for constant k [17] or a symmetric function, and the result was extended to $(n-k)$-*symmetric functions* [18,19]. Here, a function is called a *k-junta* if it only depends on k bits in the input string and called an *$(n-k)$-symmetric function* if there exist some k bits such that it becomes a symmetric function for every fixed setting of the k bits. It is also conjectured that the parameter k above is the parameter that determines the difficulty of g-isomorphism in this setting as the spectral norm is the parameter in the linear isomorphism setting [18,19].

In recent work, Grigorescu et al. [20] consider extremal examples of lower bounds for testing linear isomorphism in the same sense considered here. In that work, the authors show that there is a function that requires $\Omega(n^2)$ queries for testing linear isomorphism. The spectral norm of this function is exponential in n, so this example is quite far away from what we consider here, since we consider functions of "constant" complexity.

Very recently, [21] gave a characterization of affine-invariant properties that are constant-query testable with one-sided error. Their work does not derive our result since having small spectral norm and satisfying the condition they gave are incomparable. Also, we consider two-sided error testability of linear-invariant properties instead of one-sided error testability of affine-invariant properties.

In graph property testing, there are two major models, called the dense graph model and the bounded-degree model. In the *dense graph model*, we have access to the adjacency matrix and a graph G is called *ϵ-far* from a property P if we must add or remove at least ϵn^2 edges to make G satisfy P. As we already mentioned, [11] showed that the difficulty of testing H-isomorphism is determined by the complexity of H. The gap between its upper and lower bounds on query complexity in terms of the graph complexity was improved in [19]. [22] showed that the query complexity to test H-isomorphism for general H is $\widetilde{\Theta}(\sqrt{n})$ with two-sided error and $\widetilde{\Theta}(n)$ with one-sided error. In the *bounded-degree model* with a degree bound d, we only deal with graphs with maximum degree at most d. We have access to the incidence list and a graph G is called *ϵ-far* from a property P if we must add or remove at least ϵdn edges to make G satisfy P. [23]

showed that we can test H-isomorphism in constant time when H is hyperfinite, that is, for any $\epsilon > 0$ we can decompose H into connected components of size $s = s(\epsilon)$ by deleting ϵn edges. A notable example of hyperfinite graphs is planar graphs. Exploiting this result, [23] showed that every hyperfinite property is constant-query testable.

Organization. In Section 2, we introduce definitions and useful facts used throughout this paper. We introduce Implicit Sieve in Section 3 and proceed to prove our main theorem in Section 4. In the full version of the paper, we extend our upper bounds to the linear-invariant case using our linear isomorphism results (Theorem 4), and we prove a lower bound for testing linear isomorphism to functions of small spectral norm.

2 Preliminaries

We use bold symbols to denote random variables. For a collection of vectors $\mathcal{A}$, we write $wt_2(\mathcal{A}) = \sum_{\alpha\in\mathcal{A}} \widehat{f}(\alpha)^2$ and $wt_4(\mathcal{A}) = \sum_{\alpha\in\mathcal{A}} \widehat{f}(\alpha)^4$. This notation suppresses the dependence on f, but it will always be clear from the context. For a subspace $H \leq \mathbb{F}_2^n$ and any vector $r \in \mathbb{F}_2^n$, the *coset* $r + H$ is defined as $\{\alpha + r \mid \alpha \in H\}$. Given a set of vectors $C \subseteq \mathbb{F}_2^n$, let $\mathrm{P}_C(f)$ be the projection function for C; $\mathrm{P}_C(f)(x) = \sum_{\alpha\in C} \widehat{f}(\alpha)\chi_\alpha(x)$.

We show the following two lemmas in the full version of the paper.

Lemma 1. *For a non-singular matrix A, $\widehat{f \circ A}(\alpha) = \widehat{f}((A^{-1})^T\alpha)$. In particular, $\widehat{g}(\alpha) = \widehat{f}((A^{-1})^T\alpha)$ when $f = g \circ A$.*

Lemma 2. *For any subspace H, any vector r, and any integer $k \geq 2$, we can estimate $\mathrm{wt}_2(r + H)$ and $\mathrm{wt}_4(r + H)$ to within $\pm\tau$ with confidence $1 - \delta$ using $O(\frac{\log(1/\delta)}{\tau^2})$ queries.*

3 The Implicit Sieve

As a first step, we give a general algorithm for implicitly accessing the Fourier coefficients of an unknown function f, which we call Implicit Sieve. The guarantee of the algorithm is stated in the following lemma.

Lemma 3. *Given a threshold value $\theta > 0$ and a set $\mathcal{M} \subseteq \mathbb{F}_2^n$ with $|\mathcal{M}| = m$, there is an algorithm that uses $\mathrm{poly}(m, 1/\theta)$ queries and returns, with high probability, for some set $S = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$, a labeled set of m examples of the form $\{\langle\chi_{\alpha_1}(x), \chi_{\alpha_2}(x), \ldots, \chi_{\alpha_k}(x), f(x)\rangle \mid x \in \mathcal{M}\}$, where*

- *Every vector α such that $|\widehat{f}(\alpha)| \geq \theta$ is in S.*
- *Every vector $\alpha \in S$ satisfies $|\widehat{f}(\alpha)| \geq \frac{1}{2}\theta$.*

Due to space limitations, all the proofs are deferred to the full version. Note that the algorithm does *not* return the set S. The two guarantees here are similar to the guarantee given by the Goldreich-Levin Theorem [15] (see also [16]). The query complexity is independent of f.

We first introduce random t-coset structures.

Definition 1 ([13]). *For an integer t, we define a random t-dimensional coset structure $(H, \mathcal{C})$ as follows: We choose vectors $\beta_1, \ldots, \beta_t \in \mathbb{F}_2^n$ independently and uniformly at random and set $H = \mathrm{span}\{\beta_1, \ldots, \beta_t\}^\perp$. For each $b \in \mathbb{F}_2^t$ we define the "bucket" $C(b) = \{\alpha \in \mathbb{F}_2^n \mid \langle \alpha, \beta_i \rangle = b_i \text{ for all } i\}$. We take $\mathcal{C}$ to be the multiset of $C(b)$'s, which has cardinality 2^t. In a random permuted t-dimensional coset structure, we additionally choose a random $z \in \mathbb{F}_2^t$ and rename $C(b)$ by $C(b+z)$.*

In the (unlikely) case that β_i's are linearly dependent, some of $C(b)$'s will be cosets in $\mathbb{F}_2^n/H$ and some of them will be empty. For the empty buckets $C(b)$ we define $P_{C(b)}f$ to be identically 0. Note that the zero vector always gets mapped to the bucket $C(0)$ in a random coset structure. To avoid technical issues caused by this fact, we adopt random permuted coset structures.

Lemma 4. *Let $(H, \mathcal{C})$ be a random permuted t-dimensional coset structure, where $t \geq 2 \log \frac{16}{\theta^4} + \log 100$. Then for any set S of vectors α with $|\widehat{f}(\alpha)| \geq \frac{\theta^2}{4}$, every vector in S gets mapped into a different bucket except with probability at most $\frac{1}{100}$.*

Our algorithm is given in Algorithm 1. A bucket is *heavy* if it contains some $\alpha \in \mathbb{F}_2^n$ with $|\widehat{f}(\alpha)| \geq \frac{\theta^2}{4}$. We now explain how Algorithm 1 works.

We first choose a random permuted coset structure (Step 1). Conditioning on Lemma 4, for all the heavy buckets, there is a unique Fourier coefficient of squared magnitude at least $\frac{\theta^2}{4}$ contained in that bucket. For any heavy bucket C, let $\alpha(C)$ be this unique Fourier coefficient. Then, we can show that the Fourier mass of a bucket C is dominated by $\alpha(C)$:

Lemma 5. *Suppose t is such that $10 \cdot 2^{-t/2} + 2^{-t} \leq (\frac{\theta^3}{3200m})^2$. Assume that the condition of Lemma 4 holds. Then except with probability at most $\frac{2}{100}$, $\mathrm{wt}_2(C) \leq \widehat{f}(\alpha(C))^2 + (\frac{\theta^3}{3200m})^2$ for every heavy bucket C.*

Next, by estimating $\mathrm{wt}_2(C)$ and $\mathrm{wt}_4(C)$, we discard buckets $C \in \mathcal{C}$ that are judged to be non-heavy (Steps 2–7). Let $\mathcal{L}'$ be the set of buckets we did not discard. With high probability, we do not make any mistake and do not miss any large Fourier coefficient:

Lemma 6. *Except with probability at most $\frac{1}{100}$,*

- *Find-Heavy-Buckets discards every bucket C with $|\widehat{f}(\alpha)| < \frac{\theta^2}{4}$ for every $\alpha \in C$.*
- *Find-Heavy-Buckets does not discard any bucket C with $|\widehat{f}(\alpha(C))| \geq \theta$.*
- *$|\widehat{f}(\alpha(C))| \geq \frac{\theta}{2}$ for every $C \in \mathcal{L}'$.*

Algorithm 1. Implicit-Sieve

Parameters: θ, $\mathcal{M}$ with $|\mathcal{M}| = m$, $t = \Omega(\log \frac{m^4}{\theta^{12}})$

Output: A matrix $\mathcal{Q}$ and a vector $\mathcal{F}$ of implicit examples, where each row of $\mathcal{Q}$ and the corresponding entry of $\mathcal{F}$ corresponds to a string in $\mathbb{F}_2^n$, and each column of $\mathcal{Q}$ corresponds to some linear function χ_α evaluated at every string in $\mathcal{M}$.

1: Choose a random permuted t-dimensional coset structure $(H, \mathcal{C})$.
{Find-Heavy-Buckets}
2: **for** each bucket $C \in \mathcal{C}$ **do**
3: Estimate $\mathrm{wt}_2(C)$ to within $\pm\frac{1}{4}\theta^2$ with confidence $1 - \frac{1}{200 \cdot 2^t}$. Call it $\widetilde{\mathrm{wt}}_2(C)$.
4: Discard any bucket where $\widetilde{\mathrm{wt}}_2(C) < \frac{3}{4}\theta^2$.
5: **for** each surviving bucket $C \in \mathcal{C}$ **do**
6: Estimate $\mathrm{wt}_4(C)$ to within $\pm\frac{\theta^4}{4}\widetilde{\mathrm{wt}}_2(C)$ with confidence $1 - \frac{1}{200 \cdot 2^t}$. Call it $\widetilde{\mathrm{wt}}_4(C)$.
7: Discard any bucket where $\widetilde{\mathrm{wt}}_4(C) < \frac{3}{4}\theta^4$. Let $\mathcal{L}'$ be the set of survived buckets.
{Construct-Examples}
8: Let $\mathcal{M} = \{x_1, x_2, \ldots, x_m\}$.
9: Define the length-m column vector $\mathcal{F}$ by setting $\mathcal{F}_x = f(\boldsymbol{x})$ for each $\boldsymbol{x} \in \mathcal{M}$.
10: Draw a list $\mathcal{M}' = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_m\}$ of m uniformly random strings from $\mathbb{F}_2^n$.
{Define $m \times |\mathcal{L}'|$ matrices $\mathcal{Q}', \mathcal{Q}''$ and then $\mathcal{Q}$ as follows}
11: **for** each $i \in [m]$ and $C \in \mathcal{L}'$ **do**
12: Estimate $\mathrm{P}_C f(\boldsymbol{y}_i)$ and $\mathrm{P}_C f(x_i + \boldsymbol{y}_i)$ to within $\pm\frac{1}{8}\theta$ with confidence $1 - \frac{1}{200|\mathcal{L}'|m}$. Set $\mathcal{Q}'_{i,C}$ and $\mathcal{Q}''_{i,C}$ to be the sign of them, respectively.
13: Set $\mathcal{Q}_{i,C} = \mathcal{Q}'_{i,C} \cdot \mathcal{Q}''_{i,C}$.

From the last property, there are at most $\frac{4}{\theta^2}$ many buckets in $\mathcal{L}'$.

We proceed to analyze Construct-Examples (Steps 8–13). For each heavy bucket $C \in \mathcal{L}'$, define $S_C(f)$ to be the "small projection": $S_C(f)(x) = \sum_{\alpha \in C \setminus \{\alpha(C)\}} \widehat{f}(\alpha)\chi_\alpha(x)$.

Lemma 7. *For each heavy bucket C, $\Pr_{\boldsymbol{x}}[|S_C(f)(\boldsymbol{x})| \geq \frac{1}{4}\theta] \leq \frac{\theta^2}{800m}$.*

Since $\mathrm{P}_C(f)(x) = \widehat{f}(\alpha(C))\chi_{\alpha(C)}(x) + S_C(f)(x)$ and $|\widehat{f}(\alpha(C))|$ is much larger than $|S_C(f)(x)|$ from Lemmas 6 and 7, we have that $\mathrm{sgn}(\widehat{f}(\alpha(C))\chi_\alpha(\boldsymbol{x})$ is equal to $\mathrm{sgn}(P_C(f(\boldsymbol{x}))$ with high probability:

Lemma 8. *For each heavy bucket C, $\Pr[|\mathrm{P}_C(f)(\boldsymbol{x})| < \frac{1}{4}\theta$ or $\mathrm{sgn}(\mathrm{P}_C(f)(\boldsymbol{x})) \neq \mathrm{sgn}(\widehat{f}(\alpha(C))\chi_{\alpha(C)}(\boldsymbol{x}))] \leq \frac{\theta^2}{800m}$.*

From Lemma 8, each of the at most $2|\mathcal{L}'|m$ estimates are correct except with probability $\frac{1}{200|\mathcal{L}'|m}$, so these estimations add at most $\frac{1}{100}$ to the failure probability. Thus, each fixed entry of $\mathcal{Q}'$ and $\mathcal{Q}''$ is correct except with probability at most $\frac{\theta^2}{800m}$. We use basic self-correction to form the matrix $\mathcal{Q}$; a fixed entry of $\mathcal{Q}$ (a proposed value of $\chi_{\alpha(C)}$) is correct except with probability at most $\frac{\theta^2}{400m}$. There are only $\frac{4}{\theta^2}$ heavy buckets at most, so except with probability at most $\frac{1}{100m}$, a fixed row of $\mathcal{Q}$ is correct on any specific example. Thus, if we draw a set of at most m (implicit) examples, the entire matrix $\mathcal{Q}$ is correct except with

Algorithm 2. Tolerant g-isomorphism testing to a function with small spectral norm

Parameters: s, $g : \mathbb{F}_2^n \to \{-1, 1\}$ with $\widehat{\|g\|}_1 \leq s$, ϵ

1: Run Algorithm 1 with $\theta = \frac{\epsilon}{6s}$, $m = \widetilde{O}(\frac{s^2}{\epsilon^2})$, and $\mathcal{M}$ a uniformly random subset of $\mathbb{F}_2^n$ with $|\mathcal{M}| = m$ to get $(\mathcal{Q}, \mathcal{F})$.
2: **for** each $C \in \mathcal{L}'$ **do**
3: Use $(\mathcal{Q}, \mathcal{F})$ to estimate $\widehat{f}(\alpha(C))$ within $\pm\frac{\epsilon}{3s}$ with confidence $1 - \frac{1}{100|\mathcal{L}'|}$ and call it $\widetilde{\widehat{f}}(\alpha(C))$.
4: Set $\widetilde{\widehat{f}}(\alpha) = 0$ for all $\alpha \in \mathbb{F}_2^n \setminus \alpha(\mathcal{L}')$.
5: Accept if there is a nonsingular linear transformation A such that $\sum_{\beta \in \mathbb{F}_2^n} \widetilde{\widehat{f \circ A}}(\beta)\widehat{g}(\beta) \geq 1 - \epsilon$, and reject otherwise.

probability at most $\frac{1}{100}$, assuming all the previous estimates were correct. Overall, the total failure probability is at most $\frac{1}{100} + \frac{2}{100} + \frac{1}{100} + \frac{1}{100} + \frac{1}{100} = \frac{6}{100}$ as claimed; this establishes Lemma 3.

Enumerate the buckets of $\mathcal{L}'$ as $\{C_1, C_2, \ldots, C_{|\mathcal{L}'|}\}$. Similar to how we define $\alpha(C)$, let $\alpha(\mathcal{L}') = \{\alpha(C) \mid C \in \mathcal{L}'\}$. The final matrix $\mathcal{Q}$ and vector $\mathcal{F}$ are, except with probability at most $\frac{6}{100}$, of the following form:

$(\mathcal{Q} \mid \mathcal{F})$	C_1	C_2	$\ldots$	$C_{\lvert\mathcal{L}'\rvert}$	$f(x)$
x_1	$\chi_{\alpha(C_1)}(x_1)$	$\chi_{\alpha(C_2)}(x_1)$	$\cdots$	$\chi_{\alpha(C_{\lvert\mathcal{L}'\rvert})}(x_1)$	$f(x_1)$
x_2	$\chi_{\alpha(C_1)}(x_2)$	$\chi_{\alpha(C_2)}(x_2)$	$\cdots$	$\chi_{\alpha(C_{\lvert\mathcal{L}'\rvert})}(x_2)$	$f(x_2)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_m	$\chi_{\alpha(C_1)}(x_m)$	$\chi_{\alpha(C_2)}(x_m)$	$\cdots$	$\chi_{\alpha(C_{\lvert\mathcal{L}'\rvert})}(x_m)$	$f(x_m)$

where $\mathcal{M} = \{x_1, x_2, \ldots, x_m\}$.

The query complexity of this algorithm is dominated by the estimates in Steps 2 and 3. Since the number of buckets is $\Omega(\frac{m^4}{\theta^{12}})$, we need $\widetilde{O}(\frac{m^4}{\theta^{16}})$ queries in total to make one estimate to tolerance $\pm\frac{\theta^2}{4}$ for each bucket.

4 Tolerantly Testing Isomorphism to Functions with Small Spectral Norm

In order to talk about tolerantly testing g-isomorphism, say that f is (τ, g)-*isomorphic* if f is τ-close to a g-isomorphic function. In this section, we show the following tolerant tester for g-isomorphism.

Theorem 5. *Let $g : \mathbb{F}_2^n \to \{-1, 1\}$ be a function with $\widehat{\|g\|}_1 \leq s$. Then, there is an algorithm with query complexity $\widetilde{O}((s/\epsilon)^{24})$ such that, given a function f, it accepts with probability at least $\frac{2}{3}$ when f is $(\frac{\epsilon}{3}, g)$-isomorphic and rejects with probability at least $\frac{2}{3}$ when f is not $(\frac{2\epsilon}{3}, g)$-isomorphic.*

Our algorithm is given in Algorithm 2. We start by running Algorithm 1 with $\theta = \frac{\epsilon}{6s}$ and $m = \widetilde{O}(\frac{s^2}{\epsilon^2})$, and then we run an extra step: we use the examples to estimate each heavy Fourier coefficient to within $\frac{\epsilon}{3s}$. We estimate every other Fourier coefficient to be 0. Because we set $\theta = \frac{\epsilon}{6s}$, all estimates are correct to within $\frac{\epsilon}{3s}$, (except with probability at most $\frac{1}{100}$, if we draw enough examples) because the small Fourier coefficients estimated to be 0 without any examples have magnitude at most $\frac{\epsilon}{6s}$. The probability that any of the $|\mathcal{L}'|$ estimates is outside the stated interval is at most $\frac{1}{100}$.

We will assume that we have g given as a multilinear polynomial; $g = \sum_{\beta \in \mathbb{F}_2^n} \widehat{g}(\beta)\chi_\beta$. The algorithm tries to find the best match of the estimates of the Fourier coefficients found to the Fourier coefficients of g. An issue here is that we do not actually know $\alpha(C)$'s and hence it is not immediate how to execute Step 5. A key insight is that, to discuss linear transformations applied to these heavy coefficients, it suffices to determine the linear relationships between the coefficients. Proofs of the following lemmas are deferred to the full version.

Lemma 9. *Let $B = \{C_1, C_2, \ldots, C_k\} \subseteq \mathcal{L}'$ be a minimal collection of buckets such that $\alpha(\mathcal{L}') \subseteq \mathrm{span}(\{\alpha(C_1), \alpha(C_2), \ldots, \alpha(C_k)\})$.*

Suppose that $\mathcal{M}$ is chosen uniformly at random in Implicit Sieve*. Then the rows of $\mathcal{Q}$ are uniformly distributed when restricted to the columns in B, and every other entry in a fixed row is determined by the setting of the bits in the columns in B.*

Lemma 10. *Let $\mathcal{S} = \{S \mid S \subseteq \alpha(\mathcal{L}') \text{ and } \sum_{\alpha \in S} \alpha = 0^n\}$, where the sum is in the $\mathbb{F}_2^n$ sense. Except with probability at most $\frac{1}{100}$ over the construction of $\mathcal{Q}$, we can find sets of buckets corresponding to $\mathcal{S}$ from $\mathcal{Q}$.*

Lemma 11. *The algorithm is able to perform Step 5 by enumerating over all linear transformations without using any further queries.*

The following lemma shows that if f is close to g, then there is a very good match between the Fourier coefficients of f and g.

Lemma 12. *Let $g : \mathbb{F}_2^n \to \{-1, 1\}$ be a fixed function, and suppose f is $(\frac{\epsilon}{3}, g)$-isomorphic; further, let A be a linear transformation such that $f \circ A$ is $\frac{\epsilon}{3}$-close to g. Then $\sum_{\alpha \in \mathrm{supp}(g)} \widehat{f \circ A}(\alpha)\widehat{g}(\alpha) \geq 1 - \frac{2\epsilon}{3}$.*

Proof. Because $f \circ A$ and g are $\{+1, -1\}$-valued, $\Pr_{\boldsymbol{x}}[(f \circ A)(\boldsymbol{x}) \neq g(\boldsymbol{x})] = \frac{1}{2} + \frac{1}{2}\mathbf{E}_{\boldsymbol{x}}[(f \circ A)(\boldsymbol{x})g(\boldsymbol{x})]$. It follows that if $(f \circ A)$ and g are $\frac{\epsilon}{3}$-close, then $\mathbf{E}_{\boldsymbol{x}}[(f \circ A)(\boldsymbol{x})g(\boldsymbol{x})] \geq 1 - \frac{2\epsilon}{3}$. But $\mathbf{E}_{\boldsymbol{x}}[(f \circ A)(\boldsymbol{x})g(\boldsymbol{x})] = \sum_{\alpha \in \mathbb{F}_2^n} \widehat{f \circ A}(\alpha)\widehat{g}(\alpha)$, finishing the proof.

Theorem 5 follows from the following two lemmas.

Lemma 13. *Suppose that f is $(\frac{\epsilon}{3}, g)$-isomorphic, where $\widehat{\|g\|}_1 \leq s$. Then the algorithm accepts with probability at least $\frac{2}{3}$.*

Proof. We assume that $(\mathcal{Q}, \mathcal{F})$ is found correctly, that estimates in Algorithm 2 are correct, and that all linear dependencies are discovered. The probability of failing at least one of these is at most $\frac{6}{100} + \frac{1}{100} + \frac{1}{100} = \frac{8}{100}$.

Because f is $(\frac{\epsilon}{3}, g)$-isomorphic, there is a nonsingular linear transformation A such that $f \circ A$ is $\frac{\epsilon}{3}$-close to g. By Lemma 12, $\sum_{\beta \in \mathrm{supp}(g)} \widehat{f \circ A}(\beta)\widehat{g}(\beta) \geq 1 - \frac{2\epsilon}{3}$.

We only have estimates for each $\widehat{f \circ A}(\beta)$, and each could be off by at most $\frac{\epsilon}{3s}$. Because $\widehat{\|g\|}_1 \leq s$, using these estimates can reduce this "dot product" by at most $\frac{\epsilon}{3}$. Thus $\sum_{\beta \in \mathbb{F}_2^n} \widetilde{f \circ A}(\beta)\widehat{g}(\beta) \geq 1 - \frac{2\epsilon}{3} - \frac{\epsilon}{3s}\widehat{\|g\|}_1 \geq 1 - \epsilon$, so the algorithm will accept. Thus, the algorithm accepts with probability at least $1 - \frac{8}{100} > \frac{2}{3}$.

Lemma 14. *Suppose that the algorithm accepts with probability at least $\frac{1}{3}$. Then f is $(\frac{2\epsilon}{3}, g)$-isomorphic.*

Proof. Again, we assume that $(\mathcal{Q}, \mathcal{F})$ is found correctly, that the estimates in Algorithm 2 are correct, and that all linear dependencies are discovered. The probability of failing at least one of these is at most $\frac{6}{100} + \frac{1}{100} + \frac{1}{100} = \frac{8}{100}$. Suppose the algorithm accepts with probability at least $\frac{1}{3} > \frac{8}{100}$. Then, with nonzero probability, the algorithm finds a linear transformation A such that $\sum_{\beta \in \mathbb{F}_2^n} \widetilde{f \circ A}(\beta)\widehat{g}(\beta)$ is at least $1 - \epsilon$. As in the previous claim, each estimate of $\widehat{f \circ A}$ is within $\frac{\epsilon}{3s}$ of the true value, so $\sum_{\beta \in \mathbb{F}_2^n} \widehat{f \circ A}(\beta)\widehat{g}(\beta) \geq 1 - \epsilon - \frac{\epsilon}{3s}\widehat{\|g\|}_1 \geq 1 - \frac{4\epsilon}{3}$. Then, $\sum_{\beta \in \mathbb{F}_2^n} (\widehat{f \circ A}(\beta) - \widehat{g}(\beta))^2 = \sum_{\beta \in \mathbb{F}_2^n} \widehat{f \circ A}(\beta)^2 - 2\sum_{\beta \in \mathbb{F}_2^n} \widehat{f \circ A}(\beta)\widehat{g}(\beta) + \sum_{\beta \in \mathbb{F}_2^n} \widehat{g}(\beta)^2 \leq 2 - 2(1 - \frac{4\epsilon}{3}) = \frac{8\epsilon}{3}$. It follows that $\Pr_{\boldsymbol{x}}[(f \circ A)(\boldsymbol{x}) \neq g(\boldsymbol{x})] = \frac{1}{4}\,\mathbf{E}_{\boldsymbol{x}}[((f \circ A)(\boldsymbol{x}) - g(\boldsymbol{x}))^2] = \frac{1}{4}\sum_{\beta \in \mathbb{F}_2^n} (\widehat{f \circ A}(\beta) - \widehat{g}(\beta))^2 \leq \frac{1}{4} \cdot \frac{8\epsilon}{3} = \frac{2\epsilon}{3}$, and thus f is $(\frac{2\epsilon}{3}, g)$-isomorphic.

Proof (of Theorem 1). Suppose that g is $\frac{\epsilon}{3}$-close to a function h with $\widehat{\|h\|}_1 \leq s$. To test whether the input function f is isomorphic to g, we instead test $(\frac{\epsilon}{3}, h)$-isomorphism using Theorem 5. If f is isomorphic to g, then f is $\frac{\epsilon}{3}$-close to h, and we accept with probability at least $\frac{2}{3}$ from Lemma 13. Suppose that f is ϵ-far from being isomorphic g. Then, f is $\frac{2\epsilon}{3}$-far from being isomorphic to h from the triangle inequality. Thus, we reject with probability at least $\frac{2}{3}$ from Lemma 14.

References

1. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM J. Comput. 25(2), 252–271 (1996)
2. Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connection to learning and approximation. J. ACM 45(4), 653–750 (1998)
3. Ron, D.: Algorithmic and analysis techniques in property testing. Foundations and Trends in Theoretical Computer Science 5, 73–205 (2010)
4. Rubinfeld, R., Shapira, A.: Sublinear time algorithms. Electronic Colloquium on Computational Complexity (ECCC) 18, TR11-013 (2011)
5. Goldreich, O. (ed.): Property Testing. LNCS, vol. 6390. Springer, Heidelberg (2010)

6. Alon, N., Fischer, E., Newman, I., Shapira, A.: A combinatorial characterization of the testable graph properties: It's all about regularity. SIAM J. Comput. 39, 143–167 (2009)
7. Bhattacharyya, A., Grigorescu, E., Shapira, A.: A unified framework for testing linear-invariant properties. In: FOCS, pp. 478–487 (2010)
8. Grigorescu, E., Kaufman, T., Sudan, M.: 2-transitivity is insufficient for local testability. In: CCC, pp. 259–267 (2008)
9. Kaufman, T., Sudan, M.: Algebraic property testing: the role of invariance. In: STOC, pp. 403–412 (2008)
10. Sudan, M.: Invariance in property testing. In: Goldreich, O. (ed.) Property Testing. LNCS, vol. 6390, pp. 211–227. Springer, Heidelberg (2010)
11. Fischer, E.: The difficulty of testing for isomorphism against a graph that is given in advance. In: STOC, pp. 391–397 (2004)
12. Fischer, E., Newman, I.: Testing versus estimation of graph properties. SIAM J. Comput. 37(2), 482–501 (2008)
13. Gopalan, P., O'Donnell, R., Servedio, R., Shpilka, A., Wimmer, K.: Testing fourier dimensionality and sparsity. SIAM J. Comput. 40(4), 1075–1100 (2011)
14. Diakonikolas, I., Lee, H., Matulef, K., Onak, K., Rubinfeld, R., Servedio, R., Wan, A.: Testing for concise representations. In: FOCS, pp. 549–558 (2007)
15. Goldreich, O., Levin, L.: A hard-core predicate for all one-way functions. In: STOC, pp. 25–32 (1989)
16. Kushilevitz, E., Mansour, Y.: Learning decision trees using the fourier spectrum. In: STOC, pp. 455–464 (1991)
17. Fischer, E., Kindler, G., Ron, D., Safra, S., Samorodnitsky, A.: Testing juntas. Journal of Computer and System Sciences 68(4), 753–787 (2004)
18. Blais, E., Weinstein, A., Yoshida, Y.: Partially symmetric functions are efficiently isomorphism-testable. In: FOCS, pp. 551–560 (2012)
19. Chakraborty, S., Fischer, E., García-Soriano, D., Matsliah, A.: Junto-symmetric functions, hypergraph isomorphism, and crunching. In: CCC, pp. 148–158 (2012)
20. Grigorescu, E., Wimmer, K., Xie, N.: Tight lower bounds for testing linear isomorphism (manuscript)
21. Bhattacharyya, A., Fischer, E., Hatami, H., Hatami, P., Lovett, S.: Every locally characterized affine-invariant property is testable. In: STOC (to appear, 2013)
22. Fischer, E., Matsliah, A.: Testing graph isomorphism. In: SODA, pp. 299–308 (2006)
23. Newman, I., Sohler, C.: Every property of hyperfinite graphs is testable. In: STOC, pp. 675–684 (2011)

Author Index