Luca Aceto Ivan Damgård
Leslie Ann Goldberg Magnús M. Halldórsson
Anna Ingólfsdóttir Igor Walukiewicz (Eds.)

LNCS 5125

Automata, Languages and Programming

35th International Colloquium, ICALP 2008
Reykjavik, Iceland, July 2008
Proceedings, Part I

Part I

Springer

Lecture Notes in Computer Science 5125

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

Luca Aceto Ivan Damgård
Leslie Ann Goldberg
Magnús M. Halldórsson
Anna Ingólfsdóttir Igor Walukiewicz (Eds.)

Automata, Languages and Programming

35th International Colloquium, ICALP 2008
Reykjavik, Iceland, July 7-11, 2008
Proceedings, Part I

Springer

Volume Editors

Luca Aceto
Magnús M. Halldórsson
Anna Ingólfsdóttir
Reykjavik University, School of Computer Science
Kringlan 1, 103 Reykjavík, Iceland
E-mail: {luca, mmh, annai}@ru.is

Ivan Damgård
University of Aarhus, Department of Computer Science, IT-Parken
Åbogade 34, 8200 Århus N, Denmark
E-mail: ivan@daimi.au.dk

Leslie Ann Goldberg
University of Liverpool, Department of Computer Science
Ashton Building, Liverpool L69 3BX, UK
E-mail: l.a.goldberg@liverpool.ac.uk

Igor Walukiewicz
Université de Bordeaux-1, LaBRI
351, Cours de la Libération, 33405 Talence cedex, France
E-mail: igw@labri.fr

Library of Congress Control Number: 2008930136

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

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

ISSN 0302-9743
ISBN-10 3-540-70574-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-70574-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12322985 06/3180 5 4 3 2 1 0

Preface

ICALP 2008, the 35th edition of the International Colloquium on Automata, Languages and Programming, was held in Reykjavik, Iceland, July 7–11, 2008. ICALP is a series of annual conferences of the European Association for Theoretical Computer Science (EATCS) which first took place in 1972. This year, the ICALP program consisted of the established Track A (focusing on algorithms, automata, complexity and games) and Track B (focusing on logic, semantics and theory of programming), and of the recently introduced Track C (focusing on security and cryptography foundations).

In response to the call for papers, the Program Committees received 477 submissions, the highest ever: 269 for Track A, 122 for Track B and 86 for Track C. Out of these, 126 papers were selected for inclusion in the scientific program: 70 papers for Track A, 32 for Track B and 24 for Track C. The selection was made by the Program Committees based on originality, quality, and relevance to theoretical computer science. The quality of the manuscripts was very high indeed, and many deserving papers could not be selected.

ICALP 2008 consisted of five invited lectures and the contributed papers. This volume of the proceedings contains all contributed papers presented at the conference in Track A, together with the papers by the invited speakers S. Muthukrishnan (Google, USA) and Bruno Courcelle (Labri, Universitè Bordeaux, France). A companion volume contains all contributed papers presented in Track B and Track C together with the papers by the invited speakers Ran Canetti (IBM T.J. Watson Research Center and MIT, USA) and Javier Esparza (Technische Universität München, Germany). The program had an additional invited lecture by Peter Winkler (Dartmouth, USA), which does not appear in the proceedings.

The following workshops were held as satellite events of ICALP 2008:

ALGOSENSORS 2008 – 4th International Workshop on Algorithmic Aspects of Wireless Sensor Networks
CL&C 2008 – Second International Workshop on Classical Logic and Computation
FOCLASA 2008 – 7th International Workshop on the Foundations of Coordination Languages and Software Architectures
FIMN 2008 – Foundations of Information Management in Networks
FBTC 2008 – From Biology To Concurrency and Back
ICE 2008 – Interaction and Concurrency Experience
MatchUP 2008 – Matching Under Preferences - Algorithms and Complexity
MSFP 2008 – Second Workshop on Mathematically Structured Functional Programming
PAuL 2008 – Third International Workshop on Probabilistic Automata and Logics
QPL/DCM 2008 – 5th Workshop on Quantum Physics and Logic and 4th Workshop on Development of Computational Models

SOS 2008 – 5th Workshop on Structural Operational Semantics
IMAGINE 2008 – Second International Workshop on Mobility, Algorithms and Graph Theory in Dynamic Networks
DYNAMO 2008 – Second Training School on Algorithmic Aspects of Dynamic Networks

We wish to thank all authors who submitted extended abstracts for consideration, the Program Committees for their scholarly effort, and all referees who assisted the Program Committees in the evaluation process.

Thanks to the sponsors (CCP Games, Icelandair, IFIP TC1, Teymi) for their support, and to Reykjavik University for hosting ICALP 2008. We are also grateful to all members of the Organizing Committee in the School of Computer Science and to the Facilities and Technical staff of Reykjavik University. Thanks to Andrei Voronkov and Shai Halevi for writing the conference-management systems EasyChair and Web-Submission-and-Review software, which were used in handling the submissions and the electronic PC meeting as well as in assisting in the assembly of the proceedings.

May 2008

Luca Aceto
Ivan Damgård
Leslie Ann Goldberg
Magnús M. Halldórsson
Anna Ingólfsdóttir
Igor Walukiewicz

Organization

Program Committee

Track A

Michael Bender, State University of New York at Stony Brook, USA
Magnus Bordewich, Durham University, UK
Lenore Cowen, Tufts University, USA
Pierluigi Crescenzi, Università di Firenze, Italy
Artur Czumaj, University of Warwick, UK
Edith Elkind, University of Southampton, UK
David Eppstein, University of California at Irvine, USA
Leslie Ann Goldberg, University of Liverpool, UK (Chair)
Martin Grohe, Humboldt-Universität zu Berlin, Germany
Giuseppe F. Italiano, Università di Roma "Tor Vergata", Italy
Christos Kaklamanis, University of Patras, Greece
Peter Bro Miltersen, University of Aarhus, Denmark
Michael Mitzenmacher, Harvard University, USA
Ian Munro, University of Waterloo, Canada
Ryan O'Donnell, Carnegie Mellon University, USA
Dana Ron, Tel-Aviv University, Israel
Tim Roughgarden, Stanford University, USA
Christian Scheideler, Technische Universität München, Germany
Christian Sohler, University of Paderborn, Germany
Luca Trevisan, University of California at Berkeley, USA
Berthold Voecking, RWTH Aachen University, Germany
Gerhard Woeginger, Eindhoven University of Technology, The Netherlands

Track B

Parosh Abdulla, Uppsala University, Sweden
Luca de Alfaro, University of California, Santa Cruz, USA
Christel Baier, Technische Universität Dresden, Germany
Giuseppe Castagna, Université Paris 7, France
Rocco de Nicola, Università di Firenze, Italy
Javier Esparza, Technische Universität München, Germany
Marcelo Fiore, University of Cambridge, UK
Erich Grädel, RWTH Aachen, Germany
Jason Hickey, California Institute of Technology, USA
Martin Hofmann, Ludwig-Maximilians-Universität München, Germany
Hendrik Jan Hoogeboom, Leiden University, The Netherlands

Radha Jagadeesen, DePaul University, USA
Madhavan Mukund, Chennai Mathematical Institute, India
Luke Ong, Oxford University, UK
Dave Schmidt, Kansas State University, USA
Philippe Schnoebelen, ENS Cachan, France
Igor Walukiewicz, Labri, Université Bordeaux, France (Chair)
Mihalis Yannakakis, Columbia University, USA
Wieslaw Zielonka, Université Paris 7, France

Track C

Christian Cachin, IBM Research Zürich, Switzerland
Jan Camenisch, IBM Research Zürich, Switzerland
Ivan Damgård, University of Aarhus, Denmark (Chair)
Stefan Dziembowski, Università di Roma "La Sapienza", Italy
Dennis Hofheinz, CWI Amsterdam, The Netherlands
Susan Hohenberger, Johns Hopkins University, USA
Yuval Ishai, Technion Haifa, Israel
Lars Knudsen, DTU Copenhagen, Denmark
Arjen Lenstra, EPFL Lausanne, Switzerland
Anna Lysyanskaya, Brown University, USA
Rafael Pass, Cornell University, USA
David Pointcheval, ENS Paris, France
Dominique Unruh, Saarland University, Germany
Serge Vaudenay, EPFL Lausanne, Switzerland
Bogdan Warinschi, Bristol University, UK
Douglas Wikström, KTH Stockholm, Sweden
Stefan Wolf, ETH Zürich, Switzerland

Organizing Committee

Luca Aceto, Reykjavik University (Conference Chair)
Bjarni V. Halldórsson, Reykjavik University (Workshop Co-chair)
Magnús M. Halldórsson, Reykjavik University (Conference Chair)
Anna Ingólfsdóttir, Reykjavik University (Conference Chair)
MohammadReza Mousavi, Eindhoven University of Technology (Workshop Co-chair)

Sponsoring Institutions

CCP Games
Icelandair
IFIP TC1
Reykjavik University
Teymi

Referees

Dimitris Achlioptas
Isolde Adler
Pavan Aduri
Panos Aliferis
Andris Ambainis
Christoph Ambühl
Aris Anagnostopoulos
Spyros Angelpopolos
Chrisil Arackaparambil
James Aspnes
Albert Atserias
Peter Auer
Vincenzo Auletta
Giorgio Ausiello
Chen Avin
Yossi Azar
Nikhil Bansal
Sanjoy Baruah
Tuğkan Batu
Niel de Beaudrap
Luca Becchetti
Paul Bell
Michael Bender
Petra Berenbrink
Anna Bernasconi
Nadja Betzler
Olaf Beyersdorff
Vittorio Bilo
Eric Blais
Avrim Blum
Johannes Blömer
Hans Bodlaender
Andrej Bogdanov
Mikolaj Bojanczyk
Paul Bonsma
Endre Boros
Mark Braverman
Tomas Brazdil
Patrick Briest
Andre Brinkmann
Harry Buhrman
David Bunde
Jonathan Buss
Tiziana Calamoneri
Ioannis Caragiannis
Marco Cesati
Chandra Chekuri
Eric Chen
Ning Chen
Jianer Chen
Qi Cheng
Giorgos Christodoulou
Andrea Clementi
Amin Coja-Oghlan
Vincent Conitzer
Colin Cooper
Graham Cormode
Bruno Courcelle
Andy Curtis
Victor Dalmau
Constantinos Daskalakis
Giuseppe Di Battista
Gabriele Di Stefano
Florian Diedrich
Martin Dietzfelbinger
Irit Dinur
Shahar Dobzinski
Michael Dom
Frederic Dorn
Reza Dorri-Giv
Shaddin Dughmi
Dominic Dumrauf
Stephane Durocher
Christoph Durr
Martin Dyer
Kord Eickmeyer
Friedrich Eisenbrand
Robert Elsaesser
Matthias Englert
Amir Epstein
Leah Epstein
Funda Ergun
Jeff Erickson
Thomas Erlebach
Kousha Etessami
Guy Even
Eyal Even-Dar
Alex Fabrikant
Piotr Faliszewski
Angelo Fanelli
Martin Farach-Colton
Arash Farzan
Henning Fernau
Jiri Fiala
Jeremy Fineman
Irene Finocchi
Eldar Fischer
Felix Fischer
Simon Fischer
Michele Flammini
Abraham Flaxman
Lisa Fleischer
Fedor Fomin
Lance Fortnow
Dimitris Fotakis
Pierre Fraigniaud
Paolo Giulio Franciosa
Gudmund Frandsen
Anna Frid
Tom Friedetzky
Alan Frieze
Martin Fürer
Peter Gacs
Travis Gagie
Anna Gal
Clemente Galdi
Nicola Galesi
Luisa Gargano
Bill Gasarch
Serge Gaspers
Dmitry Gavinsky
Joachim Gehweiler
Blaise Genest
Loukas Georgiadis
Stefanie Gerke
Seth Gilbert
Christian Glasser
Wayne Goddard
Paul Goldberg
Oded Goldreich
Daniel Gottesman
Vipul Goyal

Fabrizio Grandoni
Catherine Greenhill
Alexander Grigoriev
Roberto Grossi
Jens Groth
Magdalena Grüber
Jiong Guo
Anupam Gupta
Venkatesan Guruswami
Vladimir Gurvich
Falk Hüffner
Esther Haenggi
Torben Hagerup
MohammadTaghi Hajiaghayi
Angele Hamel
Kristoffer Arnsfelt Hansen
Ramesh Hariharan
Nick Harvey
Soha Hassoun
Elad Hazan
Lisa Hellerstein
Benjamin Hescott
Jan van den Heuvel
Thomas Holenstein
Peter Hoyer
Chien-Chung Huang
Yumei Huo
Thore Husfeldt
Martin Höfer
Samuel Ieong
Nicole Immorlica
Sandy Irani
Kazuo Iwama
Riko Jacob
Markus Jalsenius
Maurice Jansen
Klaus Jansen
Peter Jeavons
Mark Jerrum
Albert Jiang
Lisa Kaati
Valentine Kabanets
Christos Kaklamanis
Panagiotis Kanellopoulos
Viggo Kann
Haim Kaplan
Sanjiv Kapoor
George Karakostas
Howard Karloff
Marek Karpinski
Telikepalli Kavitha
Steven Kelk
Hans Kellerer
Julia Kempe
David Kempe
Iordanis Kerenidis
Sanjeev Khanna
Valerie King
Carl Kingsford
Ralf Klasing
Hartmut Klauck
Robert Kleinberg
Lasse Kliemann
Bettina Klinz
Adam Klivans
Joachim Kneis
Ker-i Ko
Petr Kolman
Spyros Kontogiannis
Tsvi Kopelowitz
Swastik Kopparty
Nitish Korula
Michal Koucky
Elias Koutsoupias
Lukasz Kowalik
Darek Kowalski
Dariusz Kowalski
Robi Krauthgamer
Stephan Kreutzer
Danny Krizanc
Andrei Krokhin
Sven Krumke
Piotr Krysta
Daniela Kuehn
Ravi Kumar
Amit Kumar
Michal Kunc
Daniel Kuntze
Orna Kupferman
Dietrich Kuske
Shay Kutten
Johannes Köbler
Oded Lachish
Christiane Lammersen
Michael Langberg
Alexander Langer
John Langford
Luigi Laura
Ranko Lazic
Homin Lee
Hing Leung
David Levin
Asaf Levin
Ming Li
Andrzej Lingas
Maciej Liskiewicz
Christof Loeding
Markus Lohrey
Michele Loreti
Vadim Lozin
Eyal Lubetzky
Fabrizio Luccio
Yoad Lustig
Christof Löding
Bin Ma
Michael Mahoney
Elitza Maneva
David F. Manlove
Giovanni Manzini
Alberto Marchetti-Spaccamela
Russell Martin
Dániel Marx
Monaldo Mastrolilli
Jiri Matousek
Marios Mavronicolas
Andrew McGregor
Klaus Meer
Jan Mehler
Aranyak Mehta
Dieter van Melkebeek
Raghu Meka

Ulrich Meyer
Adam Meyerson
Zoltan Miklos
Miriam Di Ianni
Vahab Mirrokni
Bojan Mohar
Michael Molloy
Morteza Monemizadeh
Burkhard Monien
Angelo Monti
Michele Mosca
Luca Moscardelli
Georg Moser
Hannes Moser
Elchanan Mossel
Shay Mozes
S. Muthukrishnan
Veli Mäkinen
Stefanie Naewe
Daniel Nagaj
Ashwin Nayak
Jaroslav Nesetril
Vincent Nesme
Ilan Newman
Pat Nicholson
Carlo Nocentini
Lars Olbrich
Svetlana Olonetsky
Krzysztof Onak
Friedrich Otto
Rasmus Pagh
Linda Pagli
Alessandro Panconesi
Rina Panigrahy
Evi Papaioannou
Mihai Patrascu
Chris Peikert
David Peleg
Rudi Pendavingh
Paolo Penna
Carlos Perez-Delgado
Giuseppe Persiano
Andrea Pietracaprina
Alexei Piunovskiy
Wojciech Plandowski
Igor Potapov
Daniel Preda
Geppino Pucci
Rosario Pugliese
Jaikumar Radhakrishnan
Tomasz Radzik
Rajmohan Rajaraman
Jörg Rambau
Anup Rao
Robert Raussendorf
R. Ravi
Oded Regev
David Richerby
Liam Roditty
Gianluca Rossi
Wojciech Rytter
Harald Räcke
Heiko Röglin
Mohammad Salavatipour
Alejandro Salinger
Piotr Sankowski
Rahul Santhanam
Thomas Sauerwald
Petr Savicky
Nitin Saxena
Christian Schaffner
Rob Schapire
Arthur Schmidt
Henning Schnoor
Nicole Schweikardt
Robert Schweller
Thomas Schwentick
Jacob Scott
Danny Segev
Rocco Servedio
C. Seshadhri
Jiri Sgall
Hadas Shachnai
Ronen Shaltiel
Ron Shamir
Asaf Shapira
Sasha Sherstov
Yaoyun Shi
Nahum Shimkin
Amir Shpilka
Adi Shraibman
Anastasios Sidiropoulos
Riccardo Silvestri
Matthew Skala
Alexander Skopalik
Miroslava Sotakova
Holger Spakowski
Robert Spalek
Bettina Speckmann
Aravind Srinivasan
Rob van Stee
Ken Steiglitz
Martin Strauss
Mukund Sundararajan
Maxim Sviridenko
Troels Bjerre Sørensen
Prasad Tetali
Dimitrios Thilikos
Wolfgang Thomas
Marc Thurley
Alex Tiskin
Isaac K. K. To
Ben Toner
Hanjo Täubig
Walter Unger
Ugo Vaccaro
Salil Vadhan
Matt Valeriote
Gregory Valiant
Virginia Vassilevska
Carmine Ventre
Adrian Vetta
Thomas Vidick
Eric Vigoda
Emanuele Viola
Ivan Visconti
Nisheeth Vishnoi
Paola Vocca
Heribert Vollmer
Jan Vondrak
Sergei Vorobyov
Johannes Waldmann
Xin Wang
John Watrous

Renato Werneck
Matthias Westermann
Mark Weyer
Peter Widmayer
Thomas Wilke
Ryan Williams
David Williamson
Anthony Wirth
Pawel Wocjan
Philipp Woelfel
Ronald de Wolf
Prudence W.H. Wong
Qiqi Yan
Shi Yaoyun
Shengyu Zhang
Hairong Zhao
Martin Ziegler
Marius Zimand
David Zuckerman
Uri Zwick

Table of Contents – Part I

Invited Lectures

Track A: Algorithms, Automata, Complexity, and Games

Complexity: Boolean Functions and Circuits

Data Structures

Random Walks and Random Structures

Design and Analysis of Algorithms

Scheduling

Codes and Coding

Coloring

Randomness in Computation

Online and Dynamic Algorithms

Approximation Algorithms

Property Testing

Parameterized Algorithms and Complexity

Graph Algorithms

Computational Complexity

Games and Automata

Group Testing, Streaming, and Quantum

Algorithmic Game Theory

Quantum

Table of Contents – Part II

Invited Lectures

Track B: Logic, Semantics, and Theory of Programming

Bounds

Distributed Computation

Real-Time and Probabilistic Systems

Logic and Complexity

Words and Trees

Nonstandard Models of Computation

Reasoning about Computation

Verification

Track C: Security and Cryptography Foundations

Theory

Secure Computation

Two-Party Protocols and Zero-Knowledge

Encryption with Special Properties/Quantum Cryptography

Various Types of Hashing

Public-Key Cryptography/Authentication

Graph Structure and Monadic Second-Order Logic: Language Theoretical Aspects*

Bruno Courcelle

Université Bordeaux-1, LaBRI, CNRS
Institut Universitaire de France
351, Cours de la Libération
33405, Talence cedex, France
courcell@labri.fr

Abstract. *Graph structure* is a flexible concept covering many different types of graph properties. Hierarchical decompositions yielding the notions of *tree-width and clique-width*, expressed by terms written with appropriate graph operations and associated with *Monadic Second-order Logic* are important tools for the construction of Fixed-Parameter Tractable algorithms and also for the extension of methods and results of Formal Language Theory to the description of sets of finite graphs. This informal overview presents the main definitions, results and open problems and tries to answer some frequently asked questions.

Tree-width and *monadic second-order (MS) logic* are well-known tools for constructing *fixed-parameter tractable (FPT)* algorithms taking tree-width as parameter. *Clique-width* is, like tree-width, a complexity measure of graphs from which FPT algorithms can be built, in particular for problems specified in MS logic. These notions are thus essential for constructing (at least theoretically) tractable algorithms but also in the following three research fields:

- the study of the structure of graphs excluding induced subgraphs, minors or *vertex-minors* (a notion related to clique-width, see [48] or [18]);
- the extension of *language theoretical notions* in order to describe and to transform sets of finite and even countable graphs;
- the investigation of classes of finite and countable graphs on which MS logic is decidable.

Although these four research fields have been initially developed independently, they are now more and more related. In particular, new structural results for graph classes have consequences for algorithmic applications (see [7]).

This overview deals only with *finite* graphs, trees and relational structures. There is a rich theory of *countable graphs described by logical formulas, logically defined transformations, equation systems and finite automata*. The survey [1] is a good approach of this theory.

* Supported by the GRAAL project of "Agence Nationale pour la Recherche".

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 1–13, 2008.

1 Graph Structure and Logic

Graph structure is a flexible concept covering many different cases. *Hierarchical decompositions* form an important type of structuring. Those yielding the notions of *tree-width* and *clique-width* can be expressed by terms written with graph operations defined below that generalize the concatenation of words. There exist other types of hierarchical structurings that are useful for establishing results or for algorithmic purposes. Examples are the *modular decomposition* defined by Gallai ([20], [38]), the *split decomposition* (also called *join decomposition*) defined by Cunnigham ([25], [33]), the decomposition in *3-connected components* defined by Tutte ([11], [21]), the *clique-sum decomposition* ([52], [7]).

The existence of an embedding in a fixed surface, or of a homomorphism into a fixed graph (a proper vertex coloring with p colors of a loop-free graph can be defined as a homomorphism of this graph into the clique K_p) is also a type of structure ([8], [43]). Finally, the non-existence in a graph of particular induced subgraphs, minors or vertex-minors is also an important type of structural property. (See [48] for vertex-minors).

There exist nontrivial relations between these different types of structures: graphs without a fixed planar graph P as a minor have tree-width at most $f(size(P))$ for some function f ([50]); graphs embeddable in a fixed surface are characterized by finitely many excluded minors ([51]); forbidding certain induced subgraphs implies bounded clique-width ([15], [16]), just to take a few examples, to which one could add the *restricted duality theorems* of [43]. There are still many open questions concerning comparisons between various types of graph structure.

Monadic second-order logic (MS logic in short) is the extension of first-order logic with quantified variables denoting subsets of the considered relational structures, hence sets of vertices when it is used for graphs, and sets of edges when a graph is represented by its incidence graph. It can express many graph properties: degree constraints, existence of proper colorings with fixed numbers of colors, connectivity, existence of spanning trees with particular properties, absence or existence of particular induced subgraphs or minors. From characterizations by forbidden configurations, one obtains that the sets of cographs, of distance-hereditary graphs, of planar graphs, of graphs embeddable in a fixed surface, of graphs of tree-width bounded by a fixed constant are *MS-definable*, *i.e.*, can be characterized as the finite models of certain MS formulas.

Graph structure notions are related with MS logic in several ways that we can classify under two main titles: *Expressive power of MS logic* and *Construction of algorithms*. We will discuss later the language theoretical aspects.

2 Expressive Power of Monadic Second-Order Logic

It is easy to construct an MS formula expressing that a given graph has no minor or no induced subgraph isomorphic to a fixed finite graph. Hence the set of graphs of tree-width at most k is MS-definable because it is characterized

by finitely many excluded minors. A set of graphs defined by excluded induced subgraphs forming a set that is infinite but MS-definable is also MS-definable. *Perfect graphs* and *comparability graphs* are of this type. Their definitions are not directly translatable into MS formulas ([17], [38], [24]).

Monadic second-order logic can also be used to specify graph transformations. By analogy with the transformations of words and terms called *transductions* in language theory, I call *monadic second-order (MS) transductions* certain transformations of relational structures (hence of trees, graphs and hypergraphs) that can be specified by MS formulas. They generalize the notion of *interpretation* used in model theory and the *rational transductions* (transforming words into sets of words) such that the image of every word is finite. (Due to space limitations, definitions are not given formally. They can be found in the given references and in my book in preparation [3].)

In many situations concerning graph structure, one needs more than a yes or no answer. For an example, that a graph does not contain K_5 or $K_{3,3}$ as a minor implies that it is planar, but this fact does not describe any planar embedding. In other words, we are not only interested in checking that a given graph "has some structure", *e.g.* a tree-decomposition or a planar embedding, but also in having an MS transduction that constructs from the given graph some tree-decomposition or some planar embedding. Such transductions may be difficult to construct. Sometimes, they use edge set quantifications (decomposition in 3-connected components, [21]), and/or auxiliary linear orderings of the input structures. Constructions of planar embeddings, of the modular and split decompositions, of the *chord diagram defining a circle graph*, respectively are considered in this perspective in [22], [20], [25], [26].

Independently of these graph theoretical applications, MS transductions are useful tools for building MS formulas because the *inverse image of an MS-definable set* of graphs or relational structures under an MS transduction is MS-definable.

3 Construction of Algorithms

Books by Downey and Fellows [4] and by Flum and Grohe [6], survey articles by Grohe [7], and by Makowsky [9], and many other articles have popularized the facts that MS expressible graph problems have FPT algorithms for tree-width and clique-width taken as parameters. We will refer by CMS to the extension of MS logic allowing set predicates $Card_p(X)$ expressing that the cardinality of a set X is a multiple of p, by MS_2 to the extension allowing *edge set quantifications* (also called *guarded second-order logic* in [39]) and by CMS_2 to the combination of both extensions.

Theorem 1 (Fixed-Parameter Tractability Theorem): *Every CMS_2 expressible graph problem has a fixed-parameter linear algorithm for tree-width. Every CMS expressible graph problem has a fixed-parameter cubic algorithm for clique-width. These results extend to the counting and optimization problems specified in these extensions of MS logic.*

This result makes particularly interesting the expression of graph properties in CMS or CMS_2 logic. However, monadic second-order logic yields no polynomial algorithm for graphs of unbounded tree-width or clique-width : each level of the polynomial hierarchy contains complete problems expressible in MS logic ([45]).

Algebraic characterizations of tree-width and clique-width.

Tree-width and clique-width are based on graph decompositions that can be expressed with *graph operations*. Such operations generalize the concatenation of words. For defining tree-decompositions, we use graphs with distinguished vertices called *sources* (or *boundary vertices* in [4]) specified by labels. Each label designates a single vertex. The corresponding graph operations are the *parallel-composition* that glues two graphs at their sources with same labels and unary operations that remove or modify source labels. The basic graphs are isolated vertices and graphs with a single edge. Tree-decompositions correspond closely to terms built with these operations and the tree-width of a graph is the minimum number of labels to be used to construct it with these operations.

Clique-width is similar but it is defined with different operations. These operations use also labels but a label may be attached to several vertices. The relevant operations are disjoint union (denoted by $\oplus$), unary operations $add_{a,b}$ that add edges between every vertex labeled by a and every vertex labeled by b, and unary operations that modify labels. The basic graphs are isolated vertices. The clique-width of a graph is the minimum number of labels to be used to construct it with these operations. Whereas words are generated from letters by a single binary operation, operations that use countably many labels (they form a countable set) are needed for generating all graphs,

Is Theorem 1 best possible ?

No. That a graph is Hamiltonian can be decided in polynomial time on graphs of bounded clique-width, although this property is MS_2 but provably not CMS [55]. (The algorithm is not FPT).

However, some converse results do exist : *if every existential monadic second-order property* (3-vertex colorability is an example of such a property) is *decidable in polynomial time on all graphs of a set $\mathcal{C}$ that is closed under taking minors or topological minors then $\mathcal{C}$ has bounded tree-width.* The closure conditions under taking minors cannot be replaced by closure under taking subgraphs. These results are proved in [44].

Is Theorem 1 practical usable ?

Not directly, for several reasons. First, because the algorithms need appropriate hierarchical decompositions of the input graphs. A tree-decomposition of width at most k can be found in linear time if there exists one, but the constant depends exponentially on k, and the algorithm is not implementable [13]. For clique-width, the situation is similar: one can construct in cubic time a clique-width expression of width at most $2^{k+1} - 1$ if the given graph has clique-width at most k by an algorithm derived from [41] and [49], but this algorithm is too complex to be implemented. Deciding if the clique-width of a graph G is at most k for given (G, k) is NP-complete [35]. This problem is polynomial for $k < 4$ and open for $k = 4$. A second difficulty comes from the translation of MS formulas

into the finite automata on terms, on which the FPT algorithms are based. Since short formulas can express complicated properties, these automata have in the worst cases non-elementary sizes in terms of the considered formulas ([37]).

Is the situation hopeless ?

Fortunately not! There exist implementable algorithms producing non optimal but usable tree-decompositions [14]. For clique-width, there exist algorithms based on modular decomposition that can produce in linear time optimal clique-width expressions for graphs from particular classes ([16], [30]). Another possibility consists in inputting graphs that have "natural" tree-decompositions (just because of the nature of the problems they formalize) or graphs produced by context-free grammars with their derivation trees, because derivation trees are hierarchical decompositions of the appropriate types. The second difficulty appears in the general statement intended to cover all formulas, but concrete problems may yield automata of reasonable sizes. Softwares like MONA [42] may be used for graphs defined by terms written with the operations described above, as well as directly for words and terms [54].

What about first-order formulas ?

There are FPT algorithms for model-checking of first-order formulas on certain classes of graphs of unbounded tree-width or clique-width, for instance, on those that have *locally bounded tree-width* or that exclude a fixed graph as a minor ([36], [7]). The structural description of the latter types of graphs used in [52] for proving the *Graph Minor Theorem* finds here unexpected applications. Nešetřil and Ossona de Mendez also apply structure theorems to the verification of first-order formulas expressing graph inclusions [47].

Could one use terms describing graphs written with other operations than those defining tree-width and clique-width, and that would have good "compatibility" with MS logic ?

One could use the *unfolding* operation that transforms a directed graph into the tree of finite paths issued from a specified vertex, because the inverse image under it of a CMS definable set of trees is CMS definable [32]. This operation is used together with MS transductions in order to construct countable graphs having decidable MS theories. These graphs form a hierarchy defined by Caucal (see the survey [1]). This operation has not yet been used to my knowledge to describe sets of finite graphs in a similar way. By using it, one could obtain compact representations of large graphs. (Trees with 2^n nodes can be described by terms of size n that encode directed acyclic graphs.)

4 Language Theoretical Concepts Extended to Sets of Finite Graphs

Two sets of graph operations have been defined above to characterize algebraically tree-width and clique-width. They define two algebraic structures on finite labelled graphs and two types of descriptions of these graphs by terms. Algorithmic applications are based on that, but so are also the language theoretical notions of context-free graph grammars and recognizability.

Context-free grammars as equation systems.

Context-free grammars are usually defined as sets of rewriting rules, however, a classical theorem characterizes the context-free languages as the components of the least solutions of equation systems that are easily built from context-free grammars. These systems define languages in a recursive way in terms of set union, the extension to sets of the concatenation of words, letters and a constant denoting the empty word. Context-free languages are thus the *equational subsets* of the free monoid. This characterization extends to arbitrary algebras (Mezei and Wright [46]), even to those with infinite sets of operations, and in particular to our graph algebras. We obtain thus *context-free graph grammars,* defined formally as *equation systems*, without having to consider derivation sequences, permutation of derivation steps, derivation trees, because these notions are useless or are given for free in the algebraic setting.

Closure under union and under the operations of the algebra, as well as decidability results (emptiness, finiteness) can be established once and for all at the algebraic level. Since systems are finite, algorithms on them make sense although the global set of operations may be infinite.

Two robust classes of context-free graph grammars.

There are many possible graph algebras, and each of them yields a notion of equational set. However, two of them have emerged as particularly robust and interesting. These are the *HR algebra* whose operations are those characterizing tree-width, and the *VR algebra* related similarily to clique-width. The acronym HR stands for *Hyperedge Replacement* and refers to an algebra of graphs, the equational sets of which are exactly those defined independently by *hyperedge replacement (hypergraph) grammars*. Its operations are those from which tree-width can be defined. In particular, the set of graphs of tree-width at most k is HR-equational. Similarly, VR stands for *Vertex Replacement* and refers to other graph grammars that actually generate the VR-equational sets. Its operations have been designed in [28] so that the sets defined by certain grammars be the corresponding equational sets. The set of graphs of clique-width at most k is VR-equational.

In which sense are these classes robust ?

Our robustness criterium is stability under MS transductions, generalizing the fact that the families of context-free and of regular languages are closed under rational transductions. Furthermore, context-free languages are generated by rational transductions from particular context-free languages that describe trees. (Rational transductions are compositions of homomorphisms, inverse homomorphisms, intersections with regular languages. They are closed under composition and under inverse. Monadic second-order transductions are closed under composition, but not under inverse.)

The family of HR equational sets of graphs is characterized as the set of images of binary trees under MS_2 transductions (those that transform graphs *via* their incidence graphs), hence is closed under these transductions ([27]). It follows also that MS_2 transductions *preserve bounded tree-width*. The family of VR equational sets is the set of images of binary trees under MS transductions, hence

is closed under these transductions ([34]) which consequently *preserve bounded clique-width*. These facts give characterizations independent of the chosen algebras. Furthermore, they help to prove that "small variations" on the signatures preserve the corresponding classes of equational sets ([12], [29]).

What do we get from definitions of sets of graphs by equation systems ?

We get relatively *compact descriptions*. By using *derived operations*, one can make them (hopefully) more readable. For example, series-composition of graphs with two sources, denoted by $\bullet$, is not a basic operation of the HR algebra, but it is defined by a term over the basic operations, and $\bullet$ can replace this term in an equation system. The equation defining series-parallel graphs is then $S = S//S \cup S \bullet S \cup e$ where $//$ and e that denote respectively *parallel-composition* and a single edge, are basic operations. Every generated graph has at least one derivation tree, which is a term over the operations of types HR and VR and their extensions with derived operations. This term can be used for *storing the graph as a string of symbols*, and as *input to algorithms*. Extensions of the *Semi-Linearity Theorem* for context-free languages (Parikh's theorem) make it possible to extract numerical informations, like the possible numbers of vertices and/or edges in a generated graph. Incorrect graphs can thus be detected by using this result as a preliminary test. *Filtering theorems* (see below) make it possible to transform equation systems. *Parsing* is more difficult for context-free graph grammars than for context-free word grammars. It is NP-complete for certain particular grammars and polynomial for others. (See the first two chapters of [10]). Unambiguous graph grammars would be interesting for counting purposes, like are unambiguous context-free grammars, (see the book by Flajolet and Sedgewick [5]). *Ambiguity* for a graph grammar is actually not that obvious to define, in particular because of associative and commutative operations like $//$ in the above definition of series-parallel graphs.

Recognizability

Two of the various equivalent characterizations of regular languages are interesting for dealing with graphs. First, their characterization in terms of *finite congruences*, because it applies to every algebra (Mezei and Wright in [46]) and second their characterization as the set of MS-definable languages. MS-definability is only meaningful for logical structures or for objects represented by such structures, and this is the case for graphs. In the case of languages, MS-definability is equivalent to recognizability, but for graphs one has only one implication : MS-definability implies recognizability. (It is open to find restrictions on the congruences used in the definition of HR- (or VR-) recognizability so as to make it equivalent to CMS_2- (or CMS-) definability.)

Theorem 2 (Recognizability Theorem): *Every CMS_2-definable set of graphs is recognizable in the HR algebra. Every CMS-definable set of graphs is recognizable in the VR algebra.*

The opposite implications cannot hold since there are uncountably many HR- and VR- recognizable sets of graphs. This uncountability result is linked to the infiniteness of the signatures of the HR and VR algebras. However a result stated in [40] says that a set of graphs of bounded tree-width is CMS_2-definable if and

only if it is HR-recognizable. No similar result is known for VR-recognizability. An important consequence of the *Recognizability Theorem* is the *Filtering Theorem.*

Theorem 3 (Filtering Theorem): *The intersection of an HR-equational set of graphs with a CMS$_2$-definable one is effectively HR-equational. The intersection of a VR-equational set with a CMS-definable one is effectively VR-equational.*

Direct constructions for particular properties like planarity (which is MS definable) would be particularly long and technical.

How can one prove the Recognizability Theorem ?

One proof can be sketched as follows : let Θ_k be the set of monadic second-order sentences (closed formulas) of quantifier height at most k over a fixed relational signature, and written in a certain normal form. This set has large cardinality but is finite. For a structure S we let $M_k(S)$ be the *level k theory of* S, *i.e.*, the set of sentences in Θ_k that are true in S. The main lemma states that $M_k(S \oplus T)$ can be computed from $M_k(S)$ and $M_k(T)$ by a function that depends only on k ($\oplus$ is disjoint union). A second (straightforward) lemma states that if t is a transformation of structures that can be expressed by *quantifier free (QF) formulas*, (this is the case of a relabelling, of the edge creation operation $add_{a,b}$ and of the edge complement to take typical examples) then $M_k(t(S))$ can be computed from $M_k(S)$ by a function that depends only on k and t. Since all VR and HR operations are expressible in terms of $\oplus$ and operations definable by QF formulas, the proof can be completed as follows: one defines a congruence on relational structures by $S \equiv T$ iff $M_k(S) = M_k(T)$. Technical details are omitted but everything is on the table. The lemma about $\oplus$ has generalizations involving combinations of infinite families of structures presented in [9]. The case of a mapping t such that $M_k(t(S))$ can be computed from $M_{f(k)}(S)$ for some fixed function f by a function that depends only on k and t, is also interesting. This is the case of unfolding ; such operations are discussed in [9].

Recognizability is a difficult notion to handle (the emptiness of an HR-recognizable set of graphs of unbounded tree-width is undecidable) but MS logic provides a handy language for specifying recognizable sets. In many cases the expression of a graph property by a monadic second-order formula is straightforward. Finite automata and regular expressions make it easy to specify recognizable languages, and monadic second-order formulas play a similar role for recognizable sets of graphs. They replace finite automata. No existing notion of graph automaton gives an equivalence with monadic second-order logic. If graph automata of some type would be effectively equivalent to monadic second-order formulas for all graphs, their emptiness problem would be undecidable because so is the corresponding problem for MS logic.

Back to equational sets.

The *Filtering Theorem* stated above is a direct application of the Recognizability Theorem and of the following fact : *in every algebra the intersection of an equational set and an effectively given recognizable set is effectively equational.* This generalizes the fact that the intersection of a context-free language and a regular one is context-free. Together with the decidability of emptiness, we get

that MS logic is decidable on VR-equational sets, and that CMS_2 logic is decidable on HR-equational sets. This means that one can test if every graph of the considered equational set satisfies the considered formula.

Could one prove or disprove mechanically (at least theoretically) some conjectures of graph theory by using this theorem ?

Only those of the form : every graph in a particular VR- or HR-equational set, or of clique-width or tree-width at most some given k, satisfies some property expressible in MS logic. But most graph theoretic conjectures (like *Hadwiger's Conjecture*) concern all graphs, and are not restricted to equational sets. Another difficulty with this idea is that graph properties involving comparisons of cardinalities are not MS expressible in general.

On which classes can one decide MS logic ?

There is a structural necessary (but not sufficient) condition.

Theorem 4 (Structural preresquisite for MS decidability): *(1) Every set of graphs having a decidable MS_2 theory has bounded tree-width, hence is a subset of some HR-equational set. (2) Every set of graphs having a decidable C_2MS theory has bounded clique-width, hence is a subset of some VR-equational set.*

The first statement is proved in [53], and the second one in [18]. The hypothesis that the C_2MS theory (C_2MS = MS with the even cardinality set predicate $Card_2(X)$) is decidable is stronger than requiring that the MS theory is decidable. These proofs use the result of [50] that excluding a planar graph as a minor implies bounded tree-width, and an extension of it to matroids. Hence in the case of sets of graphs, there are two obstacles to the decidability of CMS (or CMS_2) logic: unbounded clique-width (or tree-width), and, for sets of bounded clique-width or tree-width, the "internal complexity" of the considered sets. Sets of words can be complex enough so as to forbid the decidability of MS logic, although words are, considered as graphs, of tree-width 1.

By using the unfolding operation applied to directed acyclic graphs and MS transductions, one can construct classes of graphs that are not VR equational, that have decidable CMS-theories but (necessarily by Theorem 4) have bounded clique-width. That such graphs have bounded clique-width follows more directly from the fact that unfolding makes a graph into a tree. The use of unfolding increases the "internal complexity" of the described graphs, while keeping decidability of MS logic.

Are there fragments of MS logic that have decidable theories on classes of graphs of unbounded clique-width ?

An example is first-order logic, which is decidable on square grids (but not on all planar graphs). The satisfiability problem for existential monadic second-order logic (sentences of the form $\exists X_1, \ldots, X_n \varphi$ with φ first-order) is undecidable on square grids. However one might look for improvements of Theorem 4 where the hypotheses are the decidabilities of $\mathcal{L}$-theories for fragments $\mathcal{L}$ of MS logic.

Does the decidability of a fragment $\mathcal{L}$ of MS logic on a set of graphs $\mathcal{C}$ imply the existence of a polynomial time algorithm for each $\mathcal{L}$-expressible property over the graphs in $\mathcal{C}$?

For CMS logic and clique-width, and for CMS_2 logic and tree-width, the *Recognizability Theorem* implies simultaneously Theorem 1 and the decidabilities of CMS and CMS_2 logics over graphs of bounded clique-width and tree-width respectively, but, the only implication uses the detour through Theorem 4.

Couldn't we extend the languages CMS and CMS_2 while keeping decidability of the corresponding theories on graphs of bounded clique-width and tree-width ?

No such extension exists to my knowledge. The extension of CMS logic by an equal cardinality predicate $Eq(X, Y)$ expressing that sets X and Y have equal cardinality is undecidable on the set of words over a single letter. Another possibility could be with a cardinality oracle. That is we fix a recursive set of integers A and we let $Card_A(X)$ mean : X has cardinality in A. If A is the set of numbers that are either a power of 2 or a power of 3, then the corresponding extension of MS is *undecidable* on words. If A is the set of prime numbers the decidability is unknown. (This extension is stronger than the extension of the linear order of natural numbers with a predicate $P_A(x)$ expressing that $x \in A$ because bijections are not definable by monadic second-order formulas).

Can't one define all graphs with a finite set of operations ?

Yes, one can define all linearly ordered loop-free undirected graphs from four unary operations : addition of a new isolated vertex as new last vertex, addition of a new edge between the first two vertices, exchange of the first two vertices and circular shift (the first vertex becomes the last one, the second one becomes the first, etc...). As single constant, one can use the empty graph. With these operations and the one that forgets the ordering, one can define all graphs by terms, but these terms that are nothing but lists of vertices and edges. No interesting hierarchical structure is obtained like with the HR and VR operations. MS logic is undecidable on the corresponding equational sets. Although "small" and "powerful" this set of graph operations is uninteresting.

Relational structures.

The results of Theorems 1,2,3 have been stated for graphs, but their extensions to *relational structures over finite relational signatures* are straightforward, because most results are proved either at the Universal Algebra level or are valid for relational structures (cf. the proof sketch of the *Recognizability Theorem*). What are the relevant algebras ? One of them, generalizing the VR algebra uses disjoint union and QF operations. Another one can be defined that extends the HR algebra. More manageable sets of operations that generate the same equational sets and the same recognizable sets have been considered in [19], [31], [12]. In most cases, proofs in terms of relational structures are no more difficult than for classes of graphs and give more general statements.

A challenging open problem: *Is it true that if a set of relational structures has a decidable CMS theory, then it is the image of a set of binary trees under an MS transduction ?* If true, this would generalize Theorem 4 (2) but the tools used for its proof do not extend obviously. I consider this question as the main one in the area of relationships between MS logic and graph structure. The corresponding extension of Theorem 4 (1) is not difficult.

Here is a last result that indicates how nicely recognizability and MS logic fit together.

Theorem 5 ([12]): The inverse image of a recognizable set of relational structures under a CMS transduction is recognizable.

In this statement, recognizability is understood with respect the algebra based on disjoint union and QF operations. This result generalizes the fact that the inverse image of a CMS definable set of relational structures under a CMS transduction is CMS definable.

Acknowledgement. I thank A. Blumensath, M. Fellows and I. Walukiewicz for many useful comments on a first draft of this overview.

The first 11 references are books and survey articles.

References

1. Blumensath, A., Colcombet, T., Löding, C.: Logical Theories and Compatible Operations. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and automata: History and Perspectives, pp. 73–106. University Press, Amsterdam (2008)
2. Courcelle, B.: The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1, pp. 313–400. World Scientific, Singapore (1997)
3. Courcelle, B.: Graph Structure and Monadic Second-order Logic. Cambridge University Press, Cambridge (in preparation), http://www.labri.fr/perso/courcell
4. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
5. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press, Cambridge (to appear)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
7. Grohe, M.: Logic, Graphs, and Algorithms. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and automata: History and Perspectives, pp. 357–422. Amsterdam University Press (2008)
8. Hell, P., Nešetřil, J.: Graphs and homomorphisms. Oxford University Press, Oxford (2004)
9. Makowsky, J.: Algorithmic uses of the Feferman-Vaught Theorem. Ann. Pure Appl. Logic 126, 159–213 (2004)
10. Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1. World Scientific, Singapore (1997)
11. Tutte, W.: Graph Theory. Addison–Wesley, Reading (1984)
12. Blumensath, A., Courcelle, B.: Recognizability, Hypergraph Operations, and Logical Types. Inf.Comput. 204, 853–919 (2006)
13. Bodlaender, H.: A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
14. Bodlaender, H.: Treewidth: Characterizations, Applications, and Computations. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 1–14. Springer, Heidelberg (2006)

15. Brandstädt, A., Dragan, F., Le, H., Mosca, R.: New Graph Classes of Bounded Clique-Width. Theory Comput. Syst. 38, 623–645 (2005)
16. Brandstädt, A., Engelfriet, J., Le, H., Lozin, V.: Clique-Width for 4-Vertex Forbidden Subgraphs. Theory Comput. Syst. 39, 561–590 (2006)
17. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: Progress on Perfect Graphs. Mathematical programming, Ser. B 97, 405–422 (2003)
18. Courcelle, B., Oum, S.: Vertex-minors, Monadic Second-Order Logic, and a Conjecture by Seese. J. Comb. Theory, Ser. B 97, 91–126 (2007)
19. Courcelle, B.: The Monadic Second-Order Logic of Graphs VII: Graphs as Relational Structures. Theor. Comput. Sci. 101, 3–33 (1992)
20. Courcelle, B.: The Monadic Second-Order Logic of Graphs X: Linear Orderings. Theor. Comput. Sci. 160, 87–143 (1996)
21. Courcelle, B.: The Monadic Second-Order Logic of Graphs XI: Hierarchical Decompositions of Connected Graphs. Theor. Comput. Sci. 224, 35–58 (1999)
22. Courcelle, B.: The Monadic Second-Order Logic of Graphs XII: Planar Graphs and Planar Maps. Theor. Comput. Sci. 237, 1–32 (2000)
23. Courcelle, B.: The Monadic Second-Order Logic of Graphs XIV: Uniformly Sparse Graphs and Edge Set Quantifications. Theor. Comput. Sci. 299, 1–36 (2003)
24. Courcelle, B.: The Monadic Second-Order Logic of Graphs XV: On a conjecture by D. Seese. J. Applied Logic 4, 79–114 (2006)
25. Courcelle, B.: The Monadic Second-Order Logic of Graphs XVI: Canonical graph decompositions. Logical Methods in Computer Science 2 (2006)
26. Courcelle, B.: Circle Graphs and Monadic Second-order logic. Journal of Applied Logic (in press)
27. Courcelle, B., Engelfriet, J.: A Logical Characterization of the Sets of Hypergraphs Defined by Hyperedge Replacement Grammars. Mathematical Systems Theory 28, 515–552 (1995)
28. Courcelle, B., Engelfriet, J., Rozenberg, G.: Handle-Rewriting Hypergraph Grammars. J. Comput. Syst. Sci. 46, 218–270 (1993)
29. Courcelle, B., Makowsky, J.: Fusion in Relational Structures and the Verification of Monadic Second-Order Properties. Mathematical Structures in Computer Science 12, 203–235 (2002)
30. Courcelle, B., Makowsky, J., Rotics, U.: Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. Theory Comput. Syst. 33, 125–150 (2000)
31. Courcelle, B., Weil, P.: The Recognizability of Sets of Graphs is a Robust Property. Theor. Comput. Sci. 342, 173–228 (2005)
32. Courcelle, B., Walukiewicz, I.: Monadic Second-Order Logic, Graph Coverings and Unfoldings of Transition Systems. Ann. Pure Appl. Logic 92, 35–62 (1998)
33. Cunnigham, W.: Decomposition of Directed Graphs. SIAM Algor. Discrete Meth. 3, 214–228 (1982)
34. Engelfriet, J., van Oostrom, V.: Logical Description of Contex-Free Graph Languages. J. Comput. Syst. Sci. 55, 489–503 (1997)
35. Fellows, M., Rosamond, F., Rotics, U., Szeider, S.: Clique-width Minimization is NP-hard. In: 38th Annual ACM Symposium on Theory of Computing, pp. 354–362 (2006)
36. Frick, M.: Generalized Model-Checking over Locally Tree-Decomposable Classes. Theor. Comput. Sci. 37, 157–191 (2004)
37. Frick, M., Grohe, M.: The Complexity of First-order and Monadic second-order Logic Revisited. Ann. Pure Appl. Logic 130, 3–31 (2004)

38. Gallai, T.: Transitiv Orientierbare Graphen. Acta Math. Acad. Sci. Hungar 18, 25–66 (1967); Translation in English by Maffray, F. Preissmann, M.: In: Ramirez Alfonsin,J.L., Reed, B.A.: (eds.), Perfect Graphs, pp. 25-66, Wiley, New York (2001)
39. Grädel, E., Hirsch, C., Otto, M.: Back and Forth Between Guarded and Modal Logics. ACM Trans. Comput. Log. 3, 418–463 (2002)
40. Lapoire, D.: Recognizability Equals Monadic Second-Order Definability for Sets of Graphs of Bounded Tree-Width. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 618–628. Springer, Heidelberg (1998)
41. Hlinený, P., Oum, S.: Finding Branch-Decompositions and Rank-Decompositions. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 163–174. Springer, Heidelberg (2007)
42. Klarlund, N.: Mona & Fido: The Logic-Automaton Connection in Practice. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414, pp. 311–326. Springer, Heidelberg (1998)
43. Madelaine, F.: Universal Structures and the Logic of Forbidden Patterns. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 471–485. Springer, Heidelberg (2006)
44. Makowsky, J., Marino, J.: Tree-width and the Monadic Quantifier Hierarchy. Theor. Comput. Sci. 303, 157–170 (2003)
45. Makowsky, J., Pnueli, Y.: Arity and Alternation in Second-Order Logic. Ann. Pure Appl. Logic 78, 189–202 (1996); Erratum: Ann. Pure Appl. Logic 92, 215 (1998)
46. Mezei, J., Wright, J.: Algebraic Automata and Context-Free Sets. Information and Control 11, 3–29 (1967)
47. Nešetřil, J., de Mendez, P.O.: Linear Time Low Tree-width Partitions and Algorithmic Consequences. In: Proc. Symp. Theory of Computation, pp. 391–400 (2006)
48. Oum, S.: Rank-width and Vertex-minors. J. Comb. Theory, Ser. B 95, 79–100 (2005)
49. Oum, S., Seymour, P.: Approximating Clique-width and Branch-width. J. Comb. Theory, Ser. B 96, 514–528 (2006)
50. Robertson, N., Seymour, P.: Graph Minors. V. Excluding a Planar Graph. J. Comb. Theory, Ser. B 41, 92–114 (1986)
51. Robertson, N., Seymour, P.: Graph minors. VIII. A Kuratowski Theorem for General Surfaces. J. Comb. Theory, Ser. B 48, 255–288 (1990)
52. Robertson, N., Seymour, P.: Graph Minors. XVI. Excluding a Non-planar Graph. J. Comb. Theory, Ser. B 89, 43–76 (2003)
53. Seese, D.: The Structure of Models of Decidable Monadic Theories of Graphs. Ann. Pure Appl. Logic 53, 169–195 (1991)
54. Soguet, D.: Génération Automatique d'Algorithmes Linéaires, Doctoral dissertation, Paris-Sud University, France (July 2008)
55. Wanke, E.: k-NLC Graphs and Polynomial Algorithms. Discrete Applied Mathematics 54, 251–266 (1994)

Internet Ad Auctions: Insights and Directions

S. Muthukrishnan

Google Inc., 76 9th Av, 4th Fl., New York, NY, 10011
muthu@google.com

Abstract. On the Internet, there are advertisements (ads) of different kinds: image, text, video and other specially marked objects that are distinct from the underlying content of the page. There is an industry behind the management of such ads, and they face a number of algorithmic challenges. This note will present a small selection of such problems, some insights and open research directions.

1 Introduction

Everyday we interact with the Internet in several ways. For example, we read news from an online source, go to a portal to check email or start at a search engine to navigate the web or for discovery. We belong to some explicit social network and interact with "friends" online. We plan and execute projects or events. Through these activities, we express our interests, intent, an implicit openness to discover new things, not only individually, but also as members of different groups. These are the *signals* we generate on the Internet.

This Internet world is valuable to businesses. They seek to benefit from this online world not only by transacting their business but also by marketing themselves. Marketing is predominantly done via advertisements (ads), using the many signals. The ads could be in different forms from text snippets to images and even videos. These ads provide some information and seek to get the attention of the users, and ultimately induce some action, direct or indirect.

There is now an industry of companies that enable the process above. This includes companies that bring users to their site, procure ads and manage the ad presentation and billing processes, as well as companies, that are mediators, merely placing ads where they are slotted and needed in others' properties. For most part, the ad placement is determined by *auction*s, so there is focus on the strategizing and gaming aspects of the behavior of advertisers and ad placement companies. In addition, there are companies that enable advertisers as well as content providers to strategize and optimize their goals across multiple parties.

All of this generates many algorithmic challenges. In this paper, we present an overview Internet ad auctions typically used. We will present some insights and directions for research. In many cases, the directions are only abstractions that are aimed at spurring some theoretical research thought and are not direct business problems. We believe that Internet ad auctions is an area of research where novel ideas will have great impact in practice because large scale Internet ad systems exist, are successful, and can be modified with reasonable effort. This

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 14–23, 2008.

is an evolving area, and this write-up covers a small vantage point only; we will maintain an updated version of the write-up over time.

2 Basics

In this section, we describe the type of ads, the process of ad campaign development and an overview of the auction mechanism that is involved.

2.1 Types of Ads

There are different types of Internet ads, depending on the nature of signals used as well as the nature of ads. We give a few examples.

Sponsored search ads. When a user poses a query at a search engine, the search engine returns search results together with advertisements that are placed into positions, usually arranged linearly down the page, top to bottom. On most major search engines, the assignment of ads to positions is determined by an auction among all advertisers who placed a bid on a keyword that matches the query. The user might click on one or more of the ads, in which case (in the pay-per-click model) the advertiser receiving the click pays the search engine a price determined by the auction. This is known as sponsored search ads.

Content ads. Users go to several sites for their intrinsic content. For example, this includes content providers such as established news sources or more individualistic blog sites and other publishing sites. Hence, content at such sites matches users' interests or intent. Content providers use these signals derived from their content to target ads to users. This is known as content ads.

Display ads. Users go to portals and other pivotal sites as starting points of their interaction with the Internet. Advertisers seek to get such users' attention by display of ads which may not be directly determined by the content of such sites. Banners and pop-ups are examples of such display ads.

Social networking ads. Users belong to one or more social networks, and interact with friends and contacts. The signal of such friends, friends' signals and so on, may be used to present ads to a user. Such ads are social networking ads. ■

2.2 Life of an Ad

Planning and execution of an ad campaign involves at least three mains stages.

- *Targeting.* Advertisers determine the potential target for ad campaigns. Targeting may take the form of demo or psycho-graphics and rely on users' signals.
- *Ad placement and optimization.* Advertisers strategize with budgets and prices for favored placement of their ads and optimize the overall impact of their budgets.

– *Ad effectiveness.* Any ad campaign needs to evaluate its over all impact. In offline ad media, effectiveness of an ad may have to be measured via surveys or coupons etc. In Internet ads, there are other measures of ad effectiveness including click-through as well as change in traffic levels due to an ad.

Life of an ad starts with its targeting, proceeds to production, optimization and execution of the ad campaign, and finally, evaluation of the effectiveness of the campaign. Algorithmic problems arise in each of these stages.

2.3 Ordered Ad Auctions

We describe the popular Internet ad auctions formally, and call them *ordered ad auctions*. An ordered ad auction is defined by a tuple (N, K, v, α, β). The set $N = \{1, \ldots n\}$ is the set of bidders (advertisers) and the set $K = \{1, \ldots, n\}$ is the set of positions, ordered top to bottom. Each bidder $i \in N$ is associated with two values, v_i which is her valuation for a click and α_i which is her *click-through rate* (ctr). Each position $\ell \in K$ is associated with a click-through multiplier β_ℓ. We have $\beta_1 = 1$ and $\beta_\ell > \beta_{\ell+1}$ thereafter, ie., the position multiplier goes down top to bottom. The standard assumption is that the actual α of bidder i in position j is *separable* [7], that is, it is the product of the bidder's ctr α_i and the position multiplier β_j: if bidder i is placed at position j then she receives a click with probability $\alpha_{i,j} = \alpha_i\beta_j$. The value of v_i is known only to bidder i while all the other parameters are publicly known.

The ad placement is determined by an *auction*. Advertiser i specifies a bid b_i which is the maximum they wish to pay. The rules of the auction determine the *ordering* of the k chosen ads in these positions, as well as their *pricing*. We describe two well known auctions.

GSP. The most natural ordering is to sort by decreasing bid, but that does not take into account the quality of ads and their suitability to users. The Generalized Second Price (GSP) mechanism ranks the bidders by $b_i\alpha_i$. Wlog assume that bidder i is assigned to position i. The price that the bidder at position i pays per click is

$$P_j^{GSP} = \frac{b_{i+1}\alpha_{i+1}}{\alpha_i}.$$

This is the ordering and pricing currently in use by search engines like Yahoo! and Google. ■

VCG. There is implementation of the well known Vickrey-Clarke-Groves (VCG) mechanism [6,1,3]. This mechanism ranks the bidders by $b_i\alpha_i$, which can be thought of as the expected advertiser value if $b_i = v_i$. Wlog assume that bidder i is assigned to position i. The VCG allocation maximizes the *social welfare*, which is the sum of the bidders' expected value, i.e., $\sum_{i \in N} v_i\alpha_i\beta_i$. The VCG mechanism charges each bidder the total value lost to other bidders caused by her presence in the auction. The VCG mechanism has the property that the

bidders' dominant strategy is to bid their true value, i.e., $b_i = v_i$.[1] Formally, the VCG price for position j is [5,4]

$$P_j^{VCG} = \sum_{i>j} \frac{b_i \alpha_i (\beta_{i-1} - \beta_i)}{\alpha_j \beta_j}.$$

Note that $\beta_j = \sum_{i>j}(\beta_{i-1} - \beta_i)$ and therefore $P_j^{VCG} \leq b_{j+1}\alpha_{j+1}/\alpha_j$. ■

Ordered ad auctions described above occur in sponsored search where the keywords that match the search query entered by the user determines the advertisers in the auction. Such ordered ads are also used in content ads where keywords that suitably match the content on a web page determine the advertisers. More generally, various properties of the users as well as the content can jointly determine the pool of advertisers in the auction.

As described above, advertisers are charged only for clicks. In alternative models, advertisers may pay for just appearing (impressions) or only if they *acquire* users according to a suitable definition of "acquire" (users buy products or spend time at the advertiser's website familiarizing themselves with the products, etc).

There is an overview of sponsored search auctions in [20], and a nice introduction to the area of mechanism design [21] which applies to Internet ad auctions. Several workshops, conferences and meetings address Internet ad auction problems. Recent plenary talk [22] discusses some of the algorithmic challenges in content ads.

3 Some Directions

This paper does not describe the issues involved in any detail. Instead, we prefer to present certain directions for research thought. Most of these problems have to be formalized more precisely and depending on the creative approaches taken, will lead to many different research problems.

3.1 Game Theory of GSP in Practice

Since GSP is the most widely used mechanism in practice for ordered ad auctions, it is worthwhile to understand its strengths. This is typically done using game theory.

A commonly accepted *utility* of bidder i at position j is:

$$u_i(j) = \alpha_i \beta_j (v_i - p_j),$$

where p_j is the price per click and is a function of all the bids. For advertisers with this utility, GSP mechanism is not truthful [7], ie., they can gain some by not revealing their true values as bids. In contrast, VCG mechanism is truthful. Still,

[1] A *dominant strategy* is a strategy that a bidder always prefers regardless of the other players' strategies. A mechanism is said to be *truthful* if revealing the true valuation is a dominant strategy for every bidder.

recent papers [4,6,7] show that there exists an *equilibrium* of GSP that is identical in prices and positions to the truthful VCG equilibrium. This characterization gives a nice way to understand the properties of GSP.

Problem 1. In practice, the GSP mechanism is implemented with certain tweaks, for example, each advertiser has an advertiser-specific *minimum price, or certain positions have a set reserve price, or more generally, each advertiser and position pair may have a minimum price. What are natural variations of GSP with these tweaks and what are the equilibrium properties of the resulting auctions?*

Recently, the tweak of just adding advertiser-specific minimum price was studied, and the authors show that GSP has a nice equilibrium even with this tweak [14]. The proof structure in [4,6,7] does not work and a significantly new proof approach was developed in [14]. In presence of the other tweaks stated in the problem above, even modifying GSP suitably may present challenges.

3.2 Multiparty Modeling

Let us examine the commodities being sold: these are clicks at various positions. To determine the best allocation, we need to understand the value of these commodities. The probability of a click on an ad depends on the ad being shown, the user seeing the ad provided it is shown, and then conditioned on that, the user clicking the ad. Of these, the last two are user-dependent. Therefore, the game we study has in fact three parties: users, advertisers and ad providers. So far, we assumed that the probability of user clicking on ad i at position j is $\alpha_{i,j} = \alpha_i \beta_j$, *independent* of other ads. In general, this is unrealistic [2] and it exogenizes the role of the user. A more principled approach would be to assume that the user is not strategic, but model the behavior of the user, and conditioned on this model, study the game between the advertisers and the ad provider.

More precisely, consider the following *markov model* for user behavior.

Problem 2. User scans the positions from the top. Consider position i with ad j. User chooses to click on the ad with probability dependent on the ad j, say $p(j)$, and chooses to scan down the list with probability $q(i, j)$ dependent on the ad seen as well as current position. Assume this markov user model. Determine an allocation of ads to slot with maximum total expected value.

One such an allocation is found, we can use pricing as determined by the VCG mechanism to get a truthful auction. See [9] for a development of this approach, for a restricted model when $q(i, j)$ is only a function of ad j. The rationale behind the model is that the suitability of the ad to their task as well as the fatigue of exploring many of them determines the likelihood of an user continuing to scan the ad list.

A more generally scenario emerges in content ads as described below.

Problem 3. We have the content provider *(say a blog writer) who has some flexibility in choosing the story they wish to tell as well as the words to describe their story (for example, draw analogies, use quotations, etc). The content provider*

will (a) choose the story that is likely to generate large number of readers*, (b) choose the words which are likely to be used by the* ad provider *to target ads (c) indirectly influence the choice of quality ads shown to the readers, and thereby (c) generate most revenue for themselves from the* advertisers*. Thus, the content provider is a strategic player. Formalize and study mechanism design in this world of four players, ie., content providers, readers, advertisers, and ad providers.*

3.3 Optimal Mechanism

Consider the problem of designing an "optimal" mechanism for ordered ad auctions. We will consider the Bayesian case.

Problem 4. Say each of the n bidders has their value in $[0, 1]$ *drawn independently randomly from a distribution* F*. There are* k *ordered positions with decreasing position-dependent click-through rates and assume separability of ctr's. What is the optimal mechanism, that is, a mechanism that maximizes the expected profit?*

If $k = 1$, the optimal Bayesian mechanism was shown by Meyerson [15], and it is a VCG auction that involves setting a reserve price for the position, and selling only if the instance of the values has one that exceeds the reserve price. In the problem above, we are interested in extending it to k position as suitable for ordered ad auctions. See [16] for a discussion of Bayesian and worst case optimal mechanism design.

3.4 Target Size Estimation

Early in the life of an ad is the step of targeting ads. In order to do that, often, advertisers need to estimate the size of their target group, say based on demographics. This is computed typically from data accumulated via surveys and other means which are expensive and do not have complete coverage. To address this, we abstract the following.

Problem 5. We have d attributes which are say hierarchical[2]*. We are given several* $d + 1$ *tuples for preprocessing, where the first d tuples are the attributes and the* $d + 1$*th attribute is the size. Each query specifies a d tuple of attributes and the output is an estimate of its size based on some model of data and size distribution. Devise a theoretical way to address the estimate quality vs complexity of estimation in this problem.*

A natural model is to assume that the sizes are uniformly distributed within the region specified by the d attributes in each tuple. There are heuristic ways to project this model to get size estimate for query region, but a theoretically sound approach that quantifies the information complexity vs accuracy tradeoff will have impact.

[2] For example, *time* is hierarchical; it is represented as a tree with root being the year, its children being the months, their children being partial weeks, down to days, hours and so on. The attribute *salary* may not be hierarchical, since $[50k, 100k]$ and $[75k, 150k]$ may be valid ranges.

3.5 Mechanisms for Heterogeneous Ads

Not all ads are equal. Here is a basic problem.

Problem 6. We have ads of Types I and T. We can either put one ad of Type I or two ordered ads of type T in the space provided. Design a GSP like mechanism for running this auction. In particular, can you formalize and satisfy the property that an advertiser pays the smallest amount needed to obtain their allotment?

3.6 Mechanisms for Heterogeneous Utilities

There has been a lot of research that assumes some silo or the other of advertisers. In practice, we get a mixture of advertisers. Some care about impressions, others care about clicks. Some care about profit maximization and others about appearing in some position, no matter the cost. Classical game theory provides insights into mechanisms for specific classes of utility functions, but in practice, one gets a mixture of utility functions.

Problem 7. Consider a collection of advertisers with mixed utilities in terms of impressions/clicks/acquisition, profit/position or other suitable parameters. Design a truthful, intuitive mechanism for such a collection and characterize the equilibrium behavior and advertiser dynamics.

The authors in [18], propose a general model and design a stable matching based mechanism which is truthful for a rich set of advertisers. They also propose an efficient algorithm to implement the mechanism. Alternative models and mechanisms, with richer mix of advertisers, will be of great interest.

3.7 Ad Effectiveness Estimation

An important task is to estimate the effectiveness of ads. While clicks are a natural measure, in some cases such as in branding or display or offline media ads, we need indirect ways to measure effectiveness, say in terms of increased traffic at sites of advertisers, or call backs or innovative methods such as 2d barcodes [23]. Motivated by this, we abstract the following.

Problem 8. We are given two disjoint sets of target groups C and A. Target groups in A are shown ads during time period $[0, T]$ and target groups in C are not shown ads. We have time series measurement of activity that pertains to the ad of each target group in C and A (eg, the measurement is the number of visits to the advertiser's website for each time instant) measured over time $(\ldots, 0, \ldots, T, \ldots)$. The task is to design a concise measure of the change in measurements in A after the ad was shown wrt prior time when the ad was not shown and also wrt other target groups in C during the entire time (since they were not shown ads). Assume the richest model of dependence among the different timeseries of measurements that makes the problem computationally feasible.

3.8 Inferring Profiles: Graph Learning

Many inference problems for ad targeting and elsewhere can be abstracted as follows. There are entities with associated metadata and links. Some of the entities are labeled, and we need to infer labels for the others. We can infer the label for an entity using its metadata. But, instead, we wish to infer the label based on just the link structure between the entities and the known labeling. For example, say each entity is a blog profile, and the label of our interest is the *age* of the blogger. Some of the blog profiles have age value entered (truthful or not) and others have missing age value. It will be beneficial to infer some age value for the unlabeled bloggers. This will be useful for targeting ads, for example, if the advertiser wants their ad shown only to those above a certain age. The problem we state below models this (and other cases with labels like say gender or interest in rock music, etc.) informally.

Problem 9. The input is a graph $G = (V, E)$ with some nodes $v \in V$ with a label $L(v)$. Output is $L(u)$ for all $u \in V$.

See [12] for a random walk approach and see [10] for the Machine Learning approach to this problem. More remains to be done, in particular, in novel approaches and re-formulations of this problem. We propose the following.

Problem 10. Initially we are given a graph H with a labeling $L : V(H) \to [k]$. Then, for each edge e of H, and independently, we remove e with probability $p_{i,j}$, where i and j are the labels of the end-points of e. Now, we present the resulting graph G, sans the labels, to the algorithm. Design a decoding algorithm which will produce a maximum likelihood labeling of the nodes of G, with high probability.

If H were the complete graph on n nodes, results in [11] apply and such decoding is possible. For general H, preliminary results on this problem are in [19]. Results for special cases of H or improved approximate decoding results are of great interest; also, the labeling could be a probability distribution over the label set, and algorithms for such cases are of interest.

3.9 Reservation Auction

We have so far considered spot auctions only, that is, the auction takes place at the instant when the commodity is available. In certain cases, typically with display ads, advertisers need to be able *reserve* ad spots ahead of time. Motivated by this, we abstract the following problem.

Problem 11. There are ad positions available at each time instant. Bids arrive online; each bid is for a subset of ad positions at some specific instant in the future. Bids must be accepted or rejected when they arrive. Assume reservations once accepted may be rejected later, for a bump fee. Design a mechanism for making these decisions online and still result in an allocation which is comparable to offline mechanisms in revenue and welfare, together with desirable game-theoretic properties.

For the case when each bidder requires a single slot only, some positive results appear in [26].

3.10 Budget Optimization and Bidding

Sometimes it is required to view ordered ad auctions from the viewpoint of advertisers. They need tools to strategize and develop a bidding method with multiple keywords and targeting groups.

Problem 12. Consider a particular advertiser. We are given budget U, a keyword-query graph G and distribution of how many clicks we get at what cost for each bid value on each query in the GSP auction, assuming all other parameters remains fixed. The goal is to output a bid for each keyword so that the expected number of clicks obtained by the advertiser is maximized, subject to the condition that budget is not exceeded. Also, characterize the equilibrium if all advertisers follow the click-maximizing strategy in GSP auction.

A bidding strategy (bidding uniformly on all keywords) and its variants are explored in [24]. Certain basic dynamics have been understood [25], but more remains to be done.

4 Concluding Remarks

This writeup provide some insight into Internet ad auctions and directions for further research. Internet ad auctions face algorithmic challenges, and will remain a rich area of research for some time. It is also likely to be useful research area because large scale auction systems exist and are successful; new research ideas can modify them or rework them completely with reasonable effort, and have the potential for great impact.

Acknowledgements

I sincerely thank my colleagues at Google.

References

1. Clarke, E.: Multipart pricing of public goods. Public Choice 11, 17–33 (1971)
2. Craswell, N., Zoeter, O., Taylor, M., Ramsey, B.: An experimental comparison of click position-bias models. In: WSDM 2008: Proceedings of the international conference on Web search and web data mining, pp. 87–94. ACM Press, New York (2008)
3. Groves, T.: Incentives in teams. Econometrica 41(4), 617–631 (1973)
4. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. In: Second workshop on sponsored search auctions (2006)

5. Varian, H.: Position auctions. International Journal of Industrial Organization 25(6), 1163–1178 (2007)
6. Vickrey, W.: Counterspeculation, auctions and competitive-sealed tenders. Finance 16(1), 8–37 (1961)
7. Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: ACM Conference on Electronic Commerce (EC) (2006)
8. Feldman, J., Muthukrishnan, S.: Algorithmic methods for sponsored search. In: Liu, Z., Xia, C. (eds.) Performance Evaluation and Modelling. Springer, Heidelberg (to appear, 2008)
9. Aggarwal, G., Feldman, J., Muthukrishnan, S., Pál, M.: Sponsored search auctions with Markovian users (submitted, 2008)
10. Graph Labelling Workshop - ECML/PKDD (2008), http://graphlab.lip6.fr/pmwiki.php
11. McSherry, F.: Spectral partitioning of random graphs. In: Proceedings of the 42th Annual IEEE Symposium on Foundations of Computer Science (2001)
12. Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., Aly, M.: Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph WWW (2008)
13. Aggarwal, G., Ailon, N., Constantin, F., Even-Dar, E., Feldman, J., Frahling, G., Henzinger, M., Muthukrishnan, S., Nisan, N., Pal, M., Sandler, M., Siridopoulos, A.: Theory research at Google. In: SIGACT News (to appear, 2008)
14. Even-Dar, E., Feldman, J., Mansour, Y., Muthukrishnan, S.: Sponsored search with bidder-specific minimum prices. In: [13] (2008)
15. Meyerson, R.: Optimal auction design. Mathematics of Operations Research 6, 58–73 (1981)
16. Hartline, J.: Lectures on optimal mechanism design. In: [17], http://www.ece.northwestern.edu/~hartline/omd.pdf
17. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V. (eds.): Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
18. Aggarwal, G., Muthukrishnan, S., Pal, D., Pal, M.: General Auction Mechanism for Search Advertising. (manuscript, 2007), http://www.cs.uwaterloo.ca/ dpal/papers/auction/ stable-matching-auctions.pdf
19. Feldman, J., Muthukrishnan, S., Sidiropoulos, A.: Graph labeling in the planted graph model. (manuscript, 2008) See [13]
20. Lahaie, S., Pennock, D., Saberi, A., Vohra Sponsored, R.: search auctions. In: [17] (2007)
21. Nisan, N.: Introduction to Mechanism Design (for Computer Scientists). In: [17] (2007)
22. Broder, A.: Computational advertising. In: Proc. ACM-SIAM SODA, p. 992 (2008)
23. Print ads 2D barcodes, http://www.google.com/adwords/printads/ads/barcode/
24. Feldman, J., Muthukrishnan, S., Pal, M., Stein, C.: Budget optimization in search-based advertising auctions. In: Proc. ACM EC (2007)
25. Borgs, C., Chayes, J., Immorlica, N., Jain, K., Etesami, O., Mahdian, M.: Dynamics of bid optimization in online advertisement auctions. In: Proc. ACM EC (2007)
26. Constantin, F., Feldman, J., Muthukrishnan, S., Pal, M.: Online Ad Slotting With Cancellations. (manuscript, 2008) See [13]

The Complexity of Boolean Formula Minimization

David Buchfuhrer[1,⋆] and Christopher Umans[2,⋆⋆]

[1] Computer Science Department
California Institute of Technology
Pasadena, CA 91125
dave@cs.caltech.edu
[2] Computer Science Department
California Institute of Technology
Pasadena, CA 91125
umans@cs.caltech.edu

Abstract. The Minimum Equivalent Expression problem is a natural optimization problem in the second level of the Polynomial-Time Hierarchy. It has long been conjectured to be Σ_2^P-complete and indeed appears as an open problem in Garey and Johnson [GJ79]. The depth-2 variant was only shown to be Σ_2^P-complete in 1998 [Uma98], and even resolving the complexity of the depth-3 version has been mentioned as a challenging open problem. We prove that the depth-k version is Σ_2^P-complete under Turing reductions for all $k \geq 3$. We also settle the complexity of the original, unbounded depth Minimum Equivalent Expression problem, by showing that it too is Σ_2^P-complete under Turing reductions.

1 Introduction

Circuit minimization problems are natural optimization problems that lie in the second level of the Polynomial-Time Hierarchy (PH). The general form of such a problem is: given a Boolean circuit, find the smallest Boolean circuit that computes the same function. The input and output circuit may be required to be circuits of a particular form, e.g., Boolean formulas, or bounded-depth circuits. These problems are central problems in the field of logic synthesis, where fairly large instances are routinely solved using heuristics [DGK94]. They are also the prime examples of natural problems that should be complete for the classes of the second level of the PH. Indeed, versions of these problems inspired the definition of the PH in the early 70s by Meyer and Stockmeyer [MS72, Sto76], and Garey and Johnson use the formula variant to motivate the definition of the second level of the PH [GJ79].

Completeness proofs for circuit minimization problems have been hard to find. The DNF version of circuit minimization was only proven to be Σ_2^P complete

⋆ Supported by NSF CCF-0346991 and BSF 2004329.
⋆⋆ Supported by NSF CCF-0346991, BSF 2004329, a Sloan Research Fellowship, and an Okawa Foundation research grant.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 24–35, 2008.

in 1998 by Umans [Uma98]; the other variants have remained prominent open problems. The only non-trivial hardness result for the formula variant – called Minimum Equivalent Expression – is a $P_{||}^{NP}$-hardness result of Hemaspaandra and Wechsung in 1997 [HW97]. One reason reductions for these problems are difficult is that one direction of the reduction entails proving a lower bound for the type of circuit under consideration. This shouldn't be a absolute barrier, though, for two reasons. First, we have lower-bound proof techniques for Boolean formulas and bounded-depth circuits; nevertheless incorporating these into a reduction seems tricky. Second, a reduction need not entail *strong* lower bounds and in principle even slightly non-trivial lower bounds could suffice. A similar difficulty for potential reductions showing the (conjectured) NP-hardness of a related problem was noted by Cai and Kabanets [KC00], although there, the use of weak lower bounds is not even an option, under a complexity assumption.

Proving Σ_2^P-completeness of the depth-3 variant was proposed [UVSV06] as a challenging first step, one that might begin to utilize techniques for proving lower bounds for bounded depth circuits (e.g., the Switching Lemma). In this paper we resolve, in one shot, the depth-3 case, as well as the depth-k variants for all $k \geq 3$. The same techniques show in addition that the unbounded depth Minimum Equivalent Expression problem is Σ_2^P-complete under Turing reductions. We are able to achieve our results by exploiting the second way around the apparent barrier of proving circuit lower bounds: our reductions entail circuit lower bounds, but we get by with very weak ones, that with some effort are incorporated naturally into the structure of the reduction.

1.1 Description of the Reduction

In this section we give a high-level description of the reduction, emphasizing a few interesting features before delving into the technical details.

The problem we reduce from is SUCCINCT SET COVER, which was defined and shown to be Σ_2^P-complete in [Uma99b]:

*Problem 1.1 (*SUCCINCT SET COVER (SSC)). Given a DNF formula D over variables $v_1, \ldots, v_m, x_1, \ldots, x_n$ and an integer k, is there a subset $I \subseteq \{1, 2, \ldots n\}$ with $|I| \leq k$ and $D \vee \bigvee_{i \in I} \overline{x_i} \equiv 1$?

This can be seen as a succinct version of Set Cover, in which the sets are implicitly specified by the ones of the formulas D, $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$, and D is mandatory in any set cover (e.g., because it covers some point not covered by any of the other sets).

Throughout this paper, we assume that the formula D accepts the all-true assignment, as it only requires polynomial time to check this and the SSC instance is trivially false otherwise.

Our reductions exploit the special structure of this "succinct" set cover instance. In particular, all of the sets other than the one implicitly specified by D have an extremely simple form (they are just halfspaces), and in our reduction to MINIMUM EQUIVALENT EXPRESSION the choice of whether they are included or excluded from a cover will manifest itself relatively easily in the size of minimum

equivalent expression. However, D may be a complicated function, one whose minimum formula size is not readily apparent. To circumvent this problem, we will use a Turing Reduction (actually a non-adaptive, or *truth-table*, reduction) which first ascertains the minimum formula size of D, and then asks one further query on a formula that incorporates D and other components, to determine whether or not the original instance of SUCCINCT SET COVER is a positive instance. This provides a somewhat rare example of a natural problem for which a Turing reduction seems crucial (in the sense that we do not know of any simple modification or alternative methods that would give a many-one reduction).

More specifically, the main idea of our reduction is to consider the following formula, derived from an instance of SUCCINCT SET COVER:

$$D \vee [z \wedge (\overline{x_1} \vee \cdots \vee \overline{x_n})] \tag{1}$$

where z is a new variable. Notice that when z is *false*, this formula is equivalent to just D, which (intuitively) forces a minimum equivalent formula to devote part of its size to computing an "unpolluted" copy of D. When z is *true*, the formula covers exactly the union of all of the "sets" in the instance of SUCCINCT SET COVER. That problem asks whether the disjunction of k or fewer $\overline{x_i}$ literals suffice to cover everything not covered by D. If the variables indexed by $I \subseteq \{1, 2, \ldots, n\}$ suffice, then a very economical equivalent formula to the one above is $D \vee \left[z \wedge \left(\bigvee_{i \in I} \overline{x_i}\right)\right]$. By forcing a minimum equivalent formula to already compute a copy of D, we can ensure that the most efficient way to compute the above equivalent formula is indeed of this intended form. We can then determine whether or not there is a cover of size k by asking whether (1) has an equivalent formula of size k plus whatever we determined the minimum equivalent formula size of D to be.

To make this actually work requires some modifications. For example, because the sets in the original instance cover all points in the domain, $D \vee z$ is already a small equivalent formula which does not depend at all on whether or not the instance of SUCCINCT SET COVER was a positive instance. But, we can solve this problem by modifying D initially to not accept the assignment in which every variable is set to *true*.

A more general technique that we use in several places in the reductions is "weighting" some variables in order to control the form of candidate small equivalent expressions. This is accomplished by replacing a single variable y with a conjunction of new variables, $y_1 \wedge \cdots \wedge y_w$, where w is the desired weight. We show that after this replacement, a minimum equivalent expression must be at least as large as the "w-minimum" expression (in which the size of a formula is measured by the number of occurrences of variables other than y plus w times the number of occurrences of the variable y).

We use this technique, for example, to weight z so highly that there can be only one occurrence of it; this then forms the conceptual pivot from which we argue that the subtrees of the formula surrounding that occurrence of z must compute D, and separately, a disjunction of as many variables as there are sets in a minimum cover of the original SUCCINCT SET COVER instance.

Outline. In Section 2 we define general notation, and the variants of the problems we will be considering. In Section 3 we give the reductions. We conclude in Section 4 with some open problems.

2 Preliminaries

Given a Boolean formula F, we use $|F|$ to mean the size of the formula F, and $\overline{F}$ for the negation of the formula F. Similarly, for a variable x, $\overline{x}$ is the negation of x.

Restrictions. Given a function $f : \{true, false\}^n \to \{true, false\}$ and a function $\rho : [n] \to \{true, false, free\}$, we define the *restriction* of f to ρ, f_ρ to be the function which fixes the ith input to $\rho(i)$ if $\rho(i)$ is not equal to *free*, and leaves it as an input otherwise. Similarly, if F is a formula for f, we define F_ρ to be the formula in which every instance of the ith input variable is replaced with $\rho(i)$ if $\rho(i) \neq free$, and is unchanged otherwise. Note that F_ρ is a formula for f_ρ.

Weighted formulae. If the variables x_i of some function f have associated weights $w(x_i)$, then the *w-weighted size* of a formula for f is the sum of the weights of the variables occurring at the leaves (in their multiplicity). The usual measure of formula size is the w-weighted size when $w(x_i) = 1$ for all x_i. Note that, as usual, size counts the number of literals at the leaves, and not the $(\vee, \wedge, \neg)$ gates.

Given a weight function w, we can take a formula F and create a formula F' which has minimum formula size that is at least the minimum w-weighted formula size of F. Formula F' is obtained by substituting $x_i^{(1)} \wedge x_i^{(2)} \wedge \cdots \wedge x_i^{(w(x_i))}$ for every occurrence of x_i in F. Note that by moving negations to the variable level, we are substituting $\overline{x_i^{(1)}} \vee \overline{x_i^{(2)}} \vee \cdots \vee \overline{x_i^{(w(x))}}$ for every occurrence of $\overline{x_i}$. We call F' the *w-expanded version* of F. The following lemma demonstrates the usefulness of this transformation:

Lemma 2.1. *Let F be a formula and w a weight function for F. Let F' be the w-expanded version of F. Then the minimum size of a formula equivalent to F' is at least the minimum w-weighted size of a formula equivalent to F.*

Proof. Consider a minimum formula $\widehat{F'}$ equivalent to F'. For each x_i, let $1 \leq j_i \leq w(x_i)$ be the integer for which $x_i^{(j_i)}$ occurs least among the x_i-leaves of $\widehat{F'}$. Consider the restriction ρ that for each i sets $x_i^{(j)}$ to *true* for $j \neq j_i$. By our choice of j_i, $|\widehat{F'}|$ is at least the w-weighted size of $\widehat{F}'_\rho$. But the formula $\widehat{F}'_\rho$ clearly is equivalent to F, so its w-weighted size is an upper bound on the minimum w-weighted size of a formula equivalent to F. □

2.1 The Problems

As mentioned in the introduction, we will reduce from the Σ_2^P-complete problem SUCCINCT SET COVER. It will be convenient to work with a slightly modified

version in which the goal is for the succinctly specified sets to cover everything *except* the "all-true" assignment.

*Problem 2.1 (*MODIFIED SUCCINCT SET COVER (MSSC)). Given a DNF formula D on variables $v_1, v_2, \ldots, v_m, x_1, x_2, \ldots, x_n$ and an integer k, is there a subset $I \subseteq \{1, 2, \ldots n\}$ with $|I| \leq k$ and for which $D \vee \bigvee_{i \in I} \overline{x_i} \equiv (\bigvee_{i=1}^{m} \overline{v_i} \vee \bigvee_{i=1}^{n} \overline{x_i})$?

It's easy to see that this variant of SUCCINCT SET COVER is Σ_2^P-complete by reducing from SUCCINCT SET COVER:

Theorem 2.1. MSSC *is* Σ_2^P*-complete under Turing reductions.*

Proof. We are given an instance of SSC: a DNF D on variables $v_1, v_2, \ldots, v_m, x_1, x_2, \ldots, x_n$ and an integer k. We next produce the instance D' , defined by $D' = D \wedge (\bigvee_{i=1}^{m} \overline{v_i} \vee \bigvee_{i=1}^{n} \overline{x_i})$ (multiplied out into DNF) paired with the same integer k. If there exists $I \subseteq [n]$ of size at most k for which $D \vee \bigvee_{i \in I} \overline{x_i} \equiv 1$ then clearly $D' \vee \bigvee_{i \in I} \overline{x_i}$ accepts everything except the "all-true" assignment, and vice versa. □

Remark 1. In both SSC and MSSC, the instances produced by the reduction have the property that taking $I = \{1, 2, \ldots, n\}$ is a feasible solution.

The central problem we are concerned with in this paper is:

*Problem 2.2 (*MINIMUM EQUIVALENT EXPRESSION (MEE)). Given a Boolean $(\wedge, \vee, \neg)$-formula F and an integer k, is there an equivalent $(\wedge, \vee, \neg)$-formula of size at most k?

We also consider the constant-depth versions. When discussing constant-depth formulas, as usual, we allow arbitrary fanin AND and OR gates and we use the convention that all NOT gates occur at the variable level.

*Problem 2.3 (*MINIMUM EQUIVALENT DEPTH d EXPRESSION (MEE_d)). Given a depth d Boolean formula F and an integer k, is there an equivalent depth d formula of size at most k?

While distributing the NOT gates to the variable level clearly does not affect formula size, it's not as clear that finding the minimum depth-d formula is equivalent to finding the minimum depth-d formula with an OR gate at the root. The latter variant, defined below, will be easier to work with.

*Problem 2.4 (*MINIMUM EQUIVALENT DEPTH d EXPRESSION WITH A TOP OR GATE (MEE_d-OR)). Given a depth d Boolean formula F and an integer k, is there an equivalent depth d formula with top OR gate, of size at most k?

In Theorem 3.3 we reduce MEE_d-OR to MEE_d, so that Σ_2^p-hardness for the latter follows from the Σ_2^p-hardness for the former.

3 Main Results

In this section we prove:

Theorem 3.1. *For every $d \geq 3$, the problem* MEE_d *is Σ_2^P-complete under polynomial time Turing reductions.*

Theorem 3.2. *The problem* MEE *is Σ_2^P-complete under polynomial time Turing reductions.*

These theorems are proved via Theorem 3.3 and the reductions in Sections 3.1 and 3.2. Theorem 3.3 allows us to restrict our attention to the MEE_d−OR problem, rather than the general MEE_d problem. Its proof is contained in the full version of this paper.

Theorem 3.3. *For every $d \geqslant 3$, there is a polynomial-time reduction from* MEE_d−OR *to* MEE_d.

3.1 Main Reduction

The following is a Turing Reduction from MSSC to MEE_d−OR. We describe the steps of a Turing Machine with access to an oracle for MEE_d−OR:

- We are given an instance of MSSC: a DNF D on variables $v_1, v_2, \ldots, v_m, x_1, x_2, \ldots, x_n$ and an integer k. Let w be the weighting function with $w(x_i) = 1$ for all i and $w(v_i) = n + 1$ for all i, and let D' be the w-expanded version of D. Note that D' has only depth 3, as we are expanding a DNF formula.
- We make at most $\log |D'|$ calls to the oracle to find the size u of the smallest equivalent depth-d formula with top OR gate for D', using binary search.
- Define the formula E involving fresh variables y_i and z as follows:

$$E = D \vee [(\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_n} \vee \overline{y_1} \vee \cdots \vee \overline{y_{u+n}}) \wedge z].$$

  Let w' be the weighting function with $w'(x_i) = 1$ for all i, $w'(v_i) = n + 1$ for all i, $w'(y_i) = 1$ for all i and $w'(z) = 2u + k + n + 1$, and let F be the w'-expanded version of E. We will label the "copies" of z used in the expanded version $z_1, z_2, \ldots, z_{2u+k+n+1}$. Note that F has only depth 3.
- We ask the oracle if F has an equivalent depth-d formula with top OR gate, of size at most $4u + 2k + 2n + 1$. We will show that the answer is "yes" iff the original MSSC instance was a positive instance.

Remark 2. Note that since this reduction utilizes logarithmically many adaptive oracle calls, it can be transformed using standard techniques into a nonadaptive reduction utilizing polynomially many oracle calls.

The remainder of this section is devoted to proving the following theorem:

Theorem 3.4. *Let $\widehat{F}$ be a minimum equivalent depth-d formula with top OR gate for F. Then $|\widehat{F}| \leqslant 4u + 2k + 2n + 1$ iff there exists $I \subseteq \{1, 2, \ldots, n\}$ with $|I| \leq k$ and and for which $D \vee \bigvee_{i \in I} \overline{x_i} \equiv (\bigvee_{i=1}^{m} \overline{v_i} \vee \bigvee_{i=1}^{n} \overline{x_i})$.*

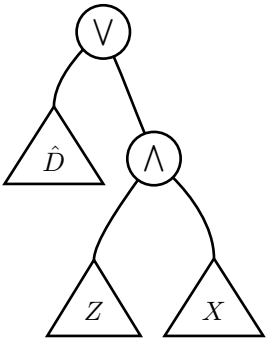


Fig. 1. The desired form of an equivalent formula for F. Here $\widehat{D}$ is a minimum depth-d formula with top OR gate equivalent to D', $Z = \bigwedge_{i=1}^{2u+k+n+1} z_i$, and X is of the form $\bigvee_{i\in I} \overline{x_i} \vee \bigvee_{i=1}^{u+n} \overline{y_i}$.

As a point of reference, Figure 1 shows the "intended" form of a minimum equivalent depth-d formula for F. Of course for one direction of the reduction we will need to show that a small formula must have this form, which is a somewhat involved argument.

In the forward (easy) direction, we claim that if the instance of MSSC is a positive instance, then there is a depth-d formula equivalent to F, of the form pictured in Figure 1, and with size at most $4u+2k+2n+1$. Let $\widehat{D}$ be a depth-d formula with top OR gate equivalent to D' of size u, and let $I \subseteq \{1, 2, \ldots, n\}$ be a set of size at most k for which $D \vee \bigvee_{i\in I} \overline{x_i} \equiv \left(\bigvee_{i=1}^{m} \overline{v_i} \vee \bigvee_{i=1}^{n} \overline{x_i}\right)$, (such a set I exists because the MSSC instance is a positive instance). Then $\widehat{D} \vee \left[\left(\bigvee_{i=1}^{2u+k+n+1} z_i\right) \wedge \left(\bigvee_{i\in I} \overline{x_i} \vee \bigvee_{i=1}^{u+n} \overline{y_i}\right)\right]$ is a depth-d formula equivalent to F of size $4u + 2k + 2n + 1$. Furthermore, it is of the form pictured in Figure 1.

In the other direction, we assume that the MSSC instance is a negative instance, and we wish to show that there is *no* depth-d formula with top OR gate equivalent to F of size at most $4u + 2k + 2n + 1$. Let $\widehat{F}$ be a minimum depth-d formula for F.

We will derive from $\widehat{F}$ a minimum depth-d formula for F that is of the form pictured in Figure 1. We prove this in the next three subsections. Note that if $\widehat{F}$ has size larger than $4u+2k+2n+1$, then we are done; therefore we will assume the contrary in what follows.

The z variable. First, we show that there is some i for which z_i occurs exactly once in $\widehat{F}$ and that it does not occur negated.

Lemma 3.1. *If a formula for F has size at most $4u + 2k + 2n + 1$, then there is some i such that z_i occurs exactly once.*

Proof. If the formula is of size at most $4u + 2k + 2n + 1$ and yet contains two or more copies of each z_i, then this is a contradiction as the number of occurrences of z_i variables alone is $2(2u + k + n + 1) = 4u + 2k + 2n + 2 > 4u + 2k + 2n + 1$.

Thus, some z_i must occur at most once. Now, since we are assuming that D came from a negative instance of MSSC, we know that D rejects some assignment to its variables other than the all-true assignment. On the other hand E accepts this assignment when the z variable is true. This implies that E depends on z and that F (the w'-expanded version of E) depends on each z_i. So some z_i occurs *exactly* once. □

Fix an i for which z_i occurs exactly once. Now, let ρ be the restriction that restricts all z_j for $j \neq i$ to *true*, and leaves all other variables free. From now on we will be working with $\widehat{F}_\rho$, which has only a single z variable.

Lemma 3.2. *Let f be the function corresponding to $\widehat{F}_\rho$. Further, let ρ_0 restrict z_i to false, leaving all other variables free and ρ_1 restrict z_i to true, leaving all other variables free. If z_i appears negated in $\widehat{F}_\rho$, then $f_{\rho_1} \Rightarrow f_{\rho_0}$.*

Proof. Consider any restriction σ that assigns all variables except z_i to 0/1 and leaves z_i free. $(\widehat{F}_\rho)_\sigma$ takes in the single input $\overline{z_i}$. Since there are no negations other than at the variable level, $(\widehat{F}_\rho)_\sigma$ is monotone in $\overline{z_i}$. Thus, $f_\sigma(true) \Rightarrow f_\sigma(false)$. Since this is true for all σ, $f_{\rho_1} \Rightarrow f_{\rho_0}$. □

Now, if we substitute *false* for z_i in $\widehat{F}_\rho$, the function that this formula computes is equivalent to D', and if we substitute *true*, it accepts everything except the "all-true" assignment to the x, y, and w-weighted v variables. Defining f, ρ_0, ρ_1 as in Lemma 3.2 (and again using the fact that D comes from a negative instance of MSSC) we have that $f_{\rho_1} \not\Rightarrow f_{\rho_0}$. Lemma 3.2 then tells us that z_i occurs non-negated in the formula $\widehat{F}_\rho$.

Properties of $\widehat{F}_\rho$. In the remainder of the proof, we will use ALLTRUE as shorthand for the all-true assignment to the variables of $\widehat{F}_\rho$ – namely, the x variables, y variables, the w'-expanded v variables, and z_i. Similarly $\overline{ALLTRUE}$ refers to the function that accepts every assignment to those variables except ALLTRUE. For future reference, we record a few useful properties of $\widehat{F}_\rho$, for which straightforward proofs appear in the full version of this paper:

Lemma 3.3. *The following properties regarding $\widehat{F}_\rho$ hold:*

1. $|\widehat{F}_\rho| \leq |\widehat{F}| - (2u + k + n)$
2. *$\widehat{F}_\rho$ is a minimum formula*
3. *When z_i is true, $\widehat{F}_\rho$ is equivalent to the formula $\overline{ALLTRUE}$ with z_i set to true.*
4. *When z_i is false, $\widehat{F}_\rho$ is equivalent to D'.*

The X subformula. In this section we show (Lemma 3.5) that a minimum formula accepting *at least* all of the assignments to the v, x and y variables not accepted by D' and not accepting the "all-ones" assignment is of the form $\left(\bigvee_{i \in I} \overline{x_i} \vee \bigvee_{i=1}^{u+n} \overline{y_i}\right)$. We will eventually use this to argue that z_i's sibling subformula in $\widehat{F}_\rho$ has the intended form, and in a technical part of Section 3.1.

The following general lemma will be useful.

Lemma 3.4. *Let $t_1, t_2, \ldots, t_n$ be a set of variables, and S a subset of $\{0,1\}^n$. A minimum formula accepting at least S and not the all-true assignment is of the form $\bigvee_{i \in I} \overline{t_i}$ for some $I \subseteq \{1, 2, \ldots, n\}$.*

Proof. Let T be a formula accepting at least S and rejecting the all-true assignment. Suppose that T depends on ℓ variables. Then $|T| \geqslant \ell$. Furthermore, in each assignment accepted by T, one of these variables is set to *false*, as T does not accept all-true. Therefore, if T depends on variables t_i for $i \in I$, the formula $T' = \bigvee_{i \in I} \overline{t_i}$ accepts at least everything that T accepts. Furthermore T' does not accept all-true and $|T'| = \ell \leqslant |T|$.

Thus, given a minimum formula T that accepts at least S but not all-true, we can find another minimum formula accepting at least S but not all-true, of the desired form. □

Applying the lemma in our setting yields the following, whose proof appears in the full version of this paper:

Lemma 3.5. *Let S be the set of assignments to the y variables plus the variables of D' (the w-expanded v variables and the x variables), that are not accepted by D'. Then a minimum formula accepting at least S but not the all-true assignment to these variables is of the form $\bigvee_{i \in I} \overline{x_i} \vee \bigvee_{i=1}^{u+n} \overline{y_i}$ for some $I \subseteq \{1, 2, \ldots, n\}$.*

Position of the z variable. Finally, we show (Lemma 3.9) that there is a minimum depth-d formula with top OR gate, equivalent to $\widehat{F}_\rho$, in which z_i occurs directly under a second-level AND gate. We begin with two general lemmas

Lemma 3.6. *Let A be a sub-formula of formula G, and suppose formula B implies G. Then, the formula obtained by replacing A with $A \vee B$ in G is equivalent to G.*

Proof. Because all of the negations have been pushed to the variable level, flipping the result of a non-input gate from *false* to *true* can only change the output of the formula from *false* to *true*, and not the reverse. Since we are replacing A with $A \vee B$, this can only change the result of the top gate of A from *false* to *true*. Furthermore, since B implies G, this can only occur when G is already true, and thus it will not change the output. □

Lemma 3.7. *Let A be a sub-formula of formula G that implies G. Then, the formula G' obtained by replacing A with false in G, and then taking the disjunction of this new formula with A is equivalent to G.*

Proof. In the case that A is true, then G must be true because A implies G. In this case G' will also be true, as A occurs directly beneath the top-level OR in G'. In the case that A is false, G' is equivalent to G with A replaced by *false*, so G' has the same result as G in this case as well. □

Note that the transformation in Lemma 3.7 does not increase the size of the formula, nor its depth if G already has a top OR gate. We now describe how the above general lemmas will be applied to $\widehat{F}_\rho$:

Lemma 3.8. *Suppose that $\widehat{F}_\rho$ has a sub-formula $A \wedge I$. If $I \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$, then there is an equivalent depth-d formula with top OR gate, $\widehat{F}'_\rho$, where either $|\widehat{F}'_\rho| < |\widehat{F}_\rho|$ or $|\widehat{F}'_\rho| = |\widehat{F}_\rho|$ and $A \wedge I$ occurs at the second level.*

Proof. By assumption $I \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$, and so $A \wedge I \wedge \overline{ALLTRUE}$ does as well. If $A \wedge I \wedge \overline{ALLTRUE}$ is equivalent to $A \wedge I$, then $A \wedge I$ implies $\widehat{F}_\rho$. Then by Lemma 3.7, there is an equivalent formula $\widehat{F}'_\rho$ with $|\widehat{F}'_\rho| = |\widehat{F}_\rho|$ and $A \wedge I$ occuring at the second level.

Otherwise $A \wedge I$ accepts ALLTRUE. Since $A \wedge I$ accepts ALLTRUE, A must accept ALLTRUE, so $A \vee \overline{ALLTRUE} \equiv 1$. Thus, $I \wedge (A \vee \overline{ALLTRUE})$ is equivalent to I. Thus, by Lemma 3.6, we can replace $A \wedge I$ with I, resulting in a formula $\widehat{F}'_\rho$ which is equivalent to $\widehat{F}_\rho$ and $|\widehat{F}'_\rho| < |\widehat{F}_\rho|$. □

Lemma 3.9. *There is a depth-d formula with top OR gate, $\widehat{F}'_\rho$, that is equivalent to $\widehat{F}_\rho$ and of no larger size, in which z_i occurs exactly once, non-negated, and directly under a second-level AND gate.*

Proof. The proof is by case analysis. There are four cases depending on where z_i occurs in $\widehat{F}_\rho$: under the top-level OR gate, under an AND gate below the second level, under an OR gate below the third level, or under an OR gate at the third level. In each case, we either derive a contradiction or show that the formula can be transformed to the desired case of the z_i occurring directly under a second-level AND gate, without increasing size.

The full proof appears in the full version of this paper. □

Finishing up. We now finish our proof using Lemma 3.10.

Lemma 3.10. *There is a minimum depth-d formula with top OR gate equivalent to F, of the form pictured in Figure 2.*

Proof. By Lemma 3.9, there is a minimum depth-d formula with top OR gate equivalent to $\widehat{F}_\rho$ that is of the form in Figure 2, but with the Z subformula

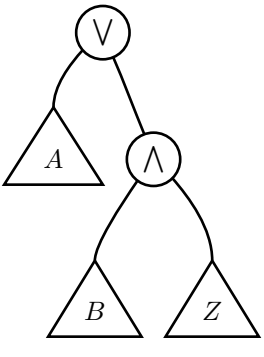


Fig. 2. The required form of a minimum depth-d formula with top OR gate equivalent to F

replaced by z_i. Replacing z_i with $\bigwedge_{j=1}^{2u+k+n+1} z_j$, we obtain a depth-$d$ formula for F, of the form pictured in Figure 2, of size at most $|\widehat{F}_\rho| + 2u + k + n$. By Lemma 3.3 (1), this quantity is a lower bound on the size of a minimum depth-d formula with top OR gate equivalent to F, so it must be minimum. □

Now we are finally able to argue that $|\widehat{F}|$ must be larger than $4u + 2k + 2n + 1$. First, observe that by Lemma 3.10, there is a depth-d formula equivalent to F of the form pictured in Figure 2 whose size is the same as the size of $\widehat{F}$. In this formula, when Z is set to false, the function simplifies to just A, and by the definition of F, this must be equivalent to D', and hence it must have size at least u. On the other hand, when Z is set to *true*, the formula must accept every assignment in which at least one variable is set to *false*. This means that B must accept everything not accepted by D' except the "all-true" assignment. By Lemma 3.5, we know that we can assume B to be a disjunction of negated variables, and that it must have size at least $u + n + k + 1$ (because the original MSSC instance was a negative instance). Adding the sizes of A and B to the number of z_i variables in Z, we have a formula of size at least $4u + 2k + 2n + 2$. We conclude that $|\widehat{F}| \geqslant 4u + 2k + 2n + 2 > 4u + 2k + 2n + 1$ as required.

This completes the proof of Theorem 3.4.

3.2 The Unbounded Depth Case

The reduction for MEE is the same as the reduction in the previous section (3.1). We never used the depth-d restriction in any of the arguments in that reduction; it was only mentioned in the context of ensuring that various manipulations *maintained* depth-d, a constraint that is no longer operative for the unbounded depth case.

It is still convenient in the reduction to think of formulas with alternating levels of unbounded-fanin AND and OR gates, and here we simply note that discussing the size of such formulas is the same as discussing the size of standard, fanin-2 $(\wedge, \vee, \neg)$-formulas.

Proposition 1. *If F is a formula with unbounded-fanin AND and OR gates of size s, then there is an equivalent formula F' with fanin-2 AND and OR gates of size s. Similarly if there is a formula F with fanin-2 AND and OR gates of size s, there is an equivalent formula F' with unbounded-fanin AND and OR gates of size s.*

4 Conclusions and Open Problems

The most natural open problems remaining are to give many-one reductions for the problems in this paper (rather than Turing reductions), and to resolve the complexity of the circuit version of the problem. Our techniques here rely heavily on the fact that we are dealing with formulas rather than circuits.

Another important direction is to study the approximability of the problems in this paper. For the depth-2 case (DNF minimization) it is known that the

problems are inapproximable to within very large (N^{ε}) factors [Uma99a]. Our reductions are quite fragile and do not seem to give any hardness of approximation results for these problems.

Finally, we note that the complexity of the "Π_2^P" versions of all of these problems remain open. These are problems of the form: given a Boolean circuit (of some specified form), is it a minimum circuit (of the specified form). Even for depth two (DNFs), this problem is not known to be Π_2^P-complete, although it is conjectured to be complete for that class.

References

[DGK94] Devadas, S., Ghosh, A., Keutzer, K.: Logic synthesis. McGraw-Hill, New York (1994)

[GJ79] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)

[HW97] Hemaspaandra, E., Wechsung, G.: The minimization problem for Boolean formulas. In: FOCS, pp. 575–584 (1997)

[KC00] Kabanets, V., Cai, J.-Y.: Circuit minimization problem. In: STOC, pp. 73–79 (2000)

[MS72] Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: FOCS, pp. 125–129. IEEE, Los Alamitos (1972)

[Sto76] Stockmeyer, L.J.: The polynomial-time hierarchy. Theor. Comput. Sci. 3(1), 1–22 (1976)

[Uma98] Umans, C.: The minimum equivalent DNF problem and shortest implicants. In: FOCS, pp. 556–563 (1998)

[Uma99a] Umans, C.: Hardness of approximating Σ_2^P minimization problems. In: FOCS, pp. 465–474 (1999)

[Uma99b] Umans, C.: On the Complexity and Inapproximability of Shortest Implicant Problems. In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 687–696. Springer, Heidelberg (1999)

[UVSV06] Umans, C., Villa, T., Sangiovanni-Vincentelli, A.L.: Complexity of two-level logic minimization. IEEE Trans. on CAD of Integrated Circuits and Systems 25(7), 1230–1246 (2006)

Optimal Cryptographic Hardness of Learning Monotone Functions*

Dana Dachman-Soled, Homin K. Lee, Tal Malkin,
Rocco A. Servedio, Andrew Wan, and Hoeteck Wee

Columbia University
{dglasner,homin,tal,rocco,atw12,hoeteck}@cs.columbia.edu

Abstract. A wide range of positive and negative results have been established for learning different classes of Boolean functions from uniformly distributed random examples. However, polynomial-time algorithms have thus far been obtained almost exclusively for various classes of *monotone* functions, while the computational hardness results obtained to date have all been for various classes of general (nonmonotone) functions. Motivated by this disparity between known positive results (for monotone functions) and negative results (for nonmonotone functions), we establish strong computational limitations on the efficient learnability of various classes of monotone functions.

We give several such hardness results which are provably almost optimal since they nearly match known positive results. Some of our results show cryptographic hardness of learning polynomial-size monotone circuits to accuracy only slightly greater than $1/2 + 1/\sqrt{n}$; this accuracy bound is close to optimal by known positive results (Blum *et al.*, FOCS '98). Other results show that under a plausible cryptographic hardness assumption, a class of constant-depth, sub-polynomial-size circuits computing monotone functions is hard to learn; this result is close to optimal in terms of the circuit size parameter by known positive results as well (Servedio, Information and Computation '04). Our main tool is a complexity-theoretic approach to hardness amplification via noise sensitivity of monotone functions that was pioneered by O'Donnell (JCSS '04).

1 Introduction

More than two decades ago Valiant introduced the Probably Approximately Correct (PAC) model of learning Boolean functions from random examples [Val84]. Since that time a great deal of research effort has been expended on trying to understand the inherent abilities and limitations of computationally efficient learning algorithms. This paper addresses a discrepancy between known positive and negative results for uniform distribution learning by establishing strong computational hardness results for learning various classes of *monotone* functions.

* Supported by NSF award CNS-0716245, DARPA grant HR0011-07-1-0012, NSF CAREER award CCF-0347282, NSF CAREER CCF-03-047839, NSF award SBE-0245014, NSF award CCF-0523664, and a Sloan Foundation Fellowship.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 36–47, 2008.

1.1 Background and Motivation

In the uniform distribution PAC learning model, a learning algorithm is given access to a source of independent random examples $(x, f(x))$ where each x is drawn uniformly from the n-dimensional Boolean cube and f is the unknown Boolean function to be learned. The goal of the learner is to construct a high-accuracy hypothesis function h, *i.e.,* one which satisfies $\Pr[f(x) \neq h(x)] \leq \epsilon$ where the probability is with respect to the uniform distribution and ϵ is an error parameter given to the learning algorithm. Algorithms and hardness results in this framework have interesting connections with topics such as discrete Fourier analysis [Man94], circuit complexity [LMN93], noise sensitivity and influence of variables in Boolean functions [KKL88, KOS04, OS07], and cryptography [BFKL93, Kha95]. For these reasons, and because the model is natural and elegant in its own right, the uniform distribution learning model has been intensively studied for almost two decades.

Monotonicity makes learning easier. For many classes of functions, uniform distribution learning algorithms have been devised which substantially improve on a naive exponential-time approach to learning via brute-force search. However, despite intensive efforts, researchers have not yet obtained poly(n)-time learning algorithms in this model for various simple classes of functions. Interestingly, in many of these cases restricting the class of functions to the corresponding class of *monotone* functions has led to more efficient – sometimes poly(n)-time – algorithms. We list some examples:

1. A simple algorithm learns monotone $O(\log n)$-juntas to perfect accuracy in poly(n) time, and a more complex algorithm [BT96] learns monotone $\tilde{O}(\log^2(n))$-juntas to any constant accuracy in poly(n) time. In contrast, the fastest known algorithm for learning arbitrary k-juntas runs in time $n^{.704k}$ [MOS04].
2. The fastest known uniform distribution learning algorithm for the general class of s-term DNF, due to Verbeurgt [Ver90], runs in time $n^{O(\log s)}$ to learn to any constant accuracy. In contrast, for s-term monotone DNF [Ser04] gives an algorithm which runs in $s^{O(\log s)}$ time. Thus the class of $2^{O(\sqrt{\log n})}$-term monotone DNF can be learned to any constant accuracy in poly(n) time, but no such result is known for $2^{O(\sqrt{\log n})}$-term general DNF.
3. The fastest known algorithm for learning poly(n)-size general decision trees to constant accuracy takes $n^{O(\log n)}$ time (this follows from [Ver90]), but poly(n)-size decision trees that compute monotone functions can be learned to any constant accuracy in poly(n) time [OS07].
4. No poly(n)-time algorithm can learn the general class of all Boolean functions on $\{0,1\}^n$ to accuracy better than $\frac{1}{2} + \frac{\text{poly}(n)}{2^n}$, but a simple polynomial-time algorithm can learn the class of all monotone Boolean functions to accuracy $\frac{1}{2} + \frac{\Omega(1)}{\sqrt{n}}$ [BBL98]. We note also that the result of [BT96] mentioned above follows from a $2^{\tilde{O}(\sqrt{n})}$-time algorithm for learning arbitrary

monotone functions on n variables to constant accuracy (it is easy to see that no comparable algorithm can exist for learning arbitrary Boolean functions to constant accuracy).

Cryptography and hardness of learning. Essentially all known representation-independent hardness of learning results (*i.e.,* results which apply to learning algorithms that do not have any restrictions on the syntactic form of the hypotheses they output) rely on some cryptographic assumption, or an assumption that easily implies a cryptographic primitive. For example, under the assumption that certain subset sum problems are hard on average, Kharitonov [Kha95] showed that the class AC^1 of logarithmic-depth, polynomial-size AND/OR/NOT circuits is hard to learn under the uniform distribution. Subsequently Kharitonov showed [Kha93] that if factoring Blum integers is 2^{n^ϵ}-hard for some fixed $\epsilon > 0$, then even the class AC^0 of constant-depth, polynomial-size AND/OR/NOT circuits similarly cannot be learned in polynomial time under the uniform distribution. In later work, Naor and Reingold [NR04] gave constructions of pseudorandom functions with very low circuit complexity; their results imply that if factoring Blum integers is super-polynomially hard, then even depth-5 TC^0 circuits (composed of MAJ and NOT gates) cannot be learned in polynomial time under uniform. We note that all of these hardness results apply even to algorithms which may make black-box "membership queries" to obtain the value $f(x)$ for inputs x of their choosing.

Monotonicity versus cryptography? Given that cryptography precludes efficient learning while monotonicity seems to make efficient learning easier, it is natural to investigate how these phenomena interact. One could argue that prior to the current work there was something of a mismatch between known positive and negative results for uniform-distribution learning: as described above a fairly broad range of polynomial-time learning results had been obtained for various classes of monotone functions, but there were no corresponding computational hardness results for monotone functions. Can all monotone Boolean functions computed by polynomial-size circuits be learned to 99% accuracy in polynomial time from uniform random examples? As far as we are aware, prior to our work answers were not known even to such seemingly basic questions about learning monotone functions as this one. This gap in understanding motivated the research presented in this paper (which, as we describe below, lets us answer "no" to the above question in a strong sense).

1.2 Our Results and Techniques: Cryptography Trumps Monotonicity

We present several different constructions of "simple" (polynomial-time computable) monotone Boolean functions and prove that these functions are hard to learn under the uniform distribution, even if membership queries are allowed. We now describe our main results, followed by a high-level description of how we obtain them.

In [BBL98] Blum *et al.* showed that arbitrary monotone functions cannot be learned to accuracy better than $\frac{1}{2} + \frac{O(\log n)}{\sqrt{n}}$ by any algorithm which makes

poly(n) many membership queries. This is an information-theoretic bound which is proved using randomly generated monotone DNF formulas of size (roughly) $n^{\log n}$. A natural goal is to obtain *computational* lower bounds for learning polynomial-time-computable monotone functions which match, or nearly match, this level of hardness (which is close to optimal by the $(\frac{1}{2} + \frac{\Omega(1)}{\sqrt{n}})$-accuracy algorithm of Blum *et al.* described above). We prove near-optimal hardness for learning polynomial-size monotone circuits:

Theorem 1 (informal). *If one-way functions exist, then there is a class of* poly(n)*-size monotone circuits that cannot be learned to accuracy* $\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$ *for any fixed* $\epsilon > 0$.

Our approach yields even stronger lower bounds if we make stronger assumptions:

- Assuming the existence of subexponential one-way functions, we improve the bound on the accuracy to $1/2 + \text{polylog}(n)/n^{1/2}$.
- Assuming the hardness of factoring Blum integers, our hard-to-learn functions may be computed in monotone NC^1.
- Assuming that Blum integers are 2^{n^ϵ}-hard to factor on average (the same hardness assumption used in Kharitonov's work [Kha93]), we obtain a lower bound for learning constant-depth circuits of sub-polynomial size that almost matches the positive result in [Ser04]. More precisely, we show that for any (sufficiently large) constant d, the class of monotone functions computed by depth-d AND/OR/NOT circuits of size $2^{(\log n)^{O(1)/(d+1)}}$ cannot be learned to accuracy 51% under the uniform distribution in poly(n) time. In contrast, the positive result of [Ser04] shows that monotone functions computed by depth-d AND/OR/NOT circuits of size $2^{O((\log n)^{1/(d+1)})}$ can be learned to any constant accuracy in poly(n) time.

These results are summarized in Figure 1.

Proof techniques. A natural first approach is to try to "pseudorandomize" [BBL98]'s construction of random $n^{\log n}$-term monotone DNFs. While we were not able to do this directly, it turns out that a closely related approach does yield some results along the desired lines. In the full version of the paper (available online), we present and analyze a simple variant of the [BBL98] information-theoretic construction and then show how to "pseudorandomize" the variant. Since our variant gives a weaker quantitative bound on the information-theoretic hardness of learning than [BBL98], this gives a construction of polynomial-time-computable monotone functions which, assuming the existence of one-way functions, cannot be learned to accuracy $\frac{1}{2} + \frac{1}{\text{polylog}(n)}$ under the uniform distribution. While this answers the question posed above (even with "51%" in place of "99%"), the $\frac{1}{\text{polylog}(n)}$ factor is rather far from the $\frac{O(\log n)}{\sqrt{n}}$ factor that one might hope for as described above.

In Section 2 we use a different construction to obtain much stronger quantitative results; another advantage of this second construction is that it enables

Hardness assumption	Complexity of f	Accuracy bound	Ref.
none	random $n^{\log n}$-term mono. DNF	$\frac{1}{2} + \frac{\omega(\log n)}{n^{1/2}}$	[BBL98]
OWF (poly)	poly-size monotone circuits	$\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$	Thm. 1
OWF (2^{n^α})	poly-size monotone circuits	$\frac{1}{2} + \frac{\text{poly}(\log n)}{n^{1/2}}$	Thm. 3
factoring BI (poly)	monotone NC^1-circuits	$\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$	FV
factoring BI (2^{n^α})	AND/OR/NOT circuits of size $2^{(\log n)^{O(1)/(d+1)}}$ and depth d	$\frac{1}{2} + o(1)$	FV

Fig. 1. Summary of known hardness results for learning monotone Boolean functions. The meaning of each row is as follows: under the stated hardness assumption, there is a class of monotone functions computed by circuits of the stated complexity which no poly(n)-time membership query algorithm can learn to the stated accuracy. In the first column, OWF and BI denote one-way functions and Blum Integers respectively, and "poly" and "2^{n^α}" means that the problems are intractable for poly(n)- and 2^{n^α}-time algorithms respectively (for some fixed $\alpha > 0$). Recall that the poly(n)-time algorithm of [BBL98] for learning monotone functions implies that the best possible accuracy bound for monotone functions is $\frac{1}{2} + \frac{\Omega(1)}{n^{1/2}}$. "FV" means the result is established in the full version of this paper.

us to show hardness of learning *monotone circuits* rather than just circuits computing monotone functions. We start with the simple observation that using standard tools it is easy to construct polynomial-size monotone circuits computing "slice" functions which are pseudorandom on the middle layer of the Boolean cube $\{0,1\}^n$. Such functions are easily seen to be mildly hard to learn, *i.e.,* hard to learn to accuracy $1 - \frac{\Omega(1)}{\sqrt{n}}$. We then use the elegant machinery of hardness amplification of monotone functions which was pioneered by O'Donnell [O'D04] to amplify the hardness of this construction to near-optimal levels (rows 2–4 of Figure 1). We obtain our result for constant depth, sub-polynomial-size circuits (row 5 of Figure 1) by augmenting this approach with an argument which at a high level is similar to one used in [AHM+06], by "scaling down" and modifying our hard-to-learn functions using a variant of Nepomnjaščiĭ's theorem [Nep70].

1.3 Preliminaries

We consider Boolean functions $f : \{0,1\}^n \to \{0,1\}$. We view $\{0,1\}^n$ as endowed with the natural partial order $x \leq y$ iff $x_i \leq y_i$ for all $i = 1, \ldots, n$. A Boolean function f is *monotone* if $x \leq y$ implies $f(x) \leq f(y)$.

We establish that a class $\mathcal{C}$ of functions is hard to learn by showing that for a uniform random $f \in \mathcal{C}$, the expected error of any poly(n)-time learning algorithm L is close to $1/2$ when run with f as the target function. Thus we bound the quantity

$$\Pr_{f \in \mathcal{C}, x \in \{0,1\}^n}[L^f(1^n) \to h; h(x) = f(x)] \tag{1}$$

where the probability is also taken over any internal randomization of the learning algorithm L. We say that *class $\mathcal{C}$ is hard to learn to accuracy* $\frac{1}{2} + \epsilon(n)$ if for every poly(n)-time membership query learning algorithm L (*i.e.,* p.p.t. oracle algorithm), we have (11) $< \frac{1}{2} + \epsilon(n)$ for all sufficiently large n. As noted in [BBL98], it is straightforward to transform a lower bound of this sort into a lower bound for the usual ϵ, δ formulation of PAC learning.

2 Lower Bounds Via Hardness Amplification of Monotone Functions

In this section we prove our main hardness results, summarized in Figure 1, for learning various classes of monotone functions under the uniform distribution with membership queries.

Let us start with a high-level explanation of the overall idea. Inspired by the work on hardness amplification within NP initiated by O'Donnell [O'D04, HVV06], we study constructions of the form

$$f(x_1, \ldots, x_m) = C(f'(x_1), \ldots, f'(x_m))$$

where C is a Boolean "combining function" with low noise stability (we give precise definitions later) which is both *efficiently computable* and *monotone*. Recall that O'Donnell showed that if f' is weakly hard to compute and the combining function C has low noise stability, then f is very hard to compute. This result holds for general (non-monotone) functions C, and thus generalizes Yao's XOR lemma, which addresses the case where C is the XOR of m bits (and hence has the lowest noise stability of all Boolean functions, see [O'D04]).

Roughly speaking, we establish an analogue of O'Donnell's result for learning. Our analogue, given in Section 2.2, essentially states that for certain well-structured[1] functions f' that are hard to learn to high accuracy, if C has low noise stability then f is hard to learn to accuracy even slightly better than 1/2. Since our ultimate goal is to establish that "simple" classes of monotone functions are hard to learn, we shall use this result with combining functions C that are computed by "simple" monotone Boolean circuits. In order for the overall function f to be monotone and efficiently computable, we need the initial f' to be well-structured, monotone, efficiently computable, and hard to learn to high accuracy. Such functions are easily obtained by a slight extension of an observation of Kearns *et al.* [KLV94]. They noted that the middle slice f' of a random Boolean function on $\{0,1\}^k$ is hard to learn to accuracy greater than $1 - \Theta(1/\sqrt{k})$ [BBL98, KLV94]; by taking the middle slice of a *pseudorandom* function instead, we obtain an f' with the desired properties. In fact, by a result of Berkowitz [Ber82] this slice function is computable by a polynomial-size monotone circuit, so the overall hard-to-learn functions we construct are computed by polynomial-size monotone circuits.

[1] As will be clear, the proof requires that f' be balanced and have a "hard-core set."

Organization. In Section 2.2 we adapt the analysis in [O'D04, HVV06] to reduce the problem of constructing hard-to-learn monotone Boolean functions to constructing monotone combining functions C with low noise stability. In Section 2.3 we show how constructions and analyses from [O'D04, MO03] can be used to obtain a "simple" monotone combining function with low noise stability. In Section 2.4 we establish Theorems 2 and 3 (lines 2 and 3 of Figure 1) by making different assumptions about the hardness of the initial pseudorandom functions. Finally, by making specific number-theoretic assumptions (namely, the hardness of factoring Blum integers), we obtain hard-to-learn monotone Boolean functions that can be computed by very simple circuits (lines 4 and 5 of Figure 1); we defer the formal statements and proofs of these results to the full version.

2.1 Preliminaries

Functions. Let $C : \{0,1\}^m \to \{0,1\}$ and $f' : \{0,1\}^k \to \{0,1\}$ be Boolean functions. We write $C \circ f'^{\otimes m}$ to denote the Boolean function over $(\{0,1\}^k)^m$ given by

$$C \circ f'^{\otimes m}(x) = C(f'(x_1), \ldots, f'(x_m)), \qquad \text{where } x = (x_1, \ldots, x_m).$$

For $g : \{0,1\}^k \to \{0,1\}$, we write $\mathsf{slice}(g)$ to denote the "middle slice" function:

$$\mathsf{slice}(g)(x) = \begin{cases} 1 & \text{if } |x| > \lfloor k/2 \rfloor \\ g(x) & \text{if } |x| = \lfloor k/2 \rfloor \\ 0 & \text{if } |x| < \lfloor k/2 \rfloor. \end{cases}$$

It is immediate that $\mathsf{slice}(g)$ is a monotone Boolean function for any g.

Bias and noise stability. Following the analysis in [O'D04, HVV06], we shall study the bias and noise stability of various Boolean functions. Specifically, we adopt the following notations and definitions from [HVV06]. The *bias* of a 0-1 random variable X is defined to be

$$\mathrm{Bias}[X] \stackrel{\text{def}}{=} |\Pr[X = 0] - \Pr[X = 1]|.$$

Recall that a probabilistic Boolean function h on $\{0,1\}^k$ is a probability distribution over Boolean functions on $\{0,1\}^k$ (so for each input x, the output $h(x)$ is a 0-1 random variable). The *expected bias* of a probabilistic Boolean function h is

$$\mathrm{ExpBias}[h] \stackrel{\text{def}}{=} \mathrm{E}_x[\mathrm{Bias}[h(x)]].$$

Let $C : \{0,1\}^m \to \{0,1\}$ be a Boolean function and $0 \leq \delta \leq \frac{1}{2}$. The *noise stability of C at noise rate δ*, denoted $\mathrm{NoiseStab}_\delta[C]$, is defined to be

$$\mathrm{NoiseStab}_\delta[C] \stackrel{\text{def}}{=} 2 \cdot \Pr_{x,\eta}[C(x) = C(x \oplus \eta)] - 1$$

where $x \in \{0,1\}^m$ is uniform random, $\eta \in \{0,1\}^m$ is a vector whose bits are each independently 1 with probability δ, and $\oplus$ denotes bitwise XOR.

2.2 Hardness Amplification for Learning

Throughout this subsection we write m for $m(n)$ and k for $k(n)$. We shall establish the following:

Lemma 1. *Let $C : \{0,1\}^m \to \{0,1\}$ be a polynomial-time computable function. Let $\mathcal{G}'$ be the family of all 2^{2^k} functions from $\{0,1\}^k$ to $\{0,1\}$, where $n = mk$ and $k = \omega(\log n)$. Then the class $\mathcal{C} = \{f = C \circ \mathsf{slice}(g)^{\otimes m} \mid g \in \mathcal{G}'\}$ of Boolean functions over $\{0,1\}^n$ is hard to learn to accuracy*

$$\frac{1}{2} + \frac{1}{2}\sqrt{\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[C]} + o(1/n).$$

This easily yields Corollary 1, which is an analogue of Lemma 1 with pseudorandom rather than truly random functions, and which we use to obtain our main hardness of learning results.

Proof of Lemma 1: Let k, m be such that $mk = n$, and let $C : \{0,1\}^m \to \{0,1\}$ be a Boolean combining function. We prove the lemma by upper bounding

$$\Pr_{g \in \mathcal{G}', x \in \{0,1\}^n}\left[L^f(1^n) \to h;\ h(x) = f(x)\right] \tag{2}$$

where L is an arbitrary p.p.t. oracle machine (running in time $\mathrm{poly}(n)$ on input 1^n) that is given oracle access to $f \stackrel{\mathrm{def}}{=} C \circ \mathsf{slice}(g)^{\otimes m}$ and outputs some hypothesis $h : \{0,1\}^n \to \{0,1\}$.

We first observe that since C is computed by a uniform family of circuits of size $\mathrm{poly}(m) \leq \mathrm{poly}(n)$, it is easy for a $\mathrm{poly}(n)$-time machine to simulate oracle access to f if it is given oracle access to g. So (2) is at most

$$\Pr_{g \in \mathcal{G}',\ x \in \{0,1\}^n}\left[L^g(1^n) \to h;\ h(x) = (C \circ \mathsf{slice}(g)^{\otimes m})(x)\right]. \tag{3}$$

To analyze the above probability, suppose that in the course of its execution L never queries g on any of the inputs $x_1, \ldots, x_m \in \{0,1\}^k$, where $x = (x_1, \ldots, x_m)$. Then the *a posteriori* distribution of $g(x_1), \ldots, g(x_m)$ (for uniform random $g \in \mathcal{G}'$) given the responses to L's queries that it received from g is identical to the distribution of $g'(x_1), \ldots, g'(x_m)$, where g' is an independent uniform draw from $\mathcal{G}'$: both distributions are uniform random over $\{0,1\}^m$. (Intuitively, this just means that if L never queries the random function g on any of $x_1, \ldots, x_m$, then giving L oracle access to g does not help it predict the value of f on $x = (x_1, \ldots, x_m)$.) Since L runs in $\mathrm{poly}(n)$ time, for any fixed $x_1, \ldots, x_m$ the probability that L queried g on any of $x_1, \ldots, x_m$ is at most $\frac{m \cdot \mathrm{poly}(n)}{2^k}$. Hence (3) is bounded by

$$\Pr_{g, g' \in \mathcal{G}',\ x \in \{0,1\}^n}\left[L^g(1^n) \to h;\ h(x) = (C \circ \mathsf{slice}(g')^{\otimes m})(x)\right] + \frac{m \cdot \mathrm{poly}(n)}{2^k}. \tag{4}$$

The first summand in (4) is the probability that L correctly predicts the value $C \circ \mathsf{slice}(g')^{\otimes m}(x)$, given oracle access to g, where g and g' are independently

random functions and x is uniform over $\{0,1\}^n$. It is clear that the best possible strategy for L is to use a maximum likelihood algorithm, *i.e.,* predict according to the function h which, for any fixed input x, outputs 1 if and only if the random variable $(C \circ \mathsf{slice}(g')^{\otimes m})(x)$ (we emphasize that the randomness here is over the choice of g') is biased towards 1. The expected accuracy of this h is precisely

$$\frac{1}{2} + \frac{1}{2}\,\mathrm{ExpBias}[C \circ \mathsf{slice}(g')^{\otimes m}]. \tag{5}$$

Now fix $\delta \stackrel{\text{def}}{=} \binom{k}{\lfloor k/2 \rfloor}/2^k = \Theta(1/\sqrt{k})$ to be the fraction of inputs in the "middle slice" of $\{0,1\}^k$. We observe that the probabilistic function $\mathsf{slice}(g')$ (where g' is truly random) is "δ-random" in the sense of ([HVV06], Definition 3.1), *i.e.,* it is balanced, truly random on inputs in the middle slice, and deterministic on all other inputs. This means that we may apply a technical lemma [HVV06, Lemma 3.7]) to $\mathsf{slice}(g')$ (see also [O'D04]) to obtain

$$\mathrm{ExpBias}[C \circ \mathsf{slice}(g')^{\otimes m}] \leq \sqrt{\mathrm{NoiseStab}_\delta[C]}. \tag{6}$$

Combining (4), (5) and (6) and recalling that $k = \omega(\log n)$, we obtain Lemma 1. □

Corollary 1. *Let $C : \{0,1\}^m \to \{0,1\}$ be a polynomial-time computable function. Let $\mathcal{G}$ be a pseudorandom family of functions from $\{0,1\}^k$ to $\{0,1\}$ which are secure against $\mathrm{poly}(n)$-time adversaries, where $n = mk$ and $k = \omega(\log n)$. Then the class $\mathcal{C} = \{f = C \circ \mathsf{slice}(g)^{\otimes m} \mid g \in \mathcal{G}\}$ of Boolean functions over $\{0,1\}^n$ is hard to learn to accuracy*

$$\frac{1}{2} + \frac{1}{2}\sqrt{\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[C]} + o(1/n).$$

Proof. The corollary follows from the fact that (3) must differ from its pseudorandom counterpart,

$$\Pr_{g \in \mathcal{G},\, x \in \{0,1\}^n}\Big[L^g(1^n) \to h;\ h(x) = (C \circ \mathsf{slice}(g)^{\otimes m})(x)\Big], \tag{7}$$

by less than $1/n^2$ (in fact by less than any fixed $1/\mathrm{poly}(n)$). Otherwise, we would easily obtain a $\mathrm{poly}(n)$-time distinguisher that, given oracle access to g, runs L to obtain a hypothesis h and checks whether $h(x) = (C \circ \mathsf{slice}(g)^{\otimes m})(x)$ for a random x to determine whether g is drawn from $\mathcal{G}$ or $\mathcal{G}'$. □

By instantiating Corollary 1 with a "simple" monotone function C having low noise stability, we obtain strong hardness results for learning simple monotone functions. We exhibit such a function C in the next section.

2.3 A Simple Monotone Combining Function with Low Noise Stability

In this section we combine known results of [O'D04, MO03] to obtain:

Proposition 1. *Given a value k, let $m = 3^\ell d 2^d$ for ℓ, d satisfying $3^\ell \le k^6 < 3^{\ell+1}$ and $d \le O(k^{.35})$. Then there exists a monotone function $C : \{0,1\}^m \to \{0,1\}$ computed by a uniform family of $\mathrm{poly}(m)$-size, $\log(m)$-depth monotone circuits such that*

$$\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[C] \le O\Big(\frac{k^6 \log m}{m}\Big). \tag{8}$$

Note that in this proposition we may have m as large as $2^{\Theta(k^{.35})}$ but not larger. O'Donnell [O'D04] gave a lower bound of $\Omega(\frac{\log^2 m}{m})$ on $\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[C]$ for every monotone m-variable function C, so the above upper bound is fairly close to the best possible (within a polylog(m) factor if $m = 2^{k^{\Theta(1)}}$).

Following [O'D04, HVV06], we use the "recursive majority of 3" function and the "tribes" function in our construction. We require the following results on noise stability:

Lemma 2 ([O'D04]). *Let* Rec-Maj-$3_\ell : \{0,1\}^{3^\ell} \to \{0,1\}$ *be defined as follows: for $x = (x^1, x^2, x^3)$ where each $x^i \in \{0,1\}^{3^{\ell-1}}$,*

$$\text{Rec-Maj-}3_\ell(x) \stackrel{def}{=} \mathrm{Maj}(\text{Rec-Maj-}3_{\ell-1}(x^1), \text{Rec-Maj-}3_{\ell-1}(x^2), \text{Rec-Maj-}3_{\ell-1}(x^3)).$$

Then for $\ell \ge \log_{1.1}(1/\delta)$, we have $\mathrm{NoiseStab}_\delta[\text{Rec-Maj-}3_\ell] \le \delta^{-1.1}(3^\ell)^{-.15}$.

Lemma 3 ([MO03]). *Let* $\mathrm{Tribes}_d : \{0,1\}^{d2^d} \to \{0,1\}$ *denote the "tribes" function on $d2^d$ variables, i.e., the read-once 2^d-term monotone d-DNF*

$$\mathrm{Tribes}_d(x_1, \ldots, x_{d2^d}) \stackrel{def}{=} (x_1 \wedge \cdots \wedge x_d) \vee (x_{d+1} \wedge \cdots \wedge x_{2d}) \vee \cdots.$$

Then if $\eta \le O(1/d)$, we have $\mathrm{NoiseStab}_{\frac{1-\eta}{2}}[\mathrm{Tribes}_d] \le O\Big(\frac{\eta d^2}{d2^d}\Big) \le O\Big(\frac{1}{2^d}\Big)$.

Lemma 4 ([O'D04]). *If h is a balanced Boolean function and $\psi : \{0,1\}^r \to \{0,1\}$ is arbitrary, then for any δ we have*

$$\mathrm{NoiseStab}_\delta[\psi \circ h^{\otimes r}] = \mathrm{NoiseStab}_{\frac{1}{2} - \frac{\mathrm{NoiseStab}_\delta[h]}{2}}[\psi].$$

Proof of Proposition 1: We take C to be $\mathrm{Tribes}_d \circ \text{Rec-Maj-}3_\ell^{\otimes d2^d}$. Since Rec-Maj-$3_\ell$ is balanced, by Lemma 4 we have

$$\mathrm{NoiseStab}_\delta[C] = \mathrm{NoiseStab}_{\frac{1}{2} - \frac{\mathrm{NoiseStab}_\delta[\text{Rec-Maj-}3_\ell]}{2}}[\mathrm{Tribes}_d].$$

Setting $\delta = \Theta(1/\sqrt{k})$ and recalling that $3^\ell \le k^6$, we have $\ell \ge \log_{1.1}(1/\delta)$ so we may apply Lemma 2 to obtain

$$\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[\text{Rec-Maj-}3_\ell] \le \Theta((\sqrt{k})^{1.1})(k^6)^{-.15} = O(k^{-.35}).$$

Since $O(k^{-.35}) \le O(1/d)$, we may apply Lemma 3 with the previous inequalities to obtain

$$\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[C] \le O\Big(\frac{1}{2^d}\Big).$$

The bound (8) follows from some easy rearrangement of the bounds on k, m, d and ℓ. It is easy to see that C can be computed by monotone circuits of depth $O(\ell) = O(\log m)$ and size $\mathrm{poly}(m)$, and the proposition is proved. □

2.4 Nearly Optimal Hardness of Learning Polynomial-Size Monotone Circuits

Given a value of k, let $m = 3^\ell d 2^d$ for ℓ, d as in Proposition 1. Let $\mathcal{G}$ be a pseudorandom family of functions $\{g : \{0,1\}^k \to \{0,1\}\}$ secure against poly(n)-time adversaries, where $n = mk$. Since we have $k = \omega(\log n)$, we may apply Corollary 1 with the combining function from Proposition 1 and conclude that the class $\mathcal{C} = \{C \circ \mathsf{slice}(g)^{\otimes m} \mid g \in \mathcal{G}\}$ is hard to learn to accuracy

$$\frac{1}{2} + O\Big(\frac{k^3\sqrt{\log m}}{\sqrt{m}}\Big) + o(1/n) \leq \frac{1}{2} + O\Big(\frac{k^{3.5}\sqrt{\log n}}{\sqrt{n}}\Big). \tag{9}$$

We claim that in fact the functions in $\mathcal{C}$ can be computed by poly(n)-size monotone circuits. This follows from a result of Berkowitz [Ber82] which states that if a k-variable slice function is computed by a AND/OR/NOT circuit of size s and depth d, then it is also computed by a monotone AND/OR/MAJ circuit of size $O(s + k)$ and depth $d + 1$. Combining these monotone circuits for $\mathsf{slice}(g)$ with the monotone circuit for C, we obtain a poly(n)-size monotone circuit for each function in $\mathcal{C}$.

By making different assumptions on the hardness of pseudorandom function families (in fact, the assumptions may be made about one-way functions–a more formal treatment is given in the full version), we obtain quantitative relationships between k (the input length for the pseudorandom functions) and n (the running time of the adversaries against which they are secure), and thus different quantitative hardness results from (9) above:

Theorem 2. *Suppose that standard one-way functions exist. Then for any constant $\epsilon > 0$ there is a class $\mathcal{C}$ of poly(n)-size monotone circuits that is hard to learn to accuracy $\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$.*

Proof. If poly(n)-hard one-way functions exist then for any arbitrarily small constant c there exists a pseudorandom family of functions from $\{0,1\}^k \to \{0,1\}$ with $k = n^c$; this corresponds to taking $d = C \log k$ for C a large constant in Proposition 1. The claimed bound on (9) easily follows. (We note that while not every n is of the required form $mk = 3^\ell d 2^d k$, it is not difficult to see that this and our subsequent theorems hold for all (sufficiently large) input lengths n by padding the hard-to-learn functions.) □

Theorem 3. *Suppose that subexponentially hard (2^{n^α} for some fixed $\alpha > 0$) one-way functions exist. Then there is a class $\mathcal{C}$ of poly(n)-size monotone circuits that is hard to learn to accuracy $\frac{1}{2} + \frac{\text{polylog}(n)}{n^{1/2}}$.*

Proof. As above, but now we take $k = \log^C n$ for some sufficiently large constant C (*i.e.*, $d = c \log k$ for a small constant c). □

References

[AHM+06] Allender, E., Hellerstein, L., McCabe, P., Pitassi, T., Saks, M.: Minimizing DNF Formulas and AC^0_d Circuits Given a Truth Table. In: CCC, pp. 237–251 (2006)

[BBL98] Blum, A., Burch, C., Langford, J.: On learning monotone boolean functions. In: 39th FOCS, pp. 408–415 (1998)

[Ber82] Berkowitz, S.J.: On some relationships between monotone and non-monotone circuit complexity. Technical Report, University of Toronto (1982)

[BFKL93] Blum, A., Furst, M., Kearns, M., Lipton, R.: Cryptographic Primitives Based on Hard Learning Problems. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (1993)

[BT96] Bshouty, N., Tamon, C.: On the Fourier spectrum of monotone functions. Journal of the ACM 43(4), 747–770 (1996)

[HVV06] Healy, A., Vadhan, S., Viola, E.: Using Nondeterminism to Amplify Hardness. SIAM Journal on Computing 35(4), 903–931 (2006)

[Kha93] Kharitonov, M.: Cryptographic hardness of distribution-specific learning. In: 25th STOC, pp. 372–381 (1993)

[Kha95] Kharitonov, M.: Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution. JCSS 50, 600–610 (1995)

[KKL88] Kahn, J., Kalai, G., Linial, N.: The influence of variables on boolean functions. In: 29th FOCS, pp. 68–80 (1988)

[KLV94] Kearns, M.J., Li, M., Valiant, L.G.: Learning boolean formulas. J. ACM 41(6), 1298–1328 (1994)

[KOS04] Klivans, A., O'Donnell, R., Servedio, R.: Learning intersections and thresholds of halfspaces. JCSS 68(4), 808–840 (2004)

[LMN93] Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, Fourier transform and learnability. Journal of the ACM 40(3), 607–620 (1993)

[Man94] Mansour, Y.: Learning Boolean functions via the Fourier transform, pp. 391–424. Kluwer Academic Publishers, Dordrecht (1994)

[MO03] Mossel, E., O'Donnell, R.: On the noise sensitivity of monotone functions. Random Struct. Algorithms 23(3), 333–350 (2003)

[MOS04] Mossel, E., O'Donnell, R., Servedio, R.: Learning functions of k relevant variables. J. Comput. & Syst. Sci. 69(3), 421–434 (2004)

[Nep70] Nepomnjaščiĭ, V.A.: Rudimentary predicates and Turing calculations. Math Dokl. 11, 1462–1465 (1970)

[NR04] Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. Journal of the ACM 51(2), 231–262 (2004)

[O'D04] O'Donnell, R.: Hardness amplification within NP. JCSS 69(1), 68–94 (2004)

[OS07] O'Donnell, R., Servedio, R.: Learning monotone decision trees in polynomial time. SIAM Journal on Computing 37(3), 827–844 (2007)

[Raz85] Razborov, A.: Lower bounds on the monotone network complexity of the logical permanent. Mat. Zametki 37, 887–900 (1985)

[Ser04] Servedio, R.: On learning monotone DNF under product distributions. Information and Computation 193(1), 57–74 (2004)

[Val84] Valiant, L.: A theory of the learnable. CACM 27(11), 1134–1142 (1984)

[Ver90] Verbeurgt, K.: Learning DNF under the uniform distribution in quasi-polynomial time. In: 3rd COLT, pp. 314–326 (1990)

On Berge Multiplication for Monotone Boolean Dualization*

Endre Boros[1], Khaled Elbassioni[2], and Kazuhisa Makino[3]

[1] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003
boros@rutcor.rutgers.edu
[2] Max-Planck-Institut für Informatik, 66111 Saarbrücken, Germany
elbassio@mpi-sb.mpg.de
[3] Department of Mathematical Informatics, University of Tokyo, Tokyo, 113-8656, Japan
makino@mist.i.u-tokyo.ac.jp

Abstract. Given the prime CNF representation ϕ of a monotone Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$, the dualization problem calls for finding the corresponding prime DNF representation ψ of f. A very simple method (called *Berge multiplication* [2, Page 52–53]) works by multiplying out the clauses of ϕ from left to right in some order, simplifying whenever possible using *the absorption law*. We show that for any monotone CNF ϕ, Berge multiplication can be done in subexponential time, and for many interesting subclasses of monotone CNF's such as CNF's with bounded size, bounded degree, bounded intersection, bounded conformality, and read-once formula, it can be done in polynomial or quasi-polynomial time.

1 Introduction

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. A function is called *monotone* (also called *positive*) if for every pair of vectors $x, y \in \{0,1\}^n$, $x \leq y$ (i.e., $x_i \leq y_i$ for all i) always implies $f(x) \leq f(y)$. Any monotone function f has a unique *prime conjunctive normal form* (*CNF*) expression

$$\phi(x) = \bigwedge_{C \in \mathcal{C}} \Big(\bigvee_{i \in C} x_i \Big), \tag{1}$$

where $\mathcal{C}$ is *Sperner* (i.e., $I \not\subseteq J$ holds for $I, J \in \mathcal{C}$ with $I \neq J$). It is well-known that $\mathcal{C}$ corresponds to the set of all *prime implicates* of f. The well-known *monotone Boolean dualization problem* is to find the corresponding *prime disjunctive normal form* (DNF) representation of f:

$$\psi(x) = \bigvee_{D \in \mathcal{D}} \Big(\bigwedge_{i \in D} x_i \Big), \tag{2}$$

* The first author is thankful for the partial support by NSF (CBET-0735910). The second and third authors thank the partial support by DIMACS, the National Science Foundation's Center for Discrete Mathematics and Theoretical Computer Science.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 48–59, 2008.

where $\mathcal{D}$ is Sperner and corresponds to the set of all *prime implicants* of f. Equivalently, the problem is to compute, for an explicitly given hypergraph $\mathcal{C} \subseteq 2^V$, the transversal hypergraph $\mathcal{D}$, consisting of all minimal transversals D of $\mathcal{H}$ (i.e., all minimal subsets $D \subseteq V$ such that $D \cap C \neq \emptyset$ for all $C \in \mathcal{C}$). This problem has received considerable attention in the literature (see e.g., [9,11,23,25]), since it is known to be polynomially or quasi-polynomially equivalent with many problems in various areas, such as artificial intelligence (e.g., [9,17]), database theory (e.g., [24]), distributed systems (e.g., [16]), machine learning and data mining (e.g., [6,15]), mathematical programming (e.g., [4,19]), matroid theory (e.g., [21]), and reliability theory (e.g., [26]).

While the size of the output DNF ψ can be exponential in the size of ϕ, it is open (for more than 25 years now, e.g., [3,10,22,23,25]) whether ψ can be computed in *output-polynomial* (or *polynomial total*) *time*, i.e., in time polynomial in the combined size of ϕ *and* ψ. Any such algorithm for the monotone dualization problem would significantly advance the state of the art of the problems in the application areas mentioned above. This is witnessed by the fact that these problems are cited in a rapidly growing body of literature and have been referenced in various survey papers and complexity theory retrospectives, e.g. [10,12,23,25].

In 1996, Fredman and Khachiyan [14] established a remarkable result that the monotone dualization problem can be solved in quasi-polynomial time $O(nN) + N^{o(\log N)}$, where $N = |\phi| + |\psi|$, thus putting the problem somewhere between polynomiality and NP-completeness. They achieved this by presenting a quasi-polynomial time algorithm for the decision-version of the problem: given two monotone Boolean formulae ϕ and ψ in CNF and DNF respectively, is $\phi \equiv \psi$? Furthermore, for several special classes of monotone formulae ϕ, the problem is known to be solvable in polynomial time, e.g., when every clause has bounded-size [5,9], when every variable has bounded degree [7,11], when clauses have bounded intersection-size [20], for read-once formulae [8], etc.

A very simple method to solve the monotone dualization problem, called *left-to-right multiplication*, or sometimes *Berge multiplication* (see [2, Page 52–53]), works by traversing the clauses of the input CNF in some order, say $j = 1, \ldots, m = |\phi|$, multiplying out clause C_j with the DNF obtained for $C_1 \wedge \ldots \wedge C_{j-1}$, and simplifying the DNF's using *the absorption law* (i.e., the identity $x \vee (x \wedge y) = x$ for all Boolean x, y) whenever possible (see Figure 1). We remark that many practical algorithms for monotone dualization are obtained from the left-to-right multiplication by putting several heuristic ideas (see e.g., [1,18]).

It is not difficult to come up with examples for which this method exhibits an *exponential blow-up* in the input-output size, e.g., the intermediate DNFs are exponential in the input size, while the final output is polynomially-bounded. Consider for instance, a CNF $\phi = \bigwedge_{1 \leq i,j \leq n} (x_i \vee y_j)$ on the set of $2n$ variables $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$. One can easily check that the corresponding prime DNF is $(x_1 \wedge \ldots \wedge x_n) \vee (y_1 \wedge \ldots \wedge y_n)$. On the other hand, if we start by multiplying the clauses $(x_1 \vee y_1), \ldots, (x_n \vee y_n)$, then we get 2^n clauses, which will be canceled out later in the process. More interestingly, Takata [27] gave an example for which the left-to-right multiplication method exhibits a *superpolynomial*

blow-up, under *any ordering* of the clauses of the input CNF. He also suggested a generalization of the multiplication method which was shown to be output quasi-polynomial in [13].

In view of this result, it is natural to ask whether there is an example where an *exponential* blow-up is unavoidable under any ordering of the clauses. In this paper, we answer this question in the *negative*. Namely, we show that, for any monotone CNF, there is an ordering of the clauses such that the size of the intermediate DNF at any stage of the left-to-right multiplication is bounded by a *subexponential* in the input-output size. Furthermore, we show that, for several interesting well-known classes of monotone CNF formulae such as read-once, bounded degree, bounded clause-size, etc., there are orderings of the clauses that guarantee (quasi-)polynomial blow-up's. The only result we are aware of this type is the one for bounded degree formulae [7].

To formally state our results, let us consider a monotone CNF $\phi = C_1 \wedge \cdots \wedge C_m$, and let $\pi \in \mathbb{S}_m$ be a permutation of the clauses, where $\mathbb{S}_m$ denotes the set of all permutations of m elements. For $j = 1, \ldots, m$, let ϕ_j^π denote the CNF having the first j clauses in ϕ according to the ordering π, i.e., $\phi_j^\pi \stackrel{\text{def}}{=} \bigwedge_{l=1}^{j} C_{\pi(l)}$. For a CNF (resp., DNF) φ, we denote by $|\varphi|$ the number of clauses (resp., terms). Denote by $\nu(\pi)$ the size of a maximum intermediate DNF produced during the left-right multiplication, i.e., $\nu(\pi) \stackrel{\text{def}}{=} \max_{1 \le j \le m} |(\phi_j^\pi)^*|$, where, for a monotone CNF φ, φ^* denotes the prime DNF corresponding to φ. Then we have the following theorem.

Theorem 1. *Let ϕ be a prime monotone CNF with n variables and m clauses. Then*

(i) *If ϕ has bounded clause-size, bounded degree, bounded intersection-size, or bounded degeneracy, then there exists a permutation π of the clauses in ϕ such that $\nu(\pi) = |\phi^*|^{O(1)}$.*
(ii) *If ϕ has bounded conformality or read-once representation, then there exists a permutation π of the clauses in ϕ such that $\nu(\pi) = |\phi^*|^{O(\log m)}$.*
(iii) *For any prime monotone CNF ϕ, there exists a permutation π of the clauses in ϕ such that $\nu(\pi) \le n^{\sqrt{n}+1}|\phi^*|^{\sqrt{n}\ln m}$.*

Furthermore, such permutations can be found in polynomial time in n and m.

The formal definitions of the types of CNF's stated in (i) and (ii) will be given in Sections 3 and 4. We remark that there is a prime monotone CNF ϕ with read-once representation such that $\nu(\pi) = |\phi^*|^{\Omega(\log\log m)}$ holds for any permutation π of clauses in ϕ [27].

It is easy to see that, for a given permutation π, the left-to-right multiplication takes polynomial time in n, m, and $\nu(\pi)$, where more careful analysis can be found in Section 2. Thus, the theorem above gives an upper bound on the running time of the left-to-right multiplication procedure.

Corollary 1. *The following three statements hold.*

(i) *If ϕ is a prime monotone CNF that has bounded clause-size, bounded degree, bounded intersection-size, or bounded degeneracy, then the left-to-right multiplication for ϕ can be done in output-polynomial time.*
(ii) *If ϕ is a prime monotone CNF that has bounded conformality or read-once representation, then the left-to-right multiplication for ϕ can be done in output-quasi-polynomial time.*
(iii) *For any prime monotone CNF, the left-to-right multiplication can be done in output-subexponential time.*

The rest of the paper is organized as follows. In the next section, we state our notation and present several properties of left-to-right multiplication used in the following sections. In Section 3, we show that the left-to-right multiplication based on reverse lexicographic ordering of clauses is an efficient way of dualizing monotone CNF's with bounded clause-size, bounded degree, or bounded clause-intersections. In Section 4, we present a more general technique for ordering the clauses of an input CNF, and derive from it the above stated results for general monotone CNF's and for some special classes.

2 Preliminaries

Let $\phi = \phi(x_1, \ldots, x_n)$ be a formula. We denote by $V(\phi)$ the set of variables in ϕ. For convenience, if ϕ is a monotone CNF (resp. DNF) and C is a clause (resp., term) in ϕ, we shall write $C \in \phi$, and view C also as the index set $C \subseteq V(\phi)$ of the variables that it contains. This way, one can also view ϕ as a subfamily of $2^{V(\phi)}$, each of which represents a clause (resp., term), and thus use ordinary set operations on it. A monotone CNF ϕ is *prime* if for all $C, C' \in \phi$, $C \subseteq C'$ implies that $C = C'$ (see [11]). If ϕ is a monotone CNF formula, we denote by ϕ^* a prime DNF formula representing the same monotone Boolean function as ϕ (see [2]). As mentioned in the Introduction, any monotone function has a unique prime CNF (DNF) expression. In this paper we consider the following problem:

Problem Monotone Boolean Dualization
Input: The prime CNF ϕ of a monotone Boolean function.
Output: The prime DNF ϕ^*.

For a given monotone CNF ϕ we use the notation: $n = |V(\phi)|$ and $m = |\phi|$.

The left-to-right multiplication given in Figure 1 is one of the simplest procedures to solve the Monotone Boolean Dualization. Here function Min$(\cdot)$ takes the conjunction of a monotone prime DNF ρ and a monotone clause C, and returns a prime monotone DNF ρ' that is equivalent to $\rho \wedge C$. It is not difficult to see that for all $j = 1, \ldots, m$, ψ_j in Figure 1 satisfies $\psi_j = (\phi_j^\pi)^*$, and hence the left-to-right multiplication correctly computes ϕ^* $(= \psi_m)$.

Proposition 1. *For a prime monotone CNF ϕ and a permutation $\pi \in \mathbb{S}_m$,* LR-Mult(ϕ, π) *can be implemented to run in $O(nm\nu(\pi)\min\{m, \nu(\pi)\})$ time.*

Procedure LR-Mult(ϕ, π)**:**
Input: The prime CNF $\phi = \wedge_{j=1}^{m} C_j$ of a monotone Boolean function and a permutation $\pi \in \mathbb{S}_m$.
Output: The prime DNF ϕ^*.

$\psi_0 := \emptyset$
for $j = 1, \ldots, m$
  $\psi_j := \mathrm{Min}(\psi_{j-1} \wedge C_{\pi(j)})$
return ψ_m

Fig. 1. The left-to-right multiplication

For a monotone CNF ϕ and $i \in V(\phi)$, we denote by $\phi_{(i)}$ the subformula of ϕ consisting of all clauses containing variable x_i, and let $\deg_\phi(i) = |\phi_{(i)}|$ be the degree of x_i in ϕ. For a subset $S \subseteq V(\phi)$ of variables, denote by ϕ_S the CNF formula obtained form ϕ by fixing $x_i = 1$ for all $i \in V(\phi) \setminus S$. Equivalently, $\phi_S = \bigwedge_{C \in \phi:\, C \subseteq S} \big(\bigvee_{i \in C} x_i \big)$. Thus we call ϕ_S the *projection* of ϕ on S. The reason that we are interested in projections is the following.

Proposition 2 ([22]). *Let ϕ be a monotone CNF. For any $S \subseteq V(\phi)$, we have $|\phi^*_S| \leq |\phi^*|$.*

Clearly, we have $|(\phi \wedge \phi')^*| \leq |\phi^*||(\phi')^*|$ for any CNF's ϕ and ϕ', and thus the above proposition implies the following claims.

Lemma 1. *Let ϕ ba a monotone CNF. If $\phi' = \phi_{S_1} \wedge \phi_{S_2} \wedge \cdots \wedge \phi_{S_k}$ for some subsets $S_\ell \subseteq V(\phi)$, $\ell = 1, .., k$, then we have $|(\phi')^*| \leq |\phi^*|^k$.*

Lemma 2. *Let $\phi = \bigwedge_{j=1}^{m} C_j$ be a monotone CNF, and let $\pi \in \mathbb{S}_m$ be a permutation of the clauses of ϕ such that for every $j = 1, \ldots, m$ there exists some subsets $S_{j,\ell} \subseteq V$, $\ell = 1, \ldots, k_j$ such that*

$$\phi_j^\pi = \phi_{S_{j,1}} \wedge \phi_{S_{j,2}} \wedge \cdots \wedge \phi_{S_{j,k_j}} \tag{3}$$

holds. Let $k = \max\{k_1, \ldots, k_m\}$. Then $\nu(\pi) \leq |\phi^*|^k$, and thus LR-Mult(ϕ, π) *computes ϕ^* in $O(nm|\phi^*|^k \min\{m, |\phi^*|^k\})$ time.*

In the following sections we show various techniques to find such an ordering π of ϕ which guarantees a *small* k in the above statement.

3 Reverse Lexicographic Orderings

Assume that $V = V(\phi)\,(= \{1, 2, \ldots, n\})$ and for subsets $A, B \subseteq V$ let us denote by $L = L(A, B)$ their *last common elements*, i.e., L is the maximal subset $L \subseteq A \cap B$ such that for all $i_1 \in (A \cup B) \setminus L$ and $i_2 \in L$ we have $i_1 < i_2$. We say that A *precedes* B if $\max(A \setminus L(A, B)) < \max(B \setminus L(A, B))$. Finally, we say that $\{C_1, C_2, \ldots, C_m\}$ is the *reverse lexicographic labeling* of ϕ (or that the clauses of ϕ are in *reverse lexicographic order*), if C_{j_1} precedes C_{j_2} for all $1 \leq j_1 < j_2 \leq m$.

Clearly, the reverse lexicographic order of the clauses is determined uniquely by the ordering of the *variable* indices in V. To denote this dependence, let us use $L_\sigma(A, B)$ for the last common elements of A and B, when V is ordered by a permutation $\sigma \in \mathbb{S}_n$, and call the corresponding ordering of the clauses of ϕ the *σ-reverse lexicographic order* of ϕ, denoted by π_σ.

Given a permutation $\sigma \in \mathbb{S}_n$, let $\mu_\sigma(\phi) \stackrel{\text{def}}{=} \max_{1 \le j < m} |L_\sigma(C_{\pi_\sigma(j)}, C_{\pi_\sigma(j+1)})|$. Note that the value of $\mu_\sigma(\phi)$ can be computed in $O(nm)$ time.

To simplify our notations, let us assume that $\sigma = (1, \ldots, n)$ and $\pi_\sigma = (1, \ldots m)$, i.e., $\{C_1, \ldots, C_m\}$ is the σ-reverse lexicographic labeling of ϕ. Given an index $1 \le j < m$, let us introduce $L_j = L_\sigma(C_j, C_{j+1})$, $\lambda = |L_j|$, and $\phi_j = \phi_j^{\pi_\sigma}$ $(= C_1 \wedge \cdots \wedge C_j)$. By definition, we have $\lambda \le \mu_\sigma(\phi)$. Furthermore, let $L_j = \{i_1, i_2, \ldots, i_\lambda\}$, where $i_1 < \cdots < i_\lambda$, and i_0 is the largest element in $C_{j+1} \setminus L_j$. Clearly, $\{i_0, \ldots, i_\lambda\}$ are the last $\lambda + 1$ elements of L_{j+1}.

Let $[i] = \{1, \ldots, i\}$ and consider the following subsets of V:

$$S_\ell = [i_\ell - 1] \cup \bigcup_{k=\ell+1}^{\lambda} \{i_k\} \quad \text{for all} \quad \ell = 0, \ldots, \lambda. \tag{4}$$

Lemma 3. *For all $1 \le j < m$ we have $\phi_j = \phi_{S_0} \wedge \cdots \wedge \phi_{S_\lambda}$.*

Lemma 4. *For every $j = 1, 2, \ldots, m$ we have $k\,(\le 1 + \mu_\sigma(\phi))$ subsets $S_{j,1}$, $S_{j,2}$, $\ldots$, $S_{j,k}$ of V such that (3) holds.*

Theorem 2. *For every CNF ϕ and permutation $\sigma \in \mathbb{S}_n$, $|\nu(\pi_\sigma)| \le |\phi^*|^{1+\mu_\sigma(\phi)}$. Thus* LR-Mult *computes ϕ^* in $O(nm|\phi^*|^{1+\mu_\sigma(\phi)} \min\{m, |\phi^*|^{1+\mu_\sigma(\phi)}\})$ time.*

Proof. The theorem follows from Proposition 1 and Lemma 4. □

We shall show in the next subsections that even with $\sigma = (1, 2, \ldots, n)$, the class of CNF's ϕ for which $\mu_\sigma(\phi)$ is a fixed constant includes several well-known classes, proving that LR-Mult provides an efficient dualization for all these cases. Before turning to special types of CNF's, let us observe a useful property of the sets introduced in (4).

Lemma 5. *For every $\ell = 0, \ldots, \lambda$, the sets in $(\phi_{S_0} \cup \phi_{S_1} \cup \cdots \cup \phi_{S_\ell}) \setminus (\phi_{S_{\ell+1}} \cup \cdots \cup \phi_{S_\lambda})$ all contain $L = \{i_{\ell+1}, \ldots, i_\lambda\}$ as their last elements according to π_σ.*

Unless otherwise stated, let us assume in the sequel that $\sigma = (1, 2, \ldots, n)$ and eliminate it from our notations, and let $\pi = \pi_\sigma$.

3.1 Degenerate CNF's

Given a CNF ϕ, let us denote by $\Delta(\phi) = \max_{i \in V} \deg_\phi(i)$ the maximum degree of a variable in ϕ. For a given k, we say that ϕ has *bounded occurrences* if $\Delta(\phi) \le k$. More generally, a CNF ϕ is said to be *k-degenerate* [11], for an integer $k \in \mathbb{Z}_+$, if for any $S \subseteq V$, $\min_{i \in S} \deg_{\phi_S}(i) \le k$. Equivalently, ϕ is k-degenerate if and only if there exists a permutation $\sigma \in \mathbb{S}_n$ of the variables such that, for all

$i = 1, \ldots, n$, $\deg_{\phi_{[i]}}(i) \leq k$. Here we note that such a permutation can be computed in $O(nm)$ time [11]. This class includes for instance formulae of bounded occurrences, bounded hypertree-width; see [11]. The following statement thus generalizes the results of [7].

Theorem 3. *If ϕ is a k-degenerate CNF and σ is a permutation of variables such that $\deg_{\phi_{[i]}}(i) \leq k$ for all $i = 1, \ldots, n$, then we have $\nu(\pi_\sigma) \leq |\phi^*| n^{k-1}$, and thus* LR-Mult *computes ϕ^* in $O(n^k m |\phi^*| \min\{m, n^{k-1}|\phi^*|\})$ time.*

Proof. Assume without loss of generality that $\sigma = (1, \ldots, n)$ is a permutation of variables such that $\deg_{\phi_{[i]}}(i) \leq k$ for all $i = 1, \ldots, n$. Let j be an integer in $[m-1]$. If $L_j = \emptyset$, then $\phi_j = \phi_{S_0}$ and hence $|(\phi_j)^*| \leq |\phi^*|$. On the other hand, if $L_j \neq \emptyset$, then by Lemma 5, the clauses in $\left(\phi_{S_0} \cup \phi_{S_1} \cup \cdots \cup \phi_{S_{\lambda-1}}\right) \setminus \phi_{S_\lambda}$ all contain i_λ as their last element, and we cannot have more than $k-1$ such clauses, since $\deg_{\phi_{[i_\lambda]}}(i_\lambda) \leq k$ and $i_\lambda \in C_{j+1}$. This implies $|(\phi_j)^*| \leq n^{k-1}|(\phi_{S_\lambda})^*| \leq n^{k-1}|\phi^*|$. □

We remark that for CNFs with bounded occurrences, any ordering σ of variables produces a good left-to-right multiplication.

3.2 CNF's with Bounded (k, r)-Intersections

Given a CNF ϕ, let $D_1(\phi)$ and $D_2(\phi)$ respectively denote the *dimension* and *intersection size* of ϕ, i.e., $D_1(\phi) = \max_{C \in \phi} |C|$ and $D_2(\phi) = \max_{\substack{C, C' \in \phi \\ C \neq C'}} |C \cap C'|$. For a given r we say that ϕ has *bounded dimension* and *intersections* if $D_1(\phi) \leq r$ and $D_2(\phi) \leq r$, respectively.

We generalize classes of monotone CNF's with bounded occurrences, bounded dimension, and bounded intersection as follows. Let $k \geq 1$ and $r \geq 0$ be integers. We denote by $\mathbb{A}(k, r)$ the class of of monotone CNF formulae with (k, r)-*bounded intersections* [20]: $\phi \in \mathbb{A}(k, r)$ if for any k distinct clauses of ϕ, $C_{j_1}, \ldots, C_{j_k}$, we have $|\bigcap_{\ell=1}^{k} C_{j_\ell}| \leq r$. Note that $\Delta(\phi) \leq k$ iff $\phi \in \mathbb{A}(k+1, 0)$, $D_1(\phi) \leq r$ iff $\phi \in \mathbb{A}(1, r)$, and $D_2(\phi) \leq r$ iff $\phi \in \mathbb{A}(2, r)$, and hence, the class $\mathbb{A}(k, r)$ contains the bounded size, bounded degree, and bounded intersections CNF's as subclasses.

Lemma 6. *Let $\phi \in \mathbb{A}(k, r)$ and let σ be an arbitrary permutation of variables. Then, for any index j with $1 \leq j < m$,*

$$|(\phi_j^{\pi_\sigma})^*| \leq \begin{cases} |\phi^*|^r & \text{if } \lambda < r \\ |\phi^*|^{r+1} & \text{if } \lambda = r \\ n^{k-2}|\phi^*|^{r+1} & \text{if } \lambda > r, \end{cases}$$

where $\lambda = |L_j|$ $(= |L_\sigma(C_{\pi_\sigma(j)}, C_{\pi_\sigma(j+1)})|)$.

Lemma 7. *Let $\phi \in \mathbb{A}(k, r)$ and let σ be an arbitrary permutation of variables. Then, for any index j with $1 \leq j < m$, $\lambda < r$ holds for $k = 1$, and $\lambda \leq r$ holds for $k = 2$, where $\lambda = |L_j|$ $(= |L_\sigma(C_{\pi_\sigma(j)}, C_{\pi_\sigma(j+1)})|)$.*

From Lemmas 6 and 7, we have the following theorem.

Theorem 4. *Let $\phi \in \mathbb{A}(k,r)$ and let $\sigma \in \mathbb{S}_n$ be an arbitrary permutation. Then*

$$\nu(\pi_\sigma) \;\le\; \begin{cases} |\phi^*|^r & \text{if } k=1 \\ |\phi^*|^{r+1} & \text{if } k=2 \\ n^{k-2}|\phi^*|^{r+1} & \text{if } k \ge 3, \end{cases}$$

and thus LR-Mult *computes ϕ^* in*

$$\begin{array}{ll} O(nm|\phi^*|^r \min\{m,|\phi^*|^r\}) \text{ time} & \text{if } k=1, \\ O(nm|\phi^*|^{r+1} \min\{m,|\phi^*|^{r+1}\}) \text{ time} & \text{if } k=2, \text{ and} \\ O(n^{k-1}m|\phi^*|^{r+1} \min\{m,n^{k-2}|\phi^*|^{r+1}\}) \text{ time} & \text{if } k \ge 3. \end{array}$$

As a corollary, for prime monotone CNFs ϕ with bounded degree $\Delta(\phi) \le k$, LR-Mult computes ϕ^* in $O(n^k m|\phi^*| \min\{m, n^{k-1}|\phi^*|\})$ time, which matches Theorem 3.

4 Multiplication-Tree Orderings

Given a monotone CNF ϕ, we build a binary tree $\mathbf{T}$, which we call a *multiplication* tree, each node v of which is associated with a monotone CNF $\phi(v)$ as follows:

(I) if v is a leaf then $\phi(v)$ is an individual clause of ϕ and every clause of ϕ appears uniquely in a leaf of $\mathbf{T}$;
(II) if v is an internal node, then it has two children u and w such that $\phi(v) = \phi(u) \wedge \phi(w)$, i.e., $\phi(v)$ is the conjunction of the subset of clauses of ϕ appearing in the leaves of the subtree of $\mathbf{T}$ rooted at v.

For a binary multiplication tree $\mathbf{T}$, we fix a planar embedding of $\mathbf{T}$ and let $\pi_{\mathbf{T}}$ be the order of clauses defined by the *left-to-right traversal* of the leaves of $\mathbf{T}$. Namely, $\pi_{\mathbf{T}}$ is obtained in the depth-first search from the root of $\mathbf{T}$ in which at each node, the left child is visited before the right one.

Note that any ordering π of clauses in ϕ can be represented by $\pi = \pi_{\mathbf{T}}$ for some multiplication tree. Denote by $\mathcal{N}(\mathbf{T})$ the set of nodes of the tree $\mathbf{T}$. For a node $v \in \mathcal{N}(\mathbf{T})$, let ϕ^v be the subformula of ϕ obtained by the left-to-right traversal of the leaves of $\mathbf{T}$ upto the right-most leaf of the subtree rooted at v: $\phi^v = \phi_r^{\pi_{\mathbf{T}}}$ $(= \bigwedge_{i=1}^{r} C_{\pi_{\mathbf{T}}(i)})$, where r is the number of leaves, counted from the left-most leaf of $\mathbf{T}$, up to the right-most leaf of the subtree rooted at v. In what follows we denote by $\nu(\mathbf{T})$ the size of a maximum intermediate DNF produced during LR-Mult$(\phi, \pi_{\mathbf{T}})$: $\nu(\mathbf{T}) = \nu(\pi_{\mathbf{T}}) = \max_{v \in \mathcal{N}(\mathbf{T})}\{|(\phi^v)^*|\}$. We denote respectively by $p(v)$, left(v), and right(v), the parent, left and right children of node $v \in \mathcal{N}(\mathbf{T})$.

A binary multiplication tree $\mathbf{T}$ is called *proper* if for every $v \in \mathcal{N}(\mathbf{T})$, the set $\phi(\text{left}(v))$ is a projection of $\phi(v)$, i.e., there exists a set $S \subseteq V(\phi)$ such that $\phi(v)_S = \phi(\text{left}(v))$. Call a node $v \in \mathcal{N}(\mathbf{T})$ an L-node (resp., R-node) if v is the left (resp., right) child of its parent in $\mathbf{T}$. Define the *right-depth* of $v \in \mathcal{N}(\mathbf{T})$, denoted by $\mathrm{d}(v)$, to be one plus the number of R-nodes in the path from the root $r(\mathbf{T})$ of $\mathbf{T}$ to v, and define the right-depth of $\mathbf{T}$, by $d(\mathbf{T}) = \max_{v \in \mathcal{N}(\mathbf{T})} \mathrm{d}(v)$.

Procedure Construct-Tree-A(ϕ, v)**:**
Input: A prime monotone CNF ϕ and a node v of the tree.
Output: A proper binary multiplication tree for ϕ rooted at v.

$\phi(v) := \phi$
if $|\phi(v)| > 1$
  Construct the left and right children left(v) and right(v) of v
  $i :=$ argmin$\{\deg_\phi(i) : \ i \in V(\phi)\}$
  Call Construct-Tree-A$(\phi_{V(\phi)\setminus\{i\}}, \text{left}(v))$
  Call Construct-Tree-A$(\phi_{(i)}, \text{right}(v))$

Fig. 2. Procedure **Construct-Tree-A** to construct a proper multiplication tree for ϕ

Theorem 5. *Let ϕ be a monotone CNF. If $\mathbf{T}$ be a proper binary multiplication tree of ϕ, then we have $\nu(\mathbf{T}) \leq |\psi^*|^{d(\mathbf{T})}$.*

4.1 Quasi-polynomial Cases

Conformal CNF's. There are several equivalent definitions for conformal CNF's (see [2, Page 90]). The most convenient for our purposes is the following: For an integer $k \geq 1$, a monotone CNF ϕ is called *k-conformal* if for every subset of variables $X \subseteq V(\phi)$, X is contained in a clause of ϕ whenever each subset of X of cardinality at most k is contained in a clause of ϕ. One can easily verify that $\phi \in \mathbb{A}(k, r)$ implies that ϕ is $(k + r)$-conformal. Thus the class of CNF's with bounded conformality includes as a special case the CNF's with bounded intersections considered in the previous section.

Although the prime DNF representation of a k-conformal CNF can be computed in polynomial time if k is constant [20], we can only show a quasi-polynomial bound for the left-to-right multiplication.

Lemma 8. *Let $\phi = \bigwedge_{j=1}^{m} C_j$ be a k-conformal prime monotone CNF. Then there is a proper binary multiplication tree $\mathbf{T}$ with $\mathrm{d}(\mathbf{T}) \leq k \ln m + 1$.*

Proof. We use the procedure in Figure 2, combined with the following claim.

Claim (C1). Let $\phi' \subseteq \phi$ be a subformula of ϕ such that $|\phi'| > 1$. Then there exists an infrequent variable $i \in V(\phi')$: $|\phi'_{(i)}| \leq (1 - \frac{1}{k})|\phi'|$.

We now argue that the right-depth of $\mathbf{T}$ is logarithmic. Consider a node $v \in \mathcal{N}(\mathbf{T})$, and let $u_1, \ldots, u_h$ be the R-nodes in the path from $\mathbf{r}(\mathbf{T})$ to v, ordered by increasing distance from $\mathbf{r}(\mathbf{T})$. Then by the selection of the branching variable, $|\phi(u_\ell)| \leq (1 - 1/k)|\phi(p(u_\ell))|$ for all $\ell \in [h]$. It follows that $|\phi(u_1)| \leq (1 - 1/k)|\phi| = (1 - 1/k)m$ and $|\phi(u_{\ell+1})| \leq (1 - 1/k)|\phi(u_\ell)|$ for all $\ell \in [h]$, and hence $|\phi(u_h)| \leq (1 - 1/k)^h m$. Since $|\phi(u_h)| \geq 1$, we get $h \leq k \ln m$. □

Theorem 6. *Let $\phi = \bigwedge_{j=1}^{m} C_j$ be a k-conformal prime monotone CNF. Then Procedure* **Construct-Tree-A** *produces a permutation $\pi_{\mathbf{T}}$ of the clauses such that $\nu(\pi_{\mathbf{T}}) \leq |\phi^*|^{k \ln m + 1}$, and thus* LR-Mult *computes ϕ^* in $O(nm|\phi^*|^{k \ln m + 1}$ $\min\{m, |\phi^*|^{k \ln m + 1}\})$ time.*

CNF's of read-once expressions. A formula φ is called *read-once* if it can be written as an $\wedge - \vee$ formula in which every variable in $V(\varphi)$ appears exactly once. A well-known equivalent definition is that ϕ is a prime monotone CNF, which can be represented by a read-once expression, if and only if $|C \cap t| = 1$ for every clause $C \in \phi$ and every term $t \in \phi^*$.

Lemma 9. *Let $\phi = \bigwedge_{j=1}^{m} C_j$ be a prime monotone CNF with a read-once expression. Then there is a proper binary multiplication tree* $\mathbf{T}$ *with* $\mathrm{d}(\mathbf{T}) \leq \log m + 1$.

Proof. We use the following claim to construct $\mathbf{T}$ by the procedure in Figure 2.

Claim (C2). Let $\phi' \subseteq \phi$ be a subformula of ϕ such that $|\phi'| > 1$. Then there exists an infrequent variable $i \in V(\phi')$: $|\phi'_{(i)}| \leq \frac{1}{2}|\phi'|$.

The rest of the proof is the same as in Lemma 8. □

Theorem 7. *Let $\phi = \bigwedge_{j=1}^{m} C_j$ be a prime monotone CNF which can be represented by a read-once expression. Then Procedure* **Construct-Tree-A** *produces a permutation $\pi_{\mathbf{T}}$ of the clauses such that $\nu(\pi_{\mathbf{T}}) \leq |\phi^*|^{\log m+1}$, and thus* LR-Mult *computes ϕ^* in $O(nm|\phi^*|^{\log m+1} \min\{m, |\phi^*|^{\log m+1}\})$ time.*

4.2 General Monotone CNF's

In this section, we consider general monotone CNFs, and show that by use of the procedure in Figure 3, the left-to-right multiplication can always be done in subexponential time. The procedure constructs a proper binary multiplication tree for ϕ which is almost identical to the procedure in Figure 2, except that the minimum-degree variable is computed with respect to the CNF ϕ' containing only small clauses of ϕ. We begin with the following two simple lemmas.

Lemma 10. *Let ϕ be a prime monotone CNF, and let k be a positive integer with $k < n/2$. If every clause of ϕ has size at least $n - k$, then any permutation π has $\nu(\pi) \leq n^{k+1}$.*

Procedure Construct-Tree-B(ϕ, v)**:**
Input: A prime monotone CNF ϕ and a node v of the tree.
Output: A proper binary multiplication tree for ϕ rooted at v.

$\phi(v) := \phi$
if $|\phi(v)| > 1$
  Construct the left and right children left(v) and right(v) of v
  $\phi' := \bigwedge\{C \in \phi : |C| \leq |V(\phi)| - \sqrt{|V(\phi)|}\}$
  $i := \mathrm{argmin}\{\deg_{\phi'}(i) : i \in V(\phi)\}$
  Call Construct-Tree-B$(\phi_{V(\phi)\setminus\{i\}}, \mathrm{left}(v))$
  Call Construct-Tree-B$(\phi_{(i)}, \mathrm{right}(v))$

Fig. 3. Procedure **Construct-Tree-B** to construct a proper multiplication tree for ϕ

Lemma 11. *Let ϕ be a prime monotone CNF, let k be a positive integer, and let ϕ' be a subformula of ϕ. If every clause of ϕ' has size at most $n-k$, then there exists an infrequent variable $i \in V(\phi)$ with respect to ϕ': $|\phi'_{(i)}| \leq (1-\frac{k}{n})|\phi'|$.*

Let us now show that the procedure in Figure 3 produces a multiplication tree with small right-depth.

Theorem 8. *Let $\phi = \bigwedge_{j=1}^{m} C_j$ be a prime monotone CNF. Then Procedure* **Construct-Tree-B** *produces a permutation $\pi_{\mathbf{T}}$ of the clauses such that $\nu(\pi_{\mathbf{T}}) \leq n^{\sqrt{n}+1}|\phi^*|^{\sqrt{n}\ln m}$, and thus* LR-Mult *computes ϕ^* in $O(n^{\sqrt{n}+2}m|\phi^*|^{\sqrt{n}\ln m}$ $\min\{m, n^{\sqrt{n}+1}|\phi^*|^{\sqrt{n}\ln m}\})$ time.*

Proof. Consider any leaf $v \in \mathcal{N}(\mathbf{T})$ and let $\mathbf{P}$ be the path from the root $\mathbf{r}(\mathbf{T})$ to v. For a node w in $\mathbf{P}$, let $V(w) = V(\phi(w))$ and $\phi'(w) = \bigwedge\{C \in \phi(w) : \ |C| \leq |V(\phi(w))| - \sqrt{|V(\phi(w))|}\}$. Note that there is a node w of $\mathbf{P}$ such that $\phi'(w) = \emptyset$. Let w_0 be the closest such node to the root, and let $u_1, u_2, \ldots, u_h$ be the R-nodes in the path $\mathbf{P}$ between $\mathbf{r}(\mathbf{T})$ and w_0, ordered by increasing distance from $\mathbf{r}(\mathbf{T})$.

For $\ell = 0, 1, \ldots, h$, let $n_\ell = |V(u_\ell)|$, where we assume $u_0 = \mathbf{r}(\mathbf{T})$. Note that

$$V(u_\ell) \subseteq V(p(u_\ell)) \subseteq V(u_{\ell-1}) \quad \text{and} \quad \phi'(u_\ell) \subseteq \phi'(p(u_\ell)) \subseteq \phi'(u_{\ell-1}),$$

for $\ell = 1, \ldots, h$. In particular, Lemma 11 implies $|\phi'(u_\ell)| \leq (1-\frac{1}{\sqrt{n_{\ell-1}}})|\phi'(u_{\ell-1})|$, for $\ell = 1, \ldots, h$, and thus $|\phi'(u_h)| \leq (1-1/\sqrt{n_0})^h|\phi'(u_0)|$. Since $|\phi'(u_h)| \geq 1$, we conclude that $\mathrm{d}(w_0) = l+1 \leq \sqrt{n}\ln m + 1$, where $n = n_0$.

From Theorem 5, we know that $|(\phi^v)^*| \leq |\phi^*|^{\mathrm{d}(w_0)-1}|(\phi'')^*|$, where $\phi'' \subseteq \phi(w_0)$ consists of clauses in $\phi(w_0) \cap \phi^v$. By definition of w_0, we have $|\phi'(w_0)| = 0$ and thus $\phi(w_0)$ consists only of clauses of size at least $|V(\phi(w_0))| - \sqrt{|V(\phi(w_0))|}$. Thus $|(\phi'')^*| \leq n^{\sqrt{n}+1}$ by Lemma 10, and hence

$$|(\phi^v)^*| \ \leq \ |\phi^*|^{\mathrm{d}(w_0)-1}n^{\sqrt{n}+1} \ \leq \ n^{\sqrt{n}+1}|\phi^*|^{\sqrt{n}\ln m}.$$

References

1. Bailey, J., Manoukian, T., Ramamohanarao, K.: A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In: ICDM, pp. 485–488 (2003)
2. Berge, C.: Hypergraphs. Elsevier-North Holand, Amsterdam (1989)
3. Bioch, J.C., Ibaraki, T.: Complexity of identification and dualization of positive Boolean functions. Information and Computation 123(1), 50–63 (1995)
4. Boros, E., Elbassioni, K., Gurvich, V., Khachiyan, L., Makino, K.: Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities. SIAM J. Comput. 31(5), 1624–1643 (2002)
5. Boros, E., Gurvich, V., Hammer, P.L.: Dual subimplicants of positive Boolean functions. Optim. Methods Softw. 10, 147–156 (1998)
6. Boros, E., Gurvich, V., Khachiyan, L., Makino, K.: On maximal frequent and minimal infrequent sets in binary matrices. Annals of Mathematics and Artificial Intelligence 39(3), 211–221 (2003)

7. Domingo, C., Mishra, N., Pitt, L.: Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries. Machine Learning 37(1), 89–110 (1999)
8. Eiter, T.: Exact transversal hypergraphs and application to Boolean μ-functions. Journal of Symbolic Computation 17(3), 215–225 (1994)
9. Eiter, T., Gottlob, G.: Identifying the minimal transversals of a hypergraph and related problems. SIAM J. Comput. 24(6), 1278–1304 (1995)
10. 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)
11. Eiter, T., Gottlob, G., Makino, K.: New results on monotone dualization and generating hypergraph transversals. SIAM J. Comput. 32(2), 514–537 (2003)
12. Eiter, T., Makino, K., Gottlob, G.: Computational aspects of monotone dualization: A brief survey. Discr. Applied Math. (2007), doi:10.1016/j.dam.2007.04.017
13. Elbassioni, K.M.: On the Complexity of the Multiplication Method for Monotone CNF/DNF Dualization. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 340–351. Springer, Heidelberg (2006)
14. Fredman, M.L., Khachiyan, L.: On the complexity of dualization of monotone disjunctive normal forms. Journal of Algorithms 21, 618–628 (1996)
15. Gunopulos, D., Mannila, H., Khardon, R., Toivonen, H.: Data mining, hypergraph transversals, and machine learning. In: PODS 1997, pp. 209–216 (1997)
16. Ibaraki, T., Kameda, T.: A theory of coteries: Mutual exclusion in distributed systems. IEEE Trans. Parallel and Dist. Sys. 4(7), 779–794 (1993)
17. Kavvadias, D.J., Papadimitriou, C.H., Sideri, M.: On horn envelopes and hypergraph transversals. In: ISAAC 1993, pp. 399–405 (1993)
18. Kavvadias, D.J., Stavropoulos, E.C.: An efficient algorithm for the transversal hypergraph generation. J. Graph Algorithms Appl. 9(2), 239–264 (2005)
19. Khachiyan, L.: Transversal hypergraphs and families of polyhedral cones. In: Advances in Convex Analysis and Global Optimization, honoring the memory of K. Carathéodory, pp. 105–118. Kluwer Academic Publishers, Dordrecht (2000)
20. Khachiyan, L., Boros, E., Elbassioni, K., Gurvich, V.: On the dualization of hypergraphs with bounded edge-intersections and other related classes of hypergraphs. Theor. Comput. Sci. 382(2), 139–150 (2007)
21. Khachiyan, L., Boros, E., Elbassioni, K., Gurvich, V., Makino, K.: On the complexity of some enumeration problems for matroids. SIAM J. Disc. Math. 19(4), 966–984 (2005)
22. Lawler, E., Lenstra, J.K., Rinnooy Kan, A.H.G.: Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. SIAM J. Comput. 9, 558–565 (1980)
23. Lovász, L.: Combinatorial optimization: some problems and trends. DIMACS Technical Report 92-53, Rutgers University (2000)
24. Mannila, H., Räihä, K.J.: Design by example: An application of armstrong relations. J. Comput. and Syst. Sci. 33(2), 126–141 (1986)
25. Papadimitriou, C.H.: NP-completeness: A retrospective. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 2–6. Springer, Heidelberg (1997)
26. Ramamurthy, K.G.: Coherent Structures and Simple Games. Kluwer Academic Publishers, Dordrecht (1990)
27. Takata, K.: On the sequential method for listing minimal hitting sets. In: DM & DM 2002: Proc. Workshop on Discr. Math. and Data Mining, pp. 109–120 (2002)

Diagonal Circuit Identity Testing and Lower Bounds

Nitin Saxena*

Hausdorff Center for Mathematics
Endenicher Allee 60
D-53115 Bonn, Germany
ns@hcm.uni-bonn.de

Abstract. In this paper we give the first deterministic polynomial time algorithm for testing whether a *diagonal* depth-3 circuit $C(x_1, \ldots, x_n)$ (i.e. C is a sum of powers of linear functions) is zero. We also prove an exponential lower bound showing that such a circuit will compute determinant or permanent only if there are exponentially many linear functions. Our techniques generalize to the following new results:

1. Suppose we are given a depth-4 circuit (over any field $\mathbb{F}$) of the form:

$$C(x_1, \ldots, x_n) := \sum_{i=1}^{k} L_{i,1}^{e_{i,1}} \cdots L_{i,s}^{e_{i,s}}$$

   where, each $L_{i,j}$ is a sum of univariate polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. We can test whether C is zero deterministically in $poly(size(C), max_i\{(1+e_{i,1}) \cdots (1+e_{i,s})\})$ field operations. In particular, this gives a deterministic polynomial time identity test for general depth-3 circuits C when the $d :=$degree(C) is logarithmic in the size(C).
2. We prove that if the above circuit $C(x_1, \ldots, x_n)$ computes the determinant (or permanent) of an $m \times m$ formal matrix with a "small" $s = o\left(\frac{m}{\log m}\right)$ then $k = 2^{\Omega(m)}$. Our lower bounds work for all fields $\mathbb{F}$. (Previous exponential lower bounds for depth-3 only work for nonzero characteristic.)
3. We also present an exponentially faster identity test for homogeneous diagonal circuits (deterministically in $poly(nk \log(d))$ field operations over finite fields).

Keywords: arithmetic circuit, identity testing, depth 3, depth 4, determinant, permanent, lower bounds.

1 Introduction

Identity Testing is the problem of checking whether a given arithmetic circuit $C(x_1, \ldots, x_n)$, computing a polynomial over a field $\mathbb{F}$, is the zero circuit. Ideally

* This work was done while the author was a postdoc in Centrum voor Wiskunde en Informatica, Amsterdam under BRICKS AFM1 and NWO VICI grants.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 60–71, 2008.

we would like to do identity testing deterministically in time polynomial in the size of the circuit C but no such algorithm is known. The simplest known general algorithm is randomized which was discovered independently by Schwartz [20] and Zippel [22]: it evaluates the given circuit at a random point and accepts if and only if the circuit evaluates to zero at that point. There are more involved randomized algorithms that use fewer random bits [2]. Besides being a natural algebraic problem, special cases of identity testing also appear in primality testing [3], testing equivalence of read-once branching programs [6], graph matching problems [15], interpolating sparse multivariate polynomials [7] and proving complexity theory results such as IP=PSPACE [21], NP=PCP($O(\log n)$, $O(1)$) [5]. Solving identity testing becomes all the more important by the work of Impagliazzo and Kabanets [11] who showed that – finding a deterministic algorithm for identity testing is, roughly, equivalent to proving circuit lower bounds for NEXP.

In this paper we consider arithmetic circuits of depth 4 and solve the identity testing problem for a natural restricted case. Our basic technique is to express the multiplication gate $(a_0+a_1x_1+\cdots+a_nx_n)^d$ in a *dual* form $\sum_j f_{j,1}(x_1)\cdots f_{j,n}(x_n)$. In full generality our dual form expresses a product-of-sum-of-univariates as a sum-of-product-of-univariates effectively (see Remark 1). Our technique of computing the dual is a new way to unfold a multiplication gate in an arithmetic circuit. The dual of a multiplication gate is obtained by using the tools of formal power series (of e^x), polynomial interpolation and working over local algebras. This dual computation is faster than a brute-force expansion and may have other applications. Finally, we also show that in the special case of homogeneous diagonal circuits we can actually do better and give a $poly(nk \log d)$ time identity test.

1.1 Known Results

There are deterministic algorithms known for identity testing only over restricted classes of arithmetic circuits. Raz and Shpilka [19] gave a deterministic polynomial time identity test for noncommutative arithmetic formulas. Dvir and Shpilka [8] attempted a characterization of zero depth-3 circuits and obtained a $poly(n, 2^{\log^{k-1} d})$ time identity test. Kayal and Saxena [14] used Chinese remaindering over local rings and gave a $poly(nd^k)$ time identity test for depth-3 circuits which is clearly a polynomial time identity test if k, the top fanin of the circuit, is bounded. In this work we allow the top fanin to be unbounded but impose the restriction that each multiplication gate has only "few" *distinct* functions as input. All these identity tests are non-black-box, i.e. they look *inside* the circuit instead of just evaluating it at points. Recently, there has been some attempts towards black-box identity testing for depth-3 circuits (see [12]). A black-box identity test even for depth-4 circuits would have important repercussions for the general identity testing problem [4].

In this paper we also prove exponential lower bounds for computing determinant or permanent by certain restricted depth-4 circuits. These restricted depth-4 circuits are the ones for which we give a deterministic polynomial time identity test. Grigoriev, Karpinski and Razborov [9,10] have also shown such

lower bounds for general depth-3 circuits but assuming a nonzero characteristic. Our lower bounds are new in the sense that they hold over all fields.

1.2 Definitions and Statement of Results

We will use $poly(M, N)$ to refer to a real function in M and N whose value is upper bounded by $(MN)^{c_1}$ for all $M, N > c_2$ where $c_1, c_2 > 0$ are absolute constants. When using $poly(M, N)$ we will not specify the value of c_1, c_2 as our main interest in this paper is only in their existence. We will use $[n]$ to refer to the set $\{1, \ldots, n\}$. We will denote the characteristic of a field $\mathbb{F}$ (i.e. smallest integer $t > 0$ such that $t = 0$ in $\mathbb{F}$ or zero if there is no such t) by $char(\mathbb{F})$. An algebra R over a field $\mathbb{F}$ is simply a ring containing $\mathbb{F}$. In this paper only finite dimensional commutative algebras appear, i.e. there is an integer $N > 0$ and basis elements $b_1, \ldots, b_N \in R$ such that any element in R can be uniquely expressed as $\sum_{i=1}^{N} \alpha_i b_i$ with α_i's in $\mathbb{F}$. We call N the *dimension* of the algebra R over $\mathbb{F}$, denoted by $dim(R)$. It is a simple exercise to see that basic operations (e.g. multiplication of two elements) in R can be done using $poly(N)$ field operations (in $\mathbb{F}$).

Our main concern in this paper are depth-3 (or depth-4) circuits. For the purposes of identity testing (also lower bounds for determinant and permanent) the hardest case is when the circuit has an addition gate at the top. These circuits are called $\Sigma\Pi\Sigma$ (or $\Sigma\Pi\Sigma\Pi$). It is clear that the output of such a $\Sigma\Pi\Sigma$ circuit $C(x_1, \ldots, x_n)$ would be: $\sum_{i=1}^{k} \ell_{i,1} \cdots \ell_{i,d_i}$, where the $\ell_{i,j} = (a_{i,j,0} + a_{i,j,1}x_1 + \cdots + a_{i,j,n}x_n)$ are linear functions over a field $\mathbb{F}$. We call k the top fanin of C, d_i the degree of the i-th multiplication gate and $d = max_i\{d_i\}$ the degree of C. The size of an arithmetic circuit is the number of addition, multiplication and input gates in its representation as a directed acyclic graph. Clearly, in the above setting $size(C)$ is dominated by knd. It is easy to see that by brute-force we can check whether a $\Sigma\Pi\Sigma$ circuit C is a zero circuit in time polynomial in $k \cdot \binom{n+d}{d}$ but this is generally exponential in $size(C)$.

In this paper we start with the case where each of the multiplication gates in C has only one distinct linear function as input. We call such a C a *diagonal* circuit and it looks like: $C(x_1, \ldots, x_n) = \sum_{i=1}^{k} b_i \cdot \ell_i^{d_i}$, where the b_i's are in $\mathbb{F}$ and the ℓ_i's are linear functions. Our techniques extend upto depth-4 circuits of the form:

$$C(x_1, \ldots, x_n) = \sum_{i=1}^{k} L_{i,1}^{e_{i,1}} \cdots L_{i,s}^{e_{i,s}} \tag{1}$$

where the $L_{i,j}$'s are not linear functions but sums of univariate polynomials, i.e. for all $i \in [k], j \in [s]$:

$$L_{i,j}(x_1, \ldots, x_n) = g_{i,j,1}(x_1) + \cdots + g_{i,j,n}(x_n)$$

where $g_{i,j,j'} \in \mathbb{F}[x_{j'}]$. Our first main theorem is:

Theorem 1. *Over any field $\mathbb{F}$, let C be a circuit given as in Equation (1). Then we can deterministically check whether C is a zero circuit in $poly(size(C), max_i\{(1 + e_{i,1}) \cdots (1 + e_{i,s})\})$ field operations.*

Thus, when s is constant or when s is logarithmic but $e_{i,j}$'s are constants we get a deterministic polynomial time identity test.

The lower bounds that we get, basically show that if a depth-3 circuit (or a restricted depth-4 circuit) computes determinant (or permanent) then either some of the multiplication gates have "lots" of distinct functions as inputs or the top fanin of the circuit is exponential. Our second main theorem is:

Theorem 2. *Over any field $\mathbb{F}$, if the circuit in Equation (1) expresses the determinant (or permanent) of a general $m \times m$ matrix with parameters $s = o\left(\frac{m}{\log m}\right)$, $n = m^2$ and $d = poly(m)$ then $k = 2^{\Omega(m)}$.*

Note that determinant (or permanent) of an $m \times m$ matrix is just a sum of $m!$ monomials. A monomial $y_1 \cdots y_m$ can be expressed as a sum of powers of $2^{O(m)}$ linear forms. Hence, determinant can be expressed by a sum of powers of at most $O(m!)$ linear forms and our lower bounds show that this is almost tight.

1.3 Our Techniques

We use non-black-box methods, i.e. we heavily use the structure of the given circuit. We use tools that previously have been used to understand noncommutative formulas, for example by Nisan, Wigderson [16,17], Raz and Shpilka [19]. We apply these old tools in a nontrivial way to understand depth-3 and depth-4 (commutative) circuits. For clarity let us note here the two old theorems in a generalized form.

A circuit $D(x_1, \ldots, x_n)$, over an algebra R over a field $\mathbb{F}$, is called noncommutative if each of its multiplication gate has ordered inputs and the variables $x_1, \ldots, x_n$ do not commute, i.e. for all $i \neq j$, $x_i \cdot x_j \neq x_j \cdot x_i$. The output $D(x_1, \ldots, x_n)$ is a formal expression in the ring $R\{x_1, \ldots, x_n\}$ of polynomials over noncommutative variables $x_1, \ldots, x_n$. Clearly, any commutative circuit $C(x_1, \ldots, x_n)$ can be turned into a noncommutative circuit $\tilde{C}(x_1, \ldots, x_n)$ by imposing an order on the inputs to its multiplication gates and assuming $x_i \cdot x_j \neq x_j \cdot x_i$ for all $i \neq j$. But now circuits C and $\tilde{C}$ are computing different polynomials and it may happen that C is a zero circuit but $\tilde{C}$ is a nonzero circuit. However, if $\tilde{C}$ is a zero circuit then C is surely a zero circuit as well. A circuit is called a *formula* if the fan-out of every gate in the circuit is at most one. Noncommutative formulas are easier to analyze compared to the commutative ones and the following identity test is relevant to us:

Theorem 3 (Theorem 2.5 of [19] generalized over algebras). *Let R be an algebra over a field $\mathbb{F}$. Given a noncommutative formula $C(x_1, \ldots, x_n) \in R\{x_1, \ldots, x_n\}$ we can verify deterministically in $poly(size(C), dim(R))$ field operations whether C is zero.*

The second result relevant to us is an extension of Theorem 5.1 of [19] that proves lower bounds for pure circuits using the partial derivative space (see the proof idea in Lemma 5.3 of [19]).

Theorem 4 (Theorem 5.1 of [19] generalized over algebras). *Let R be an algebra over a field $\mathbb{F}$, $r \in R \setminus \{0\}$, $r' \in R$ and let $det(x_{1,1}, \ldots, x_{n,n})$ denote the determinant of a formal $n \times n$ matrix $((x_{i,j}))$. If $det(x_{1,1}, \ldots, x_{n,n}) \cdot r - r'$ can be expressed as a circuit:*

$$C(x_{1,1}, \ldots, x_{n,n}) = \sum_{i=1}^{k} f_{i,1,1}(x_{1,1}) \cdots f_{i,n,n}(x_{n,n})$$

where the f_{i,j_1,j_2}'s are univariate polynomials over R, then $k \cdot dim(R) = 2^{\Omega(n)}$. A similar lower bound holds for the permanent as well.

Proof (Sketch). Since determinant is a multilinear polynomial we can ignore the nonlinear terms in the f_{i,j_1,j_2}'s. Now if we look at the suitably defined partial derivative space (as in [19]) of the circuit C then it has rank, over $\mathbb{F}$, at most $k \cdot dim(R)$ because there are k multiplication gates and the coefficients in f_{i,j_1,j_2}'s are themselves of dimension $dim(R)$ over $\mathbb{F}$. On the other hand it is known that the corresponding rank of determinant is $2^{\Omega(n)}$.

Our main contribution is a novel way to transform the multiplication gates of a circuit, hence the overall circuit, to a form on which we can apply Theorems 3 and 4. Our basic technique is to use the formal power series $e^x = 1 + x + \frac{x^2}{2!} + \cdots$ and polynomial interpolation to express the multiplication gate $(a_0 + a_1x_1 + \cdots + a_nx_n)^d$ in a *dual* form: $\sum_j f_{j,1}(x_1) \cdots f_{j,n}(x_n)$. Now this is a nice circuit as the variables $x_1, \ldots, x_n$ in it are "separated" and we can invoke the known tricks. For example, it can be viewed as a circuit in which the variables $x_1, \ldots, x_n$ do not commute, thus by Theorem 3 we get a deterministic polynomial time identity test for diagonal circuits. Also, by the lower bounds of Theorem 4 we get that a diagonal circuit can compute determinant or permanent only if it is of exponential size. These ideas generalize to circuits with $s > 1$ in Equation (1) but require more algebraic sophistication as then we work with the formal power series on larger local algebras instead of working on the base field $\mathbb{F}$.

1.4 Organization

The paper is organized as follows. In section 2 we present our results for the basic case of diagonal circuits over zero characteristic. In section 3 we show how to extend our results to restricted depth-4 circuits over zero characteristic. In section 4 we extend the previous results to nonzero characteristic. Finally, in section 5 we present an exponentially faster identity test for homogeneous diagonal circuits (deterministically in $poly(nk \log(d))$ field operations over finite fields). Some of the proofs have been omitted from the extended abstract due to space constraints.

2 Diagonal Depth-3 Circuits

The aim of this section is to define a dual expression for multiplication gates of the form $(a_0 + a_1x_1 + \cdots + a_nx_n)^d$ and use that form to give an identity

test for diagonal circuits and to prove lower bounds. We will assume throughout this section that the base field $\mathbb{F}$ is of characteristic zero. We will use the fairly standard notation $[m]f(x_1, \ldots, x_n)$ to denote the coefficient of the monomial m in a polynomial (more generally, a power series) f. For example, $[xyz](x + y + z)^3 = 6$.

2.1 Dual of a Multiplication Gate

The following lemma formalizes and computes the dual of an affine power.

Lemma 1. *Let $a_0, a_1, \ldots, a_n$ be in a field $\mathbb{F}$ of zero characteristic. Then we can compute univariate polynomials $f_{i,j}$'s in poly(nd) field operations such that for $t = (nd + d + 1)$:*

$$(a_0 + a_1x_1 + \cdots + a_nx_n)^d \;=\; \sum_{i=1}^{t} f_{i,1}(x_1) \cdots f_{i,n}(x_n)$$

Proof. We will prove this using the formal power series: $\exp(x) = 1+x+\frac{x^2}{2!}+\cdots$, where $\exp(x) = e^x$ and e is the base of natural logarithm. Define the degree d truncation of the series to be $E_d(x) = 1 + x + \cdots + \frac{x^d}{d!}$. Observe that:

$$\begin{aligned}(d!)^{-1} \cdot (a_0 + a_1x_1 + \cdots + a_nx_n)^d \;&=\; [z^d] \exp\left((a_0 + a_1x_1 + \cdots + a_nx_n) \cdot z\right) \\ &=\; [z^d] \exp(a_0z) \cdot \exp(a_1x_1z) \cdots \exp(a_nx_nz) \\ &=\; [z^d]\; E_d(a_0z) \cdot E_d(a_1x_1z) \cdots E_d(a_nx_nz)\end{aligned}$$

The product $E_d(a_0z) \cdot E_d(a_1x_1z) \cdots E_d(a_nx_nz)$ can be viewed as a univariate polynomial in z of degree $(nd+d)$. Hence, its coefficient of z^d can be computed by evaluating the polynomial at $(nd+d+1)$ distinct points $\alpha_1, \ldots, \alpha_{nd+d+1} \in \mathbb{F}$ (remember $\mathbb{F}$ is large enough) and by interpolation we can compute $\beta_1, \ldots, \beta_{nd+d+1} \in \mathbb{F}$ such that:

$$\begin{aligned}&[z^d]\; E_d(a_0z) \cdot E_d(a_1x_1z) \cdots E_d(a_nx_nz) \\ =\; &\sum_{i=1}^{nd+d+1} \beta_i \cdot E_d(a_0\alpha_i) \cdot E_d(a_1\alpha_ix_1) \cdots E_d(a_n\alpha_ix_n)\end{aligned}$$

This is the dual form of $(a_0+a_1x_1+\cdots+a_nx_n)^d$ as required. It is routine to verify that all the univariate polynomials $E_d(\cdot)$ in the above sum can be computed in *poly*(nd) field operations.

2.2 Identity Test and Lower Bounds

The dual form of multiplication gates obtained in Lemma 1 is easy to analyze. We give the ideas in the following theorems.

Theorem 5. *Over zero characteristic, identity testing for diagonal circuits can be done in deterministic polynomial time (poly(nkd) field operations).*

Proof. Suppose we are given a diagonal circuit C:

$$C(x_1, \ldots, x_n) = \sum_{i=1}^{k} b_i \cdot \ell_i^{d_i}$$

Then by Lemma 1 we can compute the dual form of each of the k multiplication gates such that:

$$C(x_1, \ldots, x_n) = \sum_{i=1}^{k} \sum_{j=1}^{nd_i+d_i+1} f_{i,j,1}(x_1) \cdots f_{i,j,n}(x_n) \tag{2}$$

where the univariate polynomials $f_{i,j,j'}$'s are of degree at most d_i.

Now observe that the variables in the circuit on the RHS of Equation (2) can be assumed to be noncommutative without affecting the output, i.e. circuit C. Thus, if we apply the identity testing algorithm of Theorem 3 to the circuit on the RHS of Equation (2) we will correctly know whether C is zero or not. Hence, C can be verified for zeroness deterministically in $poly(nkd)$ field operations.

Theorem 6. *Over zero characteristic, if a diagonal circuit expresses the determinant (or permanent) of a formal $m \times m$ matrix with $n = m^2$ variables and degree $d = poly(m)$ then the top fanin $k = 2^{\Omega(m)}$.*

Proof. Suppose a diagonal circuit C computes the determinant of a general $m \times m$ matrix. Then by Lemma 1 determinant is being computed by a circuit as given in Equation (2). Now the exponential lower bound of Theorem 4 applies and we get that $poly(ndk) = 2^{\Omega(m)}$ implying $k = 2^{\Omega(m)}$.

3 Extension to Restricted Depth-4 Circuits

In this section we extend the results of the last section to depth-4 circuits (with some success). The starting point is a dual expression for multiplication gates of the form $L_1^{e_1} \cdots L_s^{e_s}$ where the L_i's are sums of univariate polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. The proof is along the same lines as presented before but now we will work in local algebras over $\mathbb{F}$. Finally, we use that form to give identity test and prove lower bounds. We will again assume throughout this section that the base field $\mathbb{F}$ is of characteristic zero.

3.1 Dual of a Multiplication Gate

We compute the dual form of a multiplication gate of the form:

$$M(x_1, \ldots, x_n) = (g_{1,1}(x_1) + \cdots + g_{1,n}(x_n))^{e_1} \cdots (g_{s,1}(x_1) + \cdots + g_{s,n}(x_n))^{e_s} \tag{3}$$

which means that we express M as an expression: $\sum_{i=1}^{t} f_{i,1}(x_1) \cdots f_{i,n}(x_n)$ where the $f_{i,j}$'s are univariate polynomials over an $\mathbb{F}$-algebra R (unlike the diagonal case where we worked over $\mathbb{F}$). This expression with variables $x_1, \ldots, x_n$ "separated" we call a *dual* of the multiplication gate. The following lemma shows that such a dual is computable but we pay a price in terms of the dimension of algebra R which is $(e_1 + 1) \cdots (e_s + 1)$.

Lemma 2. *Let $M(x_1,\ldots,x_n)$ be the multiplication gate of Equation (3) over a field $\mathbb{F}$ of zero characteristic and $e=(e_1+\cdots+e_s)$. Then we can compute univariate polynomials $f_{i,j}$'s over an algebra $R:=\mathbb{F}[z_1,\ldots,z_s]/(z_1^{e_1+1},\ldots,z_s^{e_s+1})$ in poly$(size(M),dim(R))$ field operations such that for $t=(ne+1)$:*

$$M(x_1,\ldots,x_n)\cdot z_1^{e_1}\cdots z_s^{e_s} \;=\; \sum_{i=1}^{t} f_{i,1}(x_1)\cdots f_{i,n}(x_n) \quad \textit{over } R$$

Remark 1. Note that we can informally describe the above equation as: a product-of-sums-of-univariates can be written as a sum-of-products-of-univariates. This justifies our continued usage of the phrase "dual form".

Proof. We will again prove this using the formal power series: $\exp(x)=1+x+\frac{x^2}{2!}+\cdots$, where $\exp(x)=e^x$ and e is the base of natural logarithm. Recall that the degree d truncation of the series is $E_d(x)=1+x+\cdots+\frac{x^d}{d!}$. Let $L_1,\ldots,L_s$ be the distinct factors of M (that are now not linear functions but sums of univariate polynomials). Observe that:

$$\begin{aligned}
(e_1!\cdots e_s!)^{-1}\cdot L_1^{e_1}\cdots L_s^{e_s} &= [z^e z_1^{e_1}\cdots z_s^{e_s}]\,\exp(L_1z_1z)\cdots\exp(L_sz_sz)\\
&= [z^e z_1^{e_1}\cdots z_s^{e_s}]\,\exp(L_1z_1z+\cdots+L_sz_sz)\\
&= [z^e z_1^{e_1}\cdots z_s^{e_s}]\,\exp\left((g_{1,1}z_1+\cdots+g_{s,1}z_s)z\right)\cdots\\
&\qquad\cdots\exp\left((g_{1,n}z_1+\cdots+g_{s,n}z_s)z\right)\\
&= [z^e z_1^{e_1}\cdots z_s^{e_s}]\,E_e\left((g_{1,1}z_1+\cdots+g_{s,1}z_s)z\right)\cdots\\
&\qquad\cdots E_e\left((g_{1,n}z_1+\cdots+g_{s,n}z_s)z\right) \qquad (4)
\end{aligned}$$

Note that the last product can be viewed as a univariate polynomial in z of degree ne. Hence, its coefficient of z^e can be computed by evaluating the polynomial at $(ne+1)$ distinct points $\alpha_1,\ldots,\alpha_{ne+1}\in\mathbb{F}$ (remember that $\mathbb{F}$ is large enough) and by interpolation we can compute $\beta_1,\ldots,\beta_{ne+1}\in\mathbb{F}$ such that:

$$\begin{aligned}
&[z^e z_1^{e_1}\cdots z_s^{e_s}]\,E_e\left((g_{1,1}z_1+\cdots+g_{s,1}z_s)z\right)\cdots E_e\left((g_{1,n}z_1+\cdots+g_{s,n}z_s)z\right)\\
&= [z_1^{e_1}\cdots z_s^{e_s}]\sum_{i=1}^{ne+1}\beta_i\cdot E_e\left((g_{1,1}z_1+\cdots+g_{s,1}z_s)\alpha_i\right)\cdots\\
&\qquad\cdots E_e\left((g_{1,n}z_1+\cdots+g_{s,n}z_s)\alpha_i\right)
\end{aligned}$$

Notice that the monomials having nonzero coefficients in the above sum are of the form $z_1^{t_1}\cdots z_s^{t_s}$ such that $t_1+\cdots+t_s=e=e_1+\cdots+e_s$. Thus, if we look at the above sum modulo the ideal $(z_1^{e_1+1},\ldots,z_s^{e_s+1})$ then the surviving monomials $z_1^{t_1}\cdots z_s^{t_s}$ would be those that have $t_1\leqslant e_1,\ldots,t_s\leqslant e_s$ which together with $t_1+\cdots+t_s=e=e_1+\cdots+e_s$ uniquely determines the surviving monomial as $z_1^{e_1}\cdots z_s^{e_s}$. Consequently, we can summarize the above computations as, over R:

$$\begin{aligned}
&(e_1!\cdots e_s!)^{-1}\cdot L_1^{e_1}\cdots L_s^{e_s}\cdot z_1^{e_1}\cdots z_s^{e_s}\\
&= \sum_{i=1}^{ne+1}\beta_i\cdot E_e\left((g_{1,1}z_1+\cdots+g_{s,1}z_s)\alpha_i\right)\cdots E_e\left((g_{1,n}z_1+\cdots+g_{s,n}z_s)\alpha_i\right)\;.
\end{aligned}$$

This is the dual form of M as required. Notice that there is a nonconstant factor $z_1^{e_1} \cdots z_s^{e_s}$ appearing on the LHS but since this factor is a nonzero element of the algebra R, the dual form will be good enough for our purposes. It is routine to verify that the univariate polynomials $E_e(\cdot)$ over R in this sum can be computed in $poly(size(M), dim(R))$ field operations and that the dimension of R is $(e_1 + 1) \cdots (e_s + 1)$.

3.2 Identity Test and Lower Bounds

We can now apply the dual form of Lemma 2 to k multiplication gates and work on a bigger algebra. We formalize this idea in the following theorems.

Theorem 7. *Given a circuit C over a field $\mathbb{F}$ of zero characteristic:*

$$C(x_1, \ldots, x_n) = \sum_{i=1}^{k} L_{i,1}^{e_{i,1}} \cdots L_{i,s}^{e_{i,s}}$$

where the $L_{i,j}$'s are sums of univariate polynomials and (wlog) for all $i, e_{i,1} \neq 0$. We can test whether C is a zero circuit deterministically in $poly(size(C), max_i\{(1 + e_{i,1}) \cdots (1 + e_{i,s})\})$ field operations.

Proof. Let us apply the dual form of Lemma 2 to the i-th multiplication gate M_i, with $e_i := (e_{i,1} + \cdots + e_{i,s})$, and compute the univariate polynomials f_{i,j_1,j_2}'s, for all $1 \leqslant j_1 \leqslant t_i = (ne_i + 1)$ and $j_2 \in [n]$, over the algebra $R_i := \mathbb{F}[z_{i,1}, \ldots, z_{i,s}]/(z_{i,1}^{e_{i,1}+1}, \ldots, z_{i,s}^{e_{i,s}+1})$ in $poly(size(M_i), dim(R_i))$ field operations such that:

$$L_{i,1}^{e_{i,1}} \cdots L_{i,s}^{e_{i,s}} \cdot z_{i,1}^{e_{i,1}} \cdots z_{i,s}^{e_{i,s}} \;=\; \sum_{j_1=1}^{t_i} f_{i,j_1,1}(x_1) \cdots f_{i,j_1,n}(x_n) \;\text{ over } R_i \qquad (5)$$

With the aim of getting a dual form of the circuit C let us define a commutative algebra R that contains the algebras corresponding to each multiplication gate, i.e. $R_1, \ldots, R_k$, as "orthogonal" subalgebras and in which the following $(k-1)$ relations hold: $z_{1,1}^{e_{1,1}} \cdots z_{1,s}^{e_{1,s}} = \cdots = z_{k,1}^{e_{k,1}} \cdots z_{k,s}^{e_{k,s}}$. Explicitly, the algebra R is: $\mathbb{F}[z_{i,j} \mid \forall i \in [k], \forall j \in [s]]/\mathcal{I}$, where the ideal $\mathcal{I}$ is generated by the following three sets of relations:

- $z_{i,j}^{e_{i,j}+1} = 0$, for all $i \in [k], j \in [s]$.
- $z_{i,j} \cdot z_{i',j'} = 0$, whenever $i \neq i'$.
- $z_{i,1}^{e_{i,1}} \cdots z_{i,s}^{e_{i,s}} = z_{i',1}^{e_{i',1}} \cdots z_{i',s}^{e_{i',s}}$, for all $i, i' \in [k]$.

Note that the first set of relations just make $R_1, \ldots, R_k$ as subalgebras of R while the other two sets impose relations on certain zero-divisors in R ($e_{i,1} \neq 0$ implies that $z_{i,1}^{e_{i,1}} \cdots z_{i,s}^{e_{i,s}}$ is a zero-divisor of R). The second set of relations are put in so that the dimension of R gets down to roughly sum of the dimensions of $R_1, \ldots, R_k$. Note that the dimension of R over the base field $\mathbb{F}$ is exactly $\sum_{i=1}^{k}(1 + e_{i,1}) \cdots (1 + e_{i,s}) - 2(k-1)$ which is nonzero.

Now by using the third set of relations in R and summing up Equation (5) for all the k multiplication gates, we get over the algebra R:

$$\begin{aligned} C(x_1,\ldots,x_n)\cdot z_{1,1}^{e_{1,1}}\cdots z_{1,s}^{e_{1,s}} &= \sum_{i=1}^{k} L_{i,1}^{e_{i,1}}\cdots L_{i,s}^{e_{i,s}}\cdot z_{i,1}^{e_{i,1}}\cdots z_{i,s}^{e_{i,s}} \\ &= \sum_{i=1}^{k}\sum_{j_1=1}^{t_i} f_{i,j_1,1}(x_1)\cdots f_{i,j_1,n}(x_n) \qquad (6) \end{aligned}$$

This last expression can be viewed as a noncommutative formula in variables $x_1,\ldots,x_n$ over the algebra R. Clearly, it is zero *iff* $C(x_1,\ldots,x_n)\cdot z_{1,1}^{e_{1,1}}\cdots z_{1,s}^{e_{1,s}}$ is zero over R *iff* C is zero over $\mathbb{F}$. Thus, it is sufficient to test the circuit on the RHS of Equation (6) for zeroness. This can be done by applying the identity testing algorithm of Theorem 3, now working over the algebra R. Hence, we can deterministically verify whether C is zero in $poly(size(C), dim(R))$ field operations as required.

Theorem 8. *Following the notation of the last theorem, if C expresses the determinant (or permanent) of a formal $m \times m$ matrix with parameters $s = o\left(\frac{m}{\log m}\right)$, $n = m^2$ and $(e_1 + \cdots + e_k) := e = poly(m)$ then $k = 2^{\Omega(m)}$.*

Proof. Suppose the circuit C computes the determinant of a general $m \times m$ matrix. Recall that C has a dual form as given in Equation (6). Thus, we can apply Theorem 4 to deduce that $poly(nek, dim(R)) = 2^{\Omega(m)}$ implying:

$$poly\left(nek, max_i\{(1+e_{i,1})\cdots(1+e_{i,s})\}\right) = 2^{\Omega(m)}$$

As the $e_{i,j}$'s are at most $poly(m)$ the above implies $poly(nek, m^s) = 2^{\Omega(m)}$ which using the hypothesis further implies $k = 2^{\Omega(m)}$.

4 Extension to the Nonzero Characteristic Case

In the last section we defined the dual form of a multiplication gate $L_1^{e_1}\cdots L_s^{e_s}$, where the L_i's are sums of univariates over a field $\mathbb{F}$ of zero characteristic. In this section we note how to obtain the dual form when the characteristic of $\mathbb{F}$ is a prime $p > 1$. Note that over such a field the expressions used in the proof of Lemma 2 may not be well defined, for example if $p|d!$ then $\frac{1}{d!}$ is undefined in $\mathbb{F}$. We can show that such issues can be taken care of by working in a local algebra over a *Galois ring* of characteristic p^b instead of working over $\mathbb{F}$. This finishes the proofs of our main Theorems 1 and 2.

5 A Faster Identity Test for Diagonal Circuits

Identity testing for homogeneous diagonal circuits can be made exponentially faster in the degree d of the circuit. Unfortunately, we only know how to do this

over a finite field $\mathbb{F}$ with an extra assumption that $d < char(\mathbb{F})$ (we do believe it should be possible to do this in general). The main idea to speed up the identity test is that if the degree d of a diagonal circuit C is large compared to fanin k then an argument using Vandermonde's matrix shows that C can be zero only if each multiplication gate is zero, which can be tested in time $poly(nk \log(d))$. Thus, wlog we can assume $d \leqslant k$ and the identity test given in this paper tests $\sum_{i=1}^{k} b_i \cdot \ell_i^d = 0$ deterministically in $poly(nk)$ field operations.

6 Conclusion

In this work we gave a deterministic polynomial time identity test for restricted depth-4 circuits. Our basic idea was to define a dual operation on the multiplication gates in a depth-3 circuit that converts a product gate into a sum of product of univariate polynomials over a local algebra. This dual is efficiently computable when the multiplication gate has "few" distinct linear functions as input. In the case of a general multiplication gate of a depth-3 circuit of degree d the dual computation takes exponential time: $poly(n2^d)$. This dual computation can be viewed as a new way to unfold a given depth-3 circuit better than the direct brute-force expansion. We leave it as an open question to improve this duality to solve the identity testing problem for general depth-3 circuits.

Kayal [13] has observed that Theorems 1 and 2 for depth-3 circuits can also be obtained (nontrivially) by using the space of partial derivatives first defined by Nisan and Wigderson [17]. The basic reason is that the space of partial derivatives of a diagonal circuit has "low" rank and this can be exploited to give an identity test and proving lower bounds. However, in the case of our restricted depth-4 circuits the space of partial derivatives typically has "high" rank. For example, the partial derivative space of $(x_1^2 + \cdots + x_n^2)^n$ is of rank more than 2^n. Thus, the dual form analyzes the restricted depth-4 circuits in ways stronger than the partial derivative space.

Acknowledgements

I would like to thank Harry Buhrman and Ronald de Wolf for various useful discussions during the course of this work. I thank Neeraj Kayal and Avi Wigderson for providing insightful observations on a preliminary version of this paper.

References

1. Adleman, L.M., Lenstra, H.W.: Finding irreducible polynomials over finite fields. In: 18th ACM Symposium on Theory of Computing, pp. 350–355. ACM Press, New York (1986)
2. Agrawal, M., Biswas, S.: Primality and identity testing via Chinese remaindering. Journal of the ACM 50(4), 429–443 (2003)
3. Agrawal, M., Kayal, N., Saxena, N.: Primes is in P. Annals of Mathematics 160(2), 781–793 (2004)

4. Agrawal, M., Vinay, V.: Arithmetic Circuits: A Chasm at Depth Four (preprint, 2008)
5. Arora, S., Safra, S.: Probabilistic Checking of Proofs: A New Characterization of NP. Journal of the ACM 45(1), 70–122 (1998)
6. Blum, M., Chandra, A.K., Wegman, M.N.: Equivalence of free Boolean graphs can be tested in polynomial time. Information Processing Letters 10, 80–82 (1980)
7. Clausen, M., Dress, A., Grabmeier, J., Karpinski, M.: On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. Theoretical Computer Science 84(2), 151–164 (1991)
8. Dvir, Z., Shpilka, A.: Locally decodable codes with two queries and polynomial identity testing for depth 3 circuits. SIAM J. Comput. 36(5), 1404–1434 (2007)
9. Grigoriev, D., Karpinski, M.: An Exponential Lower Bound for Depth 3 Arithmetic Circuits. In: 30th ACM Symposium on Theory of Computing, pp. 577–582. ACM Press, New York (1998)
10. Grigoriev, D., Razborov, A.A.: Exponential Lower Bounds for Depth 3 Arithmetic Circuits in Algebras of Functions over Finite Fields. Appl. Algebra Eng. Commun. Comput. 10(6), 465–487 (2000)
11. Impagliazzo, R., Kabanets, V.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1/2), 1–46 (2004)
12. Karnin, Z., Shpilka, A.: Deterministic black box polynomial identity testing of depth-3 arithmetic circuits with bounded top fan-in. ECCC Report TR07-042 (2007)
13. Kayal, N.: Private Communication. Summer (2007)
14. Kayal, N., Saxena, N.: Polynomial Identity Testing for Depth 3 Circuits. Computational Complexity 16(2), 115–138 (2007)
15. Lovasz, L.: On determinants, matchings, and random algorithms. In: Fundamentals of Computing Theory, pp. 565–574. Akademia-Verlag (1979)
16. Nisan, N.: Lower bounds for non-commutative computation. In: 23rd ACM Symposium on Theory of Computing, pp. 410–418. ACM Press, New York (1991)
17. Nisan, N., Wigderson, A.: Lower bounds on arithmetic circuits via partial derivatives. Computational Complexity 6(3), 217–234 (1997)
18. Raz, R.: Multi-linear formulas for permanent and determinant are of super-polynomial size. In: 36th ACM Symposium on Theory of Computing, pp. 633–641. ACM Press, New York (2004)
19. Raz, R., Shpilka, A.: Deterministic polynomial identity testing in non-commutative models. Computational Complexity 14(1), 1–19 (2005)
20. Schwartz, J.T.: Fast Probabilistic Algorithms for Verification of Polynomial Identities. Journal of the ACM 27(4), 701–717 (1980)
21. Shamir, A.: IP=PSPACE. Journal of the ACM 39(4), 869–877 (1992)
22. Zippel, R.: Probabilistic Algorithms for Sparse Polynomials. In: International Symposium on Symbolic and Algebraic Computation, pp. 216–226. Springer, Heidelberg (1979)

Cell-Probe Proofs and Nondeterministic Cell-Probe Complexity

Yitong Yin*

Department of Computer Science, Yale University
yitong.yin@yale.edu

Abstract. We study the nondeterministic cell-probe complexity of static data structures. We introduce *cell-probe proofs (CPP)*, a proof system for the cell-probe model, which describes verifications instead of computations in the cell-probe model. We present a combinatorial characterization of CPP. With this novel tool, we prove the following lower bounds for the nondeterministic cell-probe complexity of static data structures:

- We show that there exist data structure problems which have super-constant nondeterministic cell-probe complexity. In particular, we show that for the exact nearest neighbor search (NNS) problem or the partial match problem in high dimensional Hamming space, there does not exist a static data structure with $\mathrm{Poly}(n)$ cells, each of which contains $n^{o(1)}$ bits, such that the nondeterministic cell-probe complexity is $O(1)$, where n is the number of points in the data set for the NNS or partial match problem.
- For the polynomial evaluation problem, if single-cell nondeterministic probes are sufficient, then either the size of a single cell is close to the size of the whole polynomial, or the total size of the data structure is close to that of a naive data structure that stores results for all possible queries.

1 Introduction

We study the problem of nondeterministic cell-probe complexity of static data structures.

Given a set Y of data instances, and a set X of possible queries, a data structure problem can be abstractly defined as a function f mapping each pair consisting of a query $x \in X$ and a data instance $y \in Y$ to an answer. One of the most well-studied examples of data structure problems is the "membership query": $X = [m]$ is a data universe, $Y = \binom{[m]}{n}$, and $f(x, y) = 1$ if $x \in y$ and $f(x, y) = 0$ if otherwise.

There are some other important examples of data structure problems:

Exact nearest neighbor search (NNS): given a metric space U, let $X = U$ and $Y = \binom{U}{n}$, and for every $x \in X$ and $y \in Y$, $f(x, y)$ is defined as the closest point to x in y according to the metric.

* Supported by a Kempner Foundation Fellowship and NSF grant CNS-0435201.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 72–83, 2008.

Partial match: $X = \{0, 1, *\}^d$, $Y = \binom{\{0,1\}^d}{n}$, and $f(x, y) \in \{0, 1\}$ such that for every $x \in X$ and $y \in Y$, $f(x, y) = 1$ if and only if there exists $z \in y$ having either $x_i = z_i$ or $x_i = *$ for every i.
Polynomial evaluation: $X = 2^k$ is a finite field, $Y = 2^{kd}$ is the set of all $(d-1)$-degree polynomials over the finite field 2^k, and $f(x, y)$ returns the value of $y(x)$.

A classic computational model for static data structures is the cell-probe model [11]. For each data instance y, a table of cells is constructed to store y. This table is called a static data structure for some problem f. Upon a query x, an all-powerful algorithm tries to compute $f(x, y)$, based on adaptive random access (probes) to the cells.

The cell-probe model is a clean and general model for static data structures and serves as a great tool for the study of lower bounds. Previous research on static data structures in the cell-probe model has focused on the complexity of adaptive cell-probes. In this work, we focus on the complexity of nondeterministic cell-probes and the tradeoff between the number of probes needed and with space. We speculate that it is an important problem because: (1) in considering the complexity of data structures, nondeterminism is a very natural extension to the cell-probe model; (2) instead of adaptive computations, nondeterministic cell-probes capture the question of verification, which is a natural and important aspect of data structures.

Although nondeterministic cell-probe complexity is an important problem, there are few general tools and techniques for studying it, especially for the case of static data structures. In fact, because of the great generality of the cell-probe model, even for deterministic cell-probe complexity, super-constant lower bounds for static data structures are rare. Nondeterminism grants the cell-probe model extra power and makes non-trivial lower bounds even rarer. For many standard examples of data structure problems, such as membership query, it is easy to construct a data structure that has standard space usage and *constant* nondeterministic cell-probe complexity.

It is thus worth asking whether there exists any data structure problem such that in data structures with feasible sizes (polynomial in the size of data set), the nondeterministic cell-probe complexity is super-constant. More importantly, it calls for a general technique to prove lower bounds on the nondeterministic cell-probe complexity of static data structures.

1.1 Our Contribution

In this paper, we initiate the study of nondeterministic cell-probe complexity for static data structures. As a first step, we characterize the power of a single-cell nondeterministic probe. Although at first glance this may seem like a very restricted case, by applying a trivial parameter reduction, we show that the case of a single-cell probe is actually a canonical case for all nondeterministic cell-probe mechanisms, and is thus sufficient to prove super-constant lower bounds for general nondeterministic cell-probe mechanisms.

We introduce cell-probe proofs, a proof system in the cell-probe model. This notion of proofs corresponds to considering verifications instead of computations in the cell-probe model. Unlike the fully adaptive computations in the traditional cell-probe model, the formulation of cell-probe proofs shows a combinatorial simplicity. We introduce a combinatorial structure that fully characterizes which problems have single-cell proofs, and general cell-probe proofs are reduced to this case.

With these novel tools, we show following lower bounds on nondeterministic cell-probe complexity:

- We show that there exist static data structure problems with super-constant nondeterministic cell-probe complexity. In particular, we show that for the exact nearest neighbor search (NNS) problem or partial match problem in high dimensional Hamming space, there does not exist a static data structure with $\text{Poly}(n)$ cells, each of which contains $n^{o(1)}$ bits, such that the nondeterministic cell-probe complexity is $O(1)$, where n is the number of points in the data set for the NNS or partial match problem.
- For the polynomial evaluation problem, if for a static data structure, the single-cell nondeterministic probes are sufficient to answer queries, then either the size of the single cell is close to the size of the whole polynomial, or the total size of the data structure is close to that of the naive data structure that stores results for all possible queries.

1.2 Related Work

To the best of our knowledge, there is no general technique for proving lower bounds for nondeterministic cell-probe complexity of static data structures. Nor do there exist any non-trivial lower bounds for this question. Previous work on static data structures in the cell-probe model have focused on the complexity of adaptive cell-probes. The most important tool for proving such lower bounds is asymmetric communication complexity as introduced by Miltersen *et al.* in [10].

In [6], Fredman and Saks introduce the *chronogram method*. This powerful technique is specialized for proving the query/update trade-off for dynamic data structures, especially for the problems which are hard only in the dynamic case. It is worth noting that the chronogram method can prove nondeterministic lower bounds for certain dynamic data structure problems. This is formally addressed by Husfeldt and Rauhe in [7], and recently by Demaine and Pătraşcu in [5]. However, as pointed in [7], this is only a by-product of the nondeterministic nature of chronogram method and can only yield amortized query/update trade-offs for dynamic data structure problems with a certain property. Because of the unique structure of the chronogram method, this technique can not be utilized to prove lower bounds for static data structures.

2 Cell-Probe Proofs

A **static data structure problem** or just **data structure problem**, is represented as a boolean function $f : X \times Y \to \{0, 1\}$. For the purposes of proving

lower bounds, we only consider decision problems. We refer to each $y \in Y$ as **data** and each $x \in X$ as a **query**. For each pair of x and y, $f(x, y)$ specifies the result of the query x to the data structure that represents the data y.

In the cell-probe model (c.f. [11,6]), the data instance y is preprocessed and stored in cells, and for each query x, the value of $f(x, y)$ is decided by adaptive probes to the cells. Formally, a cell-probe scheme consists of a table structure and a query algorithm. The table structure $T : Y \times I \to \{0,1\}^b$ specifies a table $T_y : I \to \{0,1\}^b$ for each data instance y, which maps indices of cells to their contents. Given a query x, the query algorithm makes a sequence of probes $i_1, i_2, \ldots$ to the cells, where i_k depends on x and all previous cell probes $\langle i_1, T_y(i_1)\rangle, \langle i_2, T_y(i_2)\rangle, \ldots, \langle i_{k-1}, T_y(i_{k-1})\rangle$. The value of $f(x, y)$ is decided at last based on the collected information.

In this work, we focus on nondeterministic cell-probes. Given a query x to a data instance y, a set of t cells $i_1, i_2, \ldots, i_t$ are probed nondeterministically, such that the value of $f(x, y)$ is decided based on the probed information $\langle i_1, T_y(i_1)\rangle, \langle i_2, T_y(i_2)\rangle, \ldots, \langle i_t, T_y(i_t)\rangle$.

In order to formally characterize nondeterministic cell-probes for data structures, we introduce a new concept, **cell-probe proofs**, which formalizes the notion of proofs and verifications in the cell-probe model. For a specific data structure problem f, a cell-probe proof system (CPP) may be defined for f as described below.

We can think of a cell-probe proof system as a game played between an honest verifier and an untrusted prover. Both of them have unlimited computational power. Given an instance of data, a table of cells is honestly constructed according to the rules known to both prover and verifier. Both the prover and the verifier know the query, but only the prover can observe the whole table and thus knows the data. The prover tries to convince the verifier about the result of the query to the data by revealing certain cells. After observing the revealed cells, the verifier either decides the correct answer, or rejects the proof, but can not be tricked by the prover into returning a wrong answer.

Formally, a cell-probe proof system (CPP) consists of three parts:

- A table structure $T : Y \times I \to \{0,1\}^b$. For any data y, a table $T_y : I \to \{0,1\}^b$ is a mapping from indices of cells to their contents.
- A prover P. For every x and y, $P_{xy} \subseteq I$ is a set of cells. We refer to P_{xy} as a **proof** and $\{\langle i, T_y(i)\rangle \mid i \in P_{xy}\}$ as a **certificate**.
- A verifier v, which maps the queries with the certificates to the answers $\{0, 1, \perp\}$. Given an instance of data y, for any query x, both of the following conditions hold:

$$\text{(Completeness)}\ \exists P_{xy} \subseteq I : v(x, \{\langle i, T_y(i)\rangle \mid i \in P_{xy}\}) = f(x, y), \text{ and}$$
$$\text{(Soundness)}\quad \forall P' \subseteq I :\ v(x, \{\langle i, T_y(i)\rangle \mid i \in P'\}) = \begin{cases} f(x, y) \\ \perp \end{cases}.$$

An (s, b, t)-CPP is a CPP such that for every x and y: (1) the table has s cells, i.e. $|I| = s$; (2) each cell contains b bits; (3) each proof consists of t cell probes, i.e. $|P_{xy}| = t$.

Example: For the membership problem[11], where $X = [m]$ and $Y = \binom{[m]}{n}$, and $f(x, y) = 1$ if and only if $x \in y$, a naive construction shows a 2-cell proof: with a sorted table storing y, if $x \in y$, the proof is the cell that contains x, if $x \notin y$, the proof consists of two consecutive cells which are the predecessor and successor of x. The same CPP also works for predecessor search[2].

The notion of cell-probe proofs captures the necessary information to answer queries, and characterizes the nondeterministic probes in the cell-probe model. It is natural to see that for a cell-probe scheme, for any query, the cells probed by an adaptive algorithm contain a cell-probe proof. This can be seen as a data structure counterpart of $P \subseteq NP$.

It is important to note that although a data structure problem is nothing but a boolean function, CPP is very different from the certificate complexity of boolean functions [4]. In CPP, the prover and the verifier communicate with each other via a table structure, which distinguishes CPP from standard certificate complexity. For any data structure problem, the table structure can always store the results for all queries, making one cell-probe sufficient to prove the result, which is generally impossible in the model of certificate complexity.

Unlike adaptive cell-probes, CPP has a static nature, which is convenient for reductions. As stated by the following lemma, any CPP can be trivially reduced to 1-cell proofs.

Lemma 1 (reduction lemma). *For any data structure problem f, if there exists an (s, b, t)-CPP, then there exists an $(s^t, bt, 1)$-CPP.*

Proof. Just store every t-tuple of cells in the (s, b, t)-CPP as a new cell in the $(s^t, bt, 1)$-CPP. □

3 Characterization of CPPs

We now introduce a combinatorial characterization of CPP. Given a set system $\mathcal{F} \subseteq 2^Y$, for any $y \in Y$, we let $\mathcal{F}(y) = \{F \in \mathcal{F} \mid y \in F\}$. For convenience, for a partition $\mathcal{P}$ of Y, we abuse this notation and let $\mathcal{P}(y)$ denote the set $F \in \mathcal{P}$ that $y \in F$.

Definition 1. *We say a set system $\mathcal{F} \subseteq 2^Y$ is an $s \times k$-partition of Y, if $\mathcal{F}$ is a union of s number of partitions of Y, where the cardinality of each partition is at most k.*

This particular notion of partitions of Y fully captures the structure of cell-probe proofs. In this extended abstract, we only provide the characterization of 1-cell proofs. As shown by Lemma 1, this is a canonical case for cell-probe proofs.

Theorem 1. *There is an $(s, b, 1)$-CPP for $f : X \times Y \to \{0, 1\}$, if and only if there exists an $s \times 2^b$-partition $\mathcal{F}$ of Y, such that for every $x \in X$ and every $y \in Y$, there is an $F \in \mathcal{F}(y)$ that $|f(x, F)| = 1$.*

Proof. ($\Longrightarrow$) Given a table structure $T : Y \times I \to \{0,1\}^b$, define the map of the table structure as a $s \times 2^b$ matrix M such that $M_{ij} = \{y \in Y \mid T_y(i) = j\}$, i.e. M_{ij} is the set of such data set y that the content of the i's cell of the table storing y is j. It is clear that each row i of M is a partition of Y with at most 2^b partition sets, because each data set y has one and only one value of $T_y(i)$, and there are at most 2^b possible values for a cell, therefore the matrix M is an $s \times 2^b$-partition $\mathcal{F}$ of Y, where each M_{ij} is an $F \in \mathcal{F}$.

If there is an $(s, b, 1)$-CPP of f, due to the completeness of CPP, for every $x \in X$ and every $y \in Y$, there exists a cell i that $\langle i, j\rangle$ becomes the certificate where $j = T_y(i)$, and due to the soundness of CPP, there must not be any other $y' \in Y$ such that $T_{y'}(i) = j$ and $f(x, y') \neq f(x, y)$. Note that by definition of M, M_{ij} contains all y' such that $T_{y'}(i) = j$, thus $|f(x, M_{ij})| = 1$.

($\Longleftarrow$) Assuming that $\mathcal{F}$ is an $s \times 2^b$-partition of Y such that for every x and every y there is an $F \in \mathcal{F}(y)$ that $|f(x, F)| = 1$, we rewrite $\mathcal{F}$ in the form of an $s \times 2^b$ matrix M that M_{ij} is the $F \in \mathcal{F}$ which is indexed as the jth partition set in the ith partition. We can define our table structure $T : Y \times I \to \{0,1\}^b$ in the way that $T_y(i)$ is assigned with the unique j that $y \in M_{ij}$. Because each row of M is a partition of Y, such T is well-defined.

For every $x \in X$ and every $y \in Y$, there is an $F \in \mathcal{F}(y)$ that $|f(x, F)| = 1$, i.e. there is an $M_{ij} \ni y$ that $|f(x, M_{ij})| = 1$, then we use $\langle i, j\rangle$ as the certificate. Since for every x and y, there exists such i, the corresponding CPP is complete, and since $f(x, \cdot)$ is constant on such M_{ij}, the CPP is also sound. □

Let $Y_0^x = \{y \in Y \mid f(x, y) = 0\}$ and $Y_1^x = \{y \in Y \mid f(x, y) = 1\}$. An alternative characterization is that there is a $(s, b, 1)$-CPP for a problem $f : X \times Y \to \{0,1\}$, if and only if there exists an $s \times 2^b$-partition $\mathcal{F}$ of Y, such that $\{Y_0^x, Y_1^x\}_{x \in X}$ is contained by the union-closure of $\mathcal{F}$. Note that this statement is equivalent to the statement in Theorem 1, so we state it without proof. With this formulation, we get some intuition about 1-cell proofs, that is, a problem $f : X \times Y \to \{0,1\}$ has simple proofs, if and only if there exists some set system $\mathcal{F} \subseteq 2^Y$ with a simple structure, such that the complexity of $\mathcal{F}$ matches the complexity of the problem.

4 Nearest Neighbor Search

We consider the decision version of nearest neighbor search, λ-near neighbor (λ-NN), in a high dimensional Hamming cube $\{0,1\}^d$. Here $X = \{0,1\}^d$, $Y = \binom{\{0,1\}^d}{n}$ and $f(x, y) \in \{0,1\}$ answers whether there exists a point in y within distance λ from the x. As in [3,1], we assume that $d = \omega(\log n) \cap n^{o(1)}$ to make the problem non-trivial.

We prove that with the above setting, there does not exist a $(\mathrm{Poly}(n), n^{o(1)}, 1)$-CPP for the λ-NN problem, thus due to Lemma 1, a super-constant lower bound holds for the problem. To show this, we show the same lower bound for the partial match problem [9,8], which is an instantiation of the λ-NN problem as shown in [3].

The partial match problem is defined as follow: The domain is a Hamming cube $\{0,1\}^d$, where $d = \omega(\log n) \cap n^{o(1)}$, and each data instance y is a set of n points

from the domain, i.e. $Y = \binom{\{0,1\}^d}{n}$. The set of queries is $X = \{0,1,*\}^d$. Given a data instance $y \in \binom{\{0,1\}^d}{n}$ and a query $x \in \{0,1,*\}^d$, $f(x,y) = 1$ if and only if there is a $z \in y$ such that z matches x except for the bits assigned with "$*$".

Theorem 2. *There is no $(s,b,1)$-CPP for the partial match problem, if $s = Poly(n)$ and $b = n^{o(1)}$.*

Proof. We denote the problem as f. From the characterization of $(s,b,1)$-CPP given in Theorem 1, it is sufficient to show that for any $s \times 2^b$ partition $\mathcal{F}$ of Y, there exist $x \in X$ and $y \in Y$ such that for all $F \in \mathcal{F}(y)$, $|f(x,F)| = 2$. We prove this with the probabilistic method. With some distribution of x and y, we show that for any $s \times 2^b$ partition $\mathcal{F}$ of Y, $\Pr[\forall F \in \mathcal{F}(y), |f(x,F)| = 2] > 0$.

For the rest of the proof, we assume that y is uniformly selected from Y, and x is generated by uniformly choosing $r = 2\log n$ bits and fixing each of them uniformly and independently at random with 0 or 1, and setting the other bits to "$*$".

We then prove two supporting lemmas. Recall that for a partition $\mathcal{P}$ of Y, $\mathcal{P}(y)$ denotes the set $F \in \mathcal{P}$ that $y \in F$.

Lemma 2. *For any partition $\mathcal{P}$ of Y, if $|\mathcal{P}| \le 2^b$, where $b = n^{o(1)}$, then*

$$\Pr_y\left[|\mathcal{P}(y)| \le \binom{2^d}{n} \Big/ 2^{n^{\Omega(1)}}\right] \le n^{-\omega(1)}.$$

Proof. We let $\mathcal{P} = \{F_1, F_2, \ldots, F_k\}$ where $k \le 2^b$, and let $p_i = |F_i|/|Y|$. Because $\mathcal{P}$ is a partition of Y, we know that $\sum_i p_i = 1$. We define a random variable $Z = |\mathcal{P}(y)|/|Y|$. Since y is picked uniformly at random from Y, it holds that $Z = p_i$ with probability p_i. Since there are at most 2^b different $\mathcal{P}(y)$, by union bound,

$$\begin{aligned}\Pr_y\left[|\mathcal{P}(y)| \le \binom{2^d}{n} \Big/ 2^{n^{\Omega(1)}}\right] &\le 2^b \cdot \Pr\left[Z = p_i \text{ where } p_i \le 2^{-n^{\Omega(1)}}\right] \\ &= 2^{b-n^{\Omega(1)}} \\ &= n^{-\omega(1)}.\end{aligned}$$

□

For simplicity, we generalize the notation of f to arbitrary point set $A \subseteq \{0,1\}^d$, where $f(x,A)$ is conventionally defined to indicate whether there is a $z \in A$ that matches x

Lemma 3. *For any $A \subseteq \{0,1\}^d$, if $|A| > (1-2^{-k})2^d$ for $k = \frac{1}{2}\log n$, then*

$$\Pr_x[f(x,A) = 0] \le n^{-\omega(1)}.$$

Proof. We let $B = \{0,1\}^d \backslash A$ be the complement of A in the d-dimensional cube. Note that $|B| < 2^{d-k}$. According to our definition of the distribution of x, x is in fact a random $(d-r)$-dimensional subcube in $\{0,1\}^d$, and $f(x,A) = 0$ only

if the cube specified by x is contained in B. This chance is maximized when B itself is a cube. Thus without loss of generality, we can assume that B is the set of $z \in \{0,1\}^d$ whose first k bits are all ones. Therefore,

$$\Pr_x[f(x,A)=0] \le \Pr_x[x\text{'s first } k \text{ bits are all ones}] \le \frac{\binom{d-k}{r-k}}{\binom{d}{r}} \le \left(\frac{r}{d}\right)^k = n^{-\omega(1)}. \quad \square$$

We then prove that for all $s \times 2^b$ partitions $\mathcal{F}$ of Y, the probability $\Pr[\exists F \in \mathcal{F}(y), f(x,F)=\{1\}]$ and $\Pr[\exists F \in \mathcal{F}(y), f(x,F)=\{0\}]$ are both very small.

For any $F \in \mathcal{F}(y)$, we have $y \in F$, thus $\exists F \in \mathcal{F}(y), f(x,F)=\{1\}$ implies that $f(x,y)=1$, therefore for an arbitrary $s \times 2^b$ partition $\mathcal{F}$ of Y,

$$\begin{aligned}\Pr_{x,y}[\exists F \in \mathcal{F}(y), f(x,F)=\{1\}] &\le \Pr_{x,y}[f(x,y)=1] \\ &\le \Pr_{x,y}[\exists z \in y, x \text{ matches } z] \\ &\le n \cdot 2^{-r} \\ &= o(1).\end{aligned}$$

To bound the probability $\Pr[\exists F \in \mathcal{F}(y), f(x,F)=\{0\}]$, we observe that each $s \times 2^b$ partition $\mathcal{F}$ is just a union of s many partitions of Y, each of which is with cardinality at most 2^b, therefore, by union bounds, it holds that

$$\Pr_{x,y}[\exists F \in \mathcal{F}(y), f(x,F)=\{0\}] \le s \cdot \Pr_{x,y}[f(x,\mathcal{P}(y))=\{0\}]. \tag{1}$$

for some partition $\mathcal{P}$ of Y where $|\mathcal{P}| \le 2^b$. It is then sufficient to show that for arbitrary such partition $\mathcal{P}$, the probability $\Pr[f(x,\mathcal{P}(y))=\{0\}]$ is very small.

We choose a threshold $k = \frac{1}{2}\log n$, and separate the case that $|\mathcal{P}(y)| \le \binom{(1-2^{-k})2^d}{n}$ and the case that $|\mathcal{P}(y)| > \binom{(1-2^{-k})2^d}{n}$. According to Lemma 2, for any partition $\mathcal{P}$ of Y with $|\mathcal{P}| \le 2^b$, the probability that $|\mathcal{P}(y)| \le \binom{(1-2^{-k})2^d}{n} = \binom{2^d}{n}/2^{n^{\Omega(1)}}$ is at most $n^{-\omega(1)}$.

We let $A_y = \bigcup \mathcal{P}(y) = \bigcup_{y' \in \mathcal{P}(y)} y'$. Note that $A_y \subseteq \{0,1\}^d$, and $f(x,\mathcal{P}(y))=\{0\}$ implies that $f(x,A_y)=0$. For such $\mathcal{P}(y)$ that $|\mathcal{P}(y)| > \binom{(1-2^{-k})2^d}{n}$, by the Pigeonhole Principle, it holds that $|A_y| \ge (1-2^{-k})2^d$. Due to Lemma 3, $f(x,A_y)=0$ with prohibitively small probability. Putting these together, it holds for an arbitrary partition $\mathcal{P}$ of Y with $|\mathcal{P}| \le 2^b$ that

$$\begin{aligned}\Pr_{x,y}[f(x,\mathcal{P}(y))=\{0\}] &\le \Pr_y\left[|\mathcal{P}(y)| \le \binom{(1-2^{-k})2^d}{n}\right] \\ &\quad + \Pr_{x,y}\left[f(x,\mathcal{P}(y))=\{0\} \;\middle|\; |\mathcal{P}(y)| > \binom{(1-2^{-k})2^d}{n}\right] \\ &\le n^{-\omega(1)} + \Pr_x\left[f(x,A_y)=0 \;\middle|\; |A_y| > (1-2^{-k})2^d\right] \\ &\le n^{-\omega(1)}.\end{aligned}$$

Combining with (1), we have that

$$\Pr_{x,y}[\exists F \in \mathcal{F}(y), f(x,F) = \{0\}] \leq s \cdot n^{-\omega(1)} = o(1)\,.$$

Therefore, for an arbitrary $s \times 2^b$ partition $\mathcal{F}$ of Y, it holds that

$$\begin{aligned}\Pr_{x,y}[\forall F \in \mathcal{F}(y), |f(x,F)| = 2] &\geq 1 - \Pr_{x,y}[\exists F \in \mathcal{F}(y), f(x,F) = \{1\}] \\ &\quad - \Pr_{x,y}[\exists F \in \mathcal{F}(y), f(x,F) = \{0\}] \\ &\geq 1 - o(1)\,.\end{aligned}$$

It follows that for any $s \times 2^b$ partition $\mathcal{F}$ of Y, where $s = \mathrm{Poly}(n)$ and $b = n^{o(1)}$, there exist $x \in X$ and $y \in Y$ such that for every $F \in \mathcal{F}(y)$, it holds that $|f(x,F)| = 2$. By Theorem 1, there is no $(s, b, 1)$-CPP for f with the above range of s and b. □

In [3], it is shown that the partial match problem can be reduced to the λ-NN problem. Because the reduction only involves mapping between instances of problems, the existence of an $(s, b, 1)$-CPP for λ-NN implies the existence of a CPP for partial match with essentially the same parameters. The following corollary is implied.

Corollary 1. *There does not exist a $(Poly(n), n^{o(1)}, 1)$-CPP for the nearest neighbor search problem with n points in d-dimensional Hamming space where $d = \omega(\log n) \cap n^{o(1)}$.*

Due to Lemma 1, the following super-constant lower bound on the nondeterministic cell-probe complexity holds.

Corollary 2. *There does not exist a $(Poly(n), n^{o(1)}, O(1))$-CPP for the nearest neighbor search problem or the partial match problem with n points in d-dimensional Hamming space where $d = \omega(\log n) \cap n^{o(1)}$.*

5 Polynomial Evaluation

Let 2^k be a finite field. Let $Y = 2^{kd}$ be the set of all polynomials of degree $\leq (d-1)$ over the finite field 2^k. Throughout this section, we assume that $d \leq 2^k$.

Let $X = 2^{2k}$ be the set of all pairs of elements of the finite field 2^k. A decision version of the polynomial evaluation problem f is defined as: for every query $(x, z) \in X$ and every data instance $g \in Y$, $f((x,z), g) = 1$ if $g(x) = z$ and $f((x,z), g) = 0$ otherwise. Intuitively, a polynomial g is preprocessed and stored as a data structure, so that for each query (x, z), the data structure answers whether $g(x) = z$.

There are two naive upper bounds for one-cell proofs:

1. A $(1, kd, 1)$-CPP: store the whole polynomial in a single cell, and on each query, one probe reveals the whole polynomial;

2. A $(2^k, k, 1)$-CPP: each cell corresponds to an input x, and the cell stores the value of $g(x)$, thus on each query (x, z), one probe to the cell corresponding to x answers whether $g(x) = z$.

We are going to prove that the above naive upper bounds are essentially optimal for single-probe proofs. We show that for any $(s, b, 1)$-CPP, either b is close to large enough to store a whole polynomial as in (1), or the total storage size $s \cdot b$ is exactly as large as in (2).

We first prove two lemmas. For any subset $P \subseteq Y$, let $\tau(P) = |\{x \in 2^k \mid \forall g_1, g_2 \in P, g_1(x) = g_2(x)\}|$, which represents the number of such assignments of x that all polynomials in P yield the same outcome. It is trivial to see that for $|P| \leq 1$, $\tau(P) = 2^k$.

Lemma 4. *If $|P| > 1$, it holds that*

$$\tau(P) \leq d - \frac{\log |P|}{k}.$$

Proof. We write $\tau(P)$ briefly as τ. Let $x_1, x_2, \ldots, x_\tau$ be such that all polynomials in P yield the same outcomes. We arbitrarily pick other $x_{\tau+1}, x_{\tau+2}, \ldots, x_d$. For any two different polynomials $g_1, g_2 \in P$, it can never hold that $g_1(x_i) = g_2(x_i)$ for all $i = \tau + 1, \tau + 2, \ldots, d$, since if otherwise, $g_1 \equiv g_2$ by interpolation. Recall that g is a polynomial over the finite field 2^k, thus for an arbitrary $g \in P$ and an arbitrary x, there are at most 2^k possible values for $g(x)$. Therefore, due to Pigeonhole Principle, in order to guarantee that no two polynomials in P agree on all $x_{\tau+1}, x_{\tau+2}, \ldots, x_d$, it must hold that $2^{k(d-\tau)} \geq |P|$, i.e. $\tau(P) \leq d - \frac{\log |P|}{k}$. □

Lemma 5. *Given a partition $\mathcal{P}$ of Y, let g be a uniformly random polynomial in Y. $E\{\tau(\mathcal{P}(g))\}$ represents the expected number of the input xs such that all polynomials in the partition block $\mathcal{P}(g)$ yield the same outcome, where the expectation is taken over random g. For any partition $\mathcal{P}$ of Y such that $|\mathcal{P}| \leq 2^b$ and $b < k(d-1) - \log k$, it holds that*

$$E\{\tau(\mathcal{P}(g))\} \leq \frac{b}{k}.$$

Proof. Let $P_1, P_2, \ldots, P_{2^b}$ denote the partition blocks, and let $q_1, q_2, \ldots, q_{2^b}$ be the respective cardinalities. Naturally we have that $\sum_{i=1}^{2^b} q_i = 2^{kd}$. We assume that $q_i = 0$ for $i = 1, 2, \ldots, m_0$, $q_i = 1$ for $i = m_0 + 1, m_0 + 2, \ldots, m$, and $q_i > 1$ for $i > m$. For those P_i that $i \leq m$, $|P_i| = q_i \leq 1$, thus $\tau(P_i) = 2^k$. According to Lemma 4,

$$\begin{aligned} E\{\tau(\mathcal{P}(g))\} &= \sum_{i=1}^{m_0} \frac{0}{2^{kd}} \tau(P_i) + \sum_{i=m_0+1}^{m} \frac{1}{2^{kd}} \tau(P_i) + \sum_{i=m+1}^{2^b} \frac{q_i}{2^{kd}} \tau(P_i) \\ &\leq (m - m_0) \cdot \frac{2^k}{2^{kd}} + \sum_{i=m+1}^{2^b} \frac{q_i}{2^{kd}} \left(d - \frac{\log q_i}{k} \right). \end{aligned} \tag{2}$$

Recall that $\sum_{i=m+1}^{2^b} q_i = 2^{kd} - \sum_{i=1}^{m} q_i = 2^{kd} - m + m_0$. According to Lagrange multipliers, (2) is maximized when all q_i for $i = m+1, m+2, \ldots, 2^b$ are equal. Thus (2) is less than or equal to

$$\frac{m - m_0}{2^{k(d-1)}} + \frac{2^{kd} - m + m_0}{2^{kd}} \left(d - \frac{\log(2^{kd} - m + m_0) - \log(2^b - m)}{k} \right) .$$

Let $\epsilon = \frac{m - m_0}{2^{kd}}$. The above formula becomes

$$\begin{aligned} & 2^k \epsilon + (1 - \epsilon) \left(d - \frac{\log 2^{kd}(1-\epsilon) - \log 2^b (1 - 2^{-b}(2^{kd}\epsilon + m_0))}{k} \right) \\ \leq\ & 2^k \epsilon + \frac{1}{k}(1-\epsilon) \left(b + \log(1 - 2^{kd-b}\epsilon) - \log(1-\epsilon) \right) . \end{aligned}$$

Note that $0 \leq \epsilon < 2^{b-kd}$. By standard analysis, if $b < k(d-1) - \log k$, the above function of ϵ is maximized when $\epsilon = 0$, i.e. $E\{\tau(\mathcal{P}(g))\} \leq \frac{b}{k}$. □

With the above lemmas, we can prove the following theorem.

Theorem 3. *For any $(s, b, 1)$-CPP for the polynomial evaluation problem with parameters k and d where $d \leq 2^k$, either $b \geq k(d-1) - \log k$ or $s \cdot b \geq k \cdot 2^k$.*

Proof. We will prove that there does not exist an $(s, b, 1)$-CPP for the polynomial evaluation problem if $b < k(d-1) - \log k$ and $s \cdot b < k \cdot 2^k$.

Let x be a uniformly random element of 2^k, and let g be a uniformly random polynomial from Y. For any partition $\mathcal{P}$ of Y that $|\mathcal{P}| \leq 2^b$, according to Lemma 5,

$$\Pr_{x,g}[\forall g_1, g_2 \in \mathcal{P}(g), g_1(x) = g_2(x)] = \frac{1}{2^k} E\{\tau(\mathcal{P}(g))\} \leq \frac{b}{k 2^k} .$$

Therefore, for any $s \times 2^b$ partition $\mathcal{F}$ of Y, it holds that,

$$\begin{aligned} & \Pr_{x,g}[\exists F \in \mathcal{F}(g), \forall g_1, g_2 \in F, g_1(x) = g_2(x)] \\ \leq\ & s \cdot \Pr_{x,g}[\forall g_1, g_2 \in \mathcal{P}(g), g_1(x) = g_2(x)] \\ \leq\ & \frac{s \cdot b}{k 2^k} \\ <\ & 1 , \end{aligned}$$

where the first inequality is due to the observation that $\mathcal{F}$ is a union of s instances of 2^b-partitions of Y. Therefore, for any $s \times 2^b$ partition $\mathcal{F}$ of Y,

$$\Pr_{x,g}[\forall F \in \mathcal{F}(g) \exists g_1, g_2 \in F, g_1(x) \neq g_2(x)] > 0 .$$

By probabilistic methods, we know that for any $s \times 2^b$ partition $\mathcal{F}$ of Y, there exists some $(x, z) \in X$ and some $g \in Y$ such that $g(x) = z$, but for all $F \in \mathcal{F}(g)$, there exists $h \in F$ such that $h(x) \neq z$.

According to Theorem 1, we know that there does not exist $(s, b, 1)$-CPP with the given range of s and b. □

Acknowledgment. I would like to thank James Aspnes for helpful discussions and editing assistance, and Dana Angluin for her comments on an early version of the paper.

References

1. Barkol, O., Rabani, Y.: Tighter lower bounds for nearest neighbor search and related problems in the cell probe model. Journal of Computer and System Sciences 64(4), 873–896 (2002)
2. Beame, P., Fich, F.: Optimal bounds for the predecessor problem and related problems. Journal of Computer and System Sciences 65(1), 38–72 (2002)
3. Borodin, A., Ostrovsky, R., Rabani, Y.: Lower bounds for high dimensional nearest neighbor search and related problems. In: Proceedings of the thirty-first annual ACM Symposium on Theory of Computing, pp. 312–321 (1999)
4. Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: a survey. Theoretical Computer Science 288(1), 21–43 (2002)
5. Demaine, E., Pătraşcu, M.: Logarithmic lower bounds in the cell-probe model. SIAM Journal of Computing 35(4), 932–963 (2006)
6. Fredman, M., Saks, M.: The cell probe complexity of dynamic data structures. In: Proceedings of the twenty-first annual ACM Symposium on Theory of Computing, pp. 345–354 (1989)
7. Husfeldt, T., Rauhe, T.: Hardness results for dynamic problems by extensions of Fredman and Saks' chronogram method. In: Proceedings of the 25th International Colloquium on Automata, Languages and Programming, pp. 67–78 (1998)
8. Indyk, P., Goodman, J., O'Rourke, J.: Nearest neighbors in high-dimensional spaces. In: Handbook of Discrete and Computational Geometry, ch. 39 (2004)
9. Jayram, T., Khot, S., Kumar, R., Rabani, Y.: Cell-probe lower bounds for the partial match problem. Journal of Computer and System Sciences 69(3), 435–447 (2004)
10. Miltersen, P., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. Journal of Computer and System Sciences 57(1), 37–49 (1998)
11. Yao, A.: Should tables be sorted? Journal of the ACM 28(3), 615–628 (1981)

Constructing Efficient Dictionaries in Close to Sorting Time*

Milan Ružić

ITU Copenhagen, Denmark
milan@itu.dk

Abstract. The dictionary problem is among the oldest problems in computer science. Yet our understanding of the complexity of the dictionary problem in realistic models of computation has been far from complete. Designing highly efficient dictionaries without resorting to use of randomness appeared to be a particularly challenging task. We present solutions to the static dictionary problem that significantly improve the previously known upper bounds and bring them close to obvious lower bounds. Our dictionaries have a constant lookup cost and use linear space, which was known to be possible, but the worst-case cost of construction of the structures is proportional to only $\log\log n$ times the cost of sorting the input. Our claimed performance bounds are obtained in the word RAM model and in the external memory models; only the involved sorting procedures in the algorithms need to be changed between the models.

1 Introduction

Dictionaries are among the most fundamental data structures. A dictionary stores a set S which may be any subset of *universe* U, and it answers membership queries of type "Is x in S?", for any $x \in U$. The elements of S may be accompanied by *satellite data* which can be retrieved in case $x \in S$. The size of the set S is standardly denoted by n.

We consider universes whose elements can be viewed as integers or binary strings. In this paper we concentrate on *static* dictionaries — a static dictionary is constructed over a given set S that remains fixed. *Dynamic* dictionaries allow further updates of S through insertions and deletions of elements. Even static dictionaries are sometimes used as stand-alone structures, but more often they appear as components of other algorithms and data structures, including dynamic dictionaries.

The dictionary problem has been well studied and many solutions have been given. They offer different characteristics regarding space usage, time bounds, model of computation, and universe in question. A challenge is to simultaneously achieve good characteristics on all the terms. We consider only dictionaries with realistic space usage of $O(n)$ registers of size $\Theta(\log|U|)$ bits. In the usual case

* Extended abstract. Additional details and proofs can be found in a version on the author's web page.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 84–95, 2008.

when $|U|$ is at least polynomially larger than n this amount of space is necessary (ignoring constant factors) regardless of presence of satellite data. Algorithms involved in construction of a dictionary may be randomized — they require a source of random bits and their time bounds are either *expectations* or hold with high probability. Randomized dictionaries reached a stage of high development and theoretically there is little left to be improved. On the other hand, the progress on deterministic dictionaries was much slower. While in the dynamic case we have some reason to believe that there is a considerable gap between attainable worst-case performance for deterministic dictionaries and the attainable expected performance for randomized dictionaries, there is not any evidence of a required gap in the static case.

A theoretical interest in deterministic dictionaries comes from the question of what resources are necessary to implement an efficient dictionary structure — and random bits are a resource. Having guaranteed time bounds, deterministic structures can be used in systems with strict performance demands. A sufficiently simple deterministic dictionary having comparable performance to a randomized dictionary would make the randomized structure obsolete. Unfortunately, the new solutions described here are not simple enough to be competitive in practice, except possibly in some special cases.

In this paper we focus on dictionaries with constant lookup time. Because of faster construction time, dictionaries with slightly slower lookups may sometimes be of interest. For example, a structure supporting searches in time $O(\log \log n)$ can be built in linear time on sorted input [17].

1.1 The Word RAM Model and Related Work

The word RAM is a common computational model in data structures literature. It has the machine word size of w bits and a standard instruction set, resembling the primitive instructions of the language C. Execution of any instruction takes one unit of time. A usual assumption for RAM dictionaries is that the elements of U fit in one machine word[1]. Contents of a word may be interpreted either as an integer from $\{0, \ldots, 2^w - 1\}$ or as a bit string from $\{0,1\}^w$. For more information see, e.g., [11].

We will list some important results for deterministic dictionaries with constant query time. Each of those results required a different idea, and a new insight into properties and possibilities of some family of hash functions. A seminal work by Fredman, Komlós, and Szemerédi [8] showed that in the static case it is possible to construct a linear space dictionary with a constant lookup time for arbitrary word sizes (no assumptions about relative values of w and n). This dictionary implementation is known as the *FKS scheme*. Besides a randomized version with expected $O(n)$ construction time, they also gave a deterministic construction algorithm with a running time of $O(n^3 w)$. A bottleneck was choosing of appropriate hash functions. Any *universal* family of hash functions [6] contains

[1] This assumption simplifies analysis. Some schemes, including ours, scale well when keys are multi-word strings.

functions suitable for use in the FKS scheme. Raman [16] devised a deterministic algorithm for finding good functions from a certain universal family, with a running time of $O(n^2w)$; this implies the same time bound for construction of the FKS dictionary. For $w = n^{\Omega(1)}$, an efficient static dictionary can be built in time $O(n)$ on a sorted sequence of keys. This follows from a generalization of the fusion trees of Fredman and Willard [9], and it was observed by Hagerup [11]. The previously fastest deterministic dictionary with constant lookup time is a result of Hagerup, Miltersen and Pagh [12]. Their construction method has a running time of $O(n \log n)$. There exists an issue with compile-time computation of a special constant that is required for each w, because the only known computation method is a brute-force search that takes time $2^{\Omega(w)}w$.

Allowing randomization, the FKS scheme can be dynamized to support updates in amortized expected constant time [7]. The lower bound result in the same paper states that a deterministic dynamic dictionary, based on *pure* hashing schemes, with worst-case lookup time of $t(n)$ must have amortized insertion time of $\Omega(t(n) \cdot n^{1/t(n)})$ (this lower bound does not hold in general however, e.g. see the result of Pagh [15]). A standard dynamization technique [14] applied to the static dictionary from [12] yields a similar type of trade-offs: lookups in time $O(t(n))$, insertions in time $O(n^{1/t(n)})$, and deletions in time $O(\log n)$, where t is a "reasonable" parameter function. The method in [18] was devised as an alternative to the method from [12] that eliminates the problem with the high compile-time demand. The dynamic dictionaries from [18] almost match the dynamic result from [12] — the difference is that update bounds are amortized, instead of worst-case. An illustration of complete independence from the word size w is that the structures from [18] can be easily adapted to the *Real RAM* model to work with arbitrary real numbers. This is not a feature of any other known hashing method.

1.2 External Memory Models

Real computers don't have one plain level of memory but a memory hierarchy. Transfers of data between levels of memory are often a dominant term in execution times. The theoretical I/O-model was introduced to model behavior of algorithms in such a setting [1]. The I/O-model was generalized to the cache-oblivious model [10], where the algorithm does not know the size of the internal memory M and the block size B. That is, the analysis of an algorithm should be valid for any values of B and M. Comparison-based sorting of n integers (which occupy one memory cell) takes $\Theta(\mathrm{Sort}(n))$ I/Os [10], where $\mathrm{Sort}(n) = \frac{n}{B} \log_{M/B} \frac{n}{B}$.

From the structures mentioned for the word RAM model, it can be observed that the methods of Raman [16] and Ružić [18] can easily be adapted to the external memory models and attain analogous bounds — respectively $O(\frac{n^2}{B} \log |U|)$ and $O(\frac{n^{1+\epsilon}}{B})$ I/Os. We take the block size parameter B to represent the number of $\lg |U|$-bit items that can fit in a memory block. For the dictionary from [12], no better bound than $O(n \log n)$ I/Os can be stated. The main problem was with the dictionary for universes of size polynomial in n, which is a component of the construction from [12].

1.3 Background of Our Techniques

Our contribution consists of two parts. One part is a very efficient dictionary for universes of size $n^{O(1)}$. Beside its use in composition with methods that perform *universe reduction*, this case has a significance of its own. The most prominent example of stand-alone use of dictionaries for "small" universes is representation of a graph. In the problem of storing and (random) accessing edges of a graph, the universe is of quadratic size. The problem is also of interest to some situations in practice, since in reality integer keys are not often very large relative to n. The main part of our structure uses the same kind of hash functions that were used in [12] for this case. The construction algorithm from [12] runs in time $\Theta(n \log n)$. Interestingly, those functions are very similar to the functions from [19] where construction time was $\Theta(n^2)$. We devised a different and more efficient construction algorithm.

The other part of the contribution is a follow-up on our technique of making deterministic signatures from [17]. That paper introduced a new type of hash functions and associated algorithms for injectively mapping a given set of keys to a set of signatures of $O(\log n)$ bits (a brief outline of the method is given in Section 3.1). The methods are computationally efficient in various models of computation, especially for keys of medium to large lengths. More precisely, when given keys have a length of at least $\log^{3+\epsilon} n$ bits, the algorithms for selecting perfect hash functions have a linear running cost on sorted input. The performance of evaluation is optimal[2]. Those functions have rather succinct descriptions, and they might have an application outside of dictionary structures. In our quest for a faster construction in the case $w = \log^{O(1)} n$ we will give up the requirement of complete injectiveness, and replace it with considerably weaker and rather specific properties. These weaker functions will be meaningful only within our dictionary construction.

1.4 Our Results

The result for the case of universes of polynomial size is summarized in the following theorem.

Theorem 1. *Suppose that a set of n integers from the universe $\{0, 1, \ldots, n^{O(1)}\}$ is given. A static linear space dictionary on that set can be deterministically constructed on a word RAM in time $O(n \log \log n)$, so that lookups to the dictionary take constant time. In the cache-oblivious model, and hence in the I/O model as well, a similar structure can be built using $O(\mathrm{Sort}(n) \log \log n)$ I/Os.*

The method is discussed in Section 2. The obtained structure complements additional results from [17] in the external memory setting, such as a static predecessor structure for variable and unbounded length binary strings.

[2] For the word RAM model, in the conference version evaluation time of $O(\log \log \frac{w}{\log n})$ was stated. However, with even a slightly simpler method the time is cut down to $O(1)$. We make a brief explanation of that in this paper; the explanation will also appear in the journal version of [17].

In the second part of the paper (Section 3) we give an overview of the structures and associated procedures that are efficient in the case that $w = \log^{O(1)} n$. In conjunction with the earlier results, this implies the claimed results for the general case, which are formally expressed in the following theorems. In the performance bounds we plugged in the currently known upper bounds on sorting (which may be optimal).

Theorem 2. *In the cache-oblivious model, a static linear space dictionary on a set of n keys can be deterministically constructed using $O(\mathrm{Sort}(n) \log\log n)$ I/Os, so that lookups to the dictionary take $O(1)$ I/Os.*

Theorem 3. *In the word RAM model, a static linear space dictionary on a set of n keys can be deterministically constructed in time $O(n(\log\log n)^2)$, so that lookups to the dictionary take time $O(1)$.*

We could have also listed results for strings, etc. The stated general bounds do not match the actual times in every case. We make remarks on some meaningful special cases, when performance is better.

Remark 1. Suppose that $\log |U| = \Omega(\log n \log\log n)$. The construction cost of the dictionary referred to in Theorem 2 is $O(\mathrm{Sort}(n))$ I/Os.

Remark 2. Supposing that $w > \log^{3+\epsilon} n$ and that the input set of keys is sorted, time taken to build the dictionary from Theorem 3 is $O(n)$.

Remark 3. Supposing that $w = O(\log n \log\log n)$, time taken to build the dictionary from Theorem 3 is $O(n \log\log n)$.

At the moment, our fast static dictionaries do not yield an improvement for dynamic deterministic dictionaries. It is one of major challenges in data structures research to either significantly improve performance of dynamic dictionaries, or to prove general lower bounds that would definitely establish a gap between deterministic and randomized dictionaries. How far deterministic dictionaries can go remains unknown, even in the static case.

2 Universes of Polynomial Size

Notation and comments. We use the symbol $\oplus$ to denote bitwise exclusive or operation. The number of *collisions* of a function h on a subset A of its domain represents the value $|\{\ \{x, y\}\ :\ h(x) = h(y) \wedge x, y \in A \wedge x \neq y\}|$. For multisets A and B, the value $|\{\ \{x, y\} \in A \times B\ :\ x = y\}|$, which may be thought of as the number of collisions between the multisets, is denoted by $\mathrm{coll}(A, B)$. For a multiset A, $A \oplus y$ stands for the multiset $\{x \oplus y\}_{x \in A}$. Because of space restrictions, we will give only a high-level description of the construction. Explanations of subprocedures and second-level structures will appear in a full version of the paper.

Suppose that $\phi : U \to \{0,1\}^{\Phi}$ and $\psi : U \to \{0,1\}^{\Psi}$ are functions such that the combined function (ϕ, ψ) is 1-1 on U. An easy choice is to take ϕ to be

the projection on the Φ highest order bits, and ψ to be the projection on the Ψ lowest order bits of binary representations of keys. We will assume here that $\lg n \leq \Psi < \lg n + O(1)$ and $\Phi + \lg \Phi + \lg \Psi \leq \Psi$. This assumption implies that in a basic form the structure allows universes of size $O(n^2/(\log n)^2)$. Yet, such a structure can easily be built upon to support universes of size n^c, for any constant c.

The main hash function is of type

$$h(x) = \psi(x) \oplus a_{\phi(x)} \ ,$$

where (a_i) is an array of Ψ-bit elements, with $i \in \{0,1\}^{\Phi}$. Our aim is to set values of the array elements in a way that makes the function h have no more than $3\Phi^2 n$ collisions on a given set $S \subset U$. It will become clear that this is always possible. After the function h is fixed, buckets of elements colliding under h need to be resolved. This is much easier than the original problem, since the average size of buckets is small. If the size of a bucket is less than $\Phi^3\Psi$ then a structure specialized for small sets will handle it. The total number of elements in the remaining ("large") buckets is $O(\frac{n}{\Phi\Psi})$. This can be seen by analyzing the function $\sum_i b_i$ under constraint $\sum_i \binom{b_i}{2} \leq 3\Phi^2 n$ and with variable domains $[\Phi^3\Psi, \ \infty)$. Let S' be the subset of S comprising the elements that fall in the "large" buckets. Constructing an efficient dictionary over S' will be an easier task, because we can afford to spend $O(|S'|\Phi\Psi + 2^{\Psi})$ construction time on it. No additional new techniques are required to design these second-level structures.

We will now give an overview of the algorithm for selecting values of the elements of the array a. The array a is initially set to all-zeros. Values of array elements will be decided in stages, with each stage being responsible for a separate set of bit positions. In our numbering of bit positions, position 0 refers to the most-significant bit position. Let $i_* = \lfloor \lg \Psi - \lg \lg \Phi - 1 \rfloor$. There will be a total of $2i_* + 2 = O(\log \Psi)$ stages. In the stages numbered $1, 2, \ldots, i_*$ the sizes of the *active* sets of bit positions decrease roughly geometrically, while in the remaining $i_* + 2$ stages they have the same (small) size. Let $p_0 = 0$, $p_i = \lfloor (1 - 2^{-i})\Psi \rfloor - i \cdot \lfloor \lg \Phi \rfloor$ for $0 < i \leq i_*$, and $p_i = p_{i-1} + \lfloor \lg \Phi \rfloor$ for $i_* < i \leq 2i_* + 2$. In the ith stage bits at positions between p_{i-1} and $p_i - 1$ (inclusive) are decided on all elements of a.

The last $i_* + 2$ stages can be replaced with different and shorter sequences. Yet, in this presentation of the algorithm we keep the chosen setting because it is relatively simple and incurs a relatively small increase in the overall constant factor. Operation in all the stages is done by the same procedure, parameterized by values p_{i-1} and p_i. We introduce symbols η_i to denote $2^{p_i - p_{i-1} + \lfloor \lg \Phi \rfloor}$.

After the ith stage of the algorithm, the projection of $h(x)$ on the high-order p_i bits is known. In other words, for any $x \in U$ the value of $h(x)$ div $2^{\Psi - p_i}$ is fixed after the ith stage. To describe operation of the algorithm in stage i, we will define sets $T(v, j, k)$, $v \in \{0,1\}^{p_{i-1}}$, $0 \leq j \leq \Phi$, $0 \leq k < 2^{\Phi - j}$ (whenever we talk about sets $T(v, j, k)$ the stage number i is assumed to be fixed). Sets $T(v, j, k)$ are defined recursively as follows:

- For any $v \in \{0,1\}^{p_{i-1}}$, $T(v, \Phi, 0) = \{x \in S \mid h(x) \text{ div } 2^{\Psi - p_{i-1}} = v\}$.

– For $j < \Phi$, if $|T(v, j+1, k \text{ div } 2)| < \eta_i$ then $T(v,j,k) = \emptyset$.
– For $j < \Phi$, if $|T(v, j+1, k \text{ div } 2)| \geq \eta_i$ then

$$T(v,j,k) = \{x \in T(v,\Phi,0) \mid k2^j \leq \phi(x) < (k+1)2^j\} \ .$$

Only non-empty sets $T(v,j,k)$ are of interest to us. For any fixed v, subset relation on the family of non-empty sets $T(v,j,k)$ can be described by a binary tree, with nodes labeled by pairs (j,k). Sets $T(v,j,k)$ that correspond to leaves of that tree are those that satisfy $j = 0$ or $T(v,j-1,2k) \cup T(v,j-1,2k+1) = \emptyset$. Let $\{S_{vl}\}_{v,l}$ be the collection of all such "leaf" sets, over $v \in \{0,1\}^{p_{i-1}}$. The collection $\{S_{vl}\}$ is a partition of the set S.

No matter how the elements of the array a are modified in current and later stages, that is on bit positions from p_{i-1} to $\Psi - 1$, the number of collisions that h may create is bounded by $\sum_v \sum_{l_1<l_2} |S_{vl_1}| \cdot |S_{vl_2}|$ plus a bound on the total number of collisions within the sets S_{vl}. If a set S_{vl} has size greater than η_i then it has to be one of the sets $T(v,0,k)$. However, the set $\{x \in S \mid \phi(x) = k\} \supset T(v,0,k)$ is always mapped injectively by h. This follows from the definition of the function h, the fact that (ϕ, ψ) is 1-1 on U, and the properties of xor operation. Therefore collisions may happen only within the sets S_{vl} such that $|S_{vl}| < \eta_i$. An upper bound on the total number of collisions that may happen within the sets S_{vl} is $\frac{1}{2} n \eta_i$, which can easily be seen by analyzing the function $\frac{1}{2}\sum_{j=1}^{n} b_j^2$ under constraint $\sum_j b_j = n$ and over domain $[0, \eta_i]^n$. The goal of processing in stage i is to modify the values in the array a so that the number of collisions of h on S does not exceed

$$\eta_i \frac{n}{2} + \frac{1}{2^{p_i - p_{i-1}}} \sum_v \sum_{l_1<l_2} |S_{vl_1}| \cdot |S_{vl_2}| \ , \tag{1}$$

when the stage ends. By solving appropriate recurrences, the following technical lemma can be shown.

Lemma 1. *If modifications to the array a by the selection algorithm make the number of collisions of h on S not exceed (1) at the end of stage i, for each i, then the final number of collisions will be less than $3\Phi^2 n$.*

The term $\sum_v \sum_{l_1<l_2} |S_{vl_1}| \cdot |S_{vl_2}|$ from (1) can be re-expressed in an algorithmically more useful form. Each set $T(v,j,k)$ is the union of some sets S_{vl}. Thus, we may write $|T(v,j,k)| = \sum |S_{vl}|$, where the sum is over all l such that $S_{vl} \subset T(v,j,k)$. The product $|S_{vl_1}| \cdot |S_{vl_2}|$, for some l_1, l_2, will be a term in the expanded expression for a product of type $|T(v,j,2k)| \cdot |T(v,j,2k+1)|$. Actually it will appear as a component of exactly one such product — in the mentioned binary tree the node with label $(j+1,k)$ has to be the lowest common ancestor of the nodes that correspond to the sets S_{vl_1} and S_{vl_2}. As a result, it holds that:

$$\sum_v \sum_{l_1<l_2} |S_{vl_1}| \cdot |S_{vl_2}| = \sum_v \sum_{j=0}^{\Phi-1} \sum_{k=0}^{2^{\Phi-j-1}-1} |T(v,j,2k)| \cdot |T(v,j,2k+1)| \ .$$

After we specified the goal of processing in every stage, we proceed to giving a high-level description of the sequence of operations done at each stage. We introduce multiset variables $X(v, j, k)$, and we implicitly initialize all of them to $\emptyset$. In the outermost loop of the procedure, j takes values from 0 to $\Phi - 1$. We describe principal operations performed for a fixed j. First, for all "leaf" sets $T(v, j, k)$, i.e. those that equal one of the sets S_{vl}, we make the assignment

$$X(v, j, k) = \{(h(x) \text{ div } 2^{\Psi - p_i}) \bmod 2^{p_i - p_{i-1}} \mid x \in T(v, j, k)\} \ ,$$

where values $h(x)$ are taken to be determined by the current state of the array a. We effectively calculated the projections of the current values $h(x)$, $x \in T(v, j, k)$, on the bits at positions p_{i-1} through $p_i - 1$. The multisets can be stored as sets of element-multiplicity pairs. For each k, $0 \leq k \leq 2^{\Phi - j - 1} - 1$, the algorithm will find a value $\delta \in \{0, 1\}^{p_i - p_{i-1}}$ such that

$$\sum_v \text{coll}(X(v, j, 2k), X(v, j, 2k+1) \oplus \delta) \leq \frac{1}{2^{p_i - p_{i-1}}} \sum_v |T(v, j, 2k)| \cdot |T(v, j, 2k+1)|$$

and then make assignments $X(v, j+1, k) = X(v, j, 2k) \cup (X(v, j, 2k+1) \oplus \delta)$, where the union is in the multiset sense. The elements of the array a are modified so that $a_l = a_l \ \oplus \ 0^{p_{i-1}} \delta \, 0^{\Psi - p_i}$, for $(2k+1)2^j \leq l < (2k+2)2^j$. At the end of the current iteration of the loop over j, the equality

$$X(v, j+1, k) = \{(h(x) \text{ div } 2^{\Psi - p_i}) \bmod 2^{p_i - p_{i-1}} \mid x \in T(v, j+1, k)\} \ ,$$

holds for every non-leaf set $T(v, j+1, k)$.

It is not hard to formally verify that a procedure conforming with this high-level description meets the specified goal of reducing the number of collisions of the function h on the set S. We leave out the explanations of subprocedures, such as for determining and arranging sets S_{vl}, and for finding suitable δ values. We mention that the following fact is used in the performance analysis.

Lemma 2. *There can be at most* $4n\frac{\Phi+1}{\eta_i}$ *non-empty multisets* $X(v, j, k)$.

3 Larger Universes

3.1 Background on Signature Functions

The basic type of functions used in [17] is $f(x, s, a) = x \text{ div } 2^s + a \cdot (x \bmod 2^s)$, where a is a parameter chosen from $\{1, 2, \ldots, n^c - 1\}$, $c \geq 2$. The parameter s has a value dependent only on the domain of x, for example $s = \lfloor \frac{1}{2} \lg |U| \rfloor$ The integer division and modulo functions were chosen as they are perhaps the simplest of all pairs of functions (ϕ, ψ) such that (ϕ, ψ) is 1-1 on U, and so that both functions map to a (significantly) smaller universe. In a more general form, we write $f(x, a) = \phi(x) + a \cdot \psi(x)$. Suppose that K is the number of keys that can be packed in a machine word. With $c = 3.42$, on a given set of n keys a value for the parameter a that makes the function f injective on the set can be

found in time $O(n(\log n)^2 \frac{\log K}{K} + \log n)$. The basic function can be combined in different ways to achieve larger reduction of universe. The ultimate goal is to have a function that maps original keys to signatures of size $O(\log n)$ bits. One approach is to view keys as strings over some chosen alphabet, and then apply function f on the characters and combine the values in some way. We will see two concrete methods that fall under this approach.

Let $x[i]_\sigma$ denote the ith character of key x viewed as a string over the alphabet $\{0, \ldots, 2^\sigma - 1\}$. If σ is not too small, e.g. $\sigma > 4 \lg n$, we may apply f to individual characters and concatenate the resulting values, viewed as binary strings. We use the same function parameter for all characters; thus, the length-reduced value for key x can be written as $f(x[0]_\sigma, s, a) f(x[1]_\sigma, s, a) \ldots f(x[q-1]_\sigma, s, a)$, where x is zero-padded to make $x[q-1]$ a σ-bit value (if necessary). The process is repeated with different multipliers and possibly different alphabets at subsequent levels of reduction. We will later refer to this way of combining function f as the *parallel reduction*.

To make the mapping to the reduced universe injective on S it is not necessary to make f injective on all character values that appear in the keys. Also, by making a special relation between alphabet sizes at different reduction levels, it is possible to improve performance of both construction and evaluation of the final function. The composed function that maps keys to a range of size $O(\log n \log \frac{w}{\log n})$ bits can be expressed as a certain dot product. For explanations about this string approach of combining function f and details about operations that need to be performed see [17]. Here we note one fact, initially overlooked, that simplifies and improves evaluation procedure. On a word RAM, dot product of integer vectors packed into single words can be computed in constant time when one operand is a constant. The constant operand can be preprocessed during the construction procedure — the order of the fields within the operand is reversed offline, during the construction. The dot product can later be computed via standard multiplication. To prevent overflows from spoiling the result, the operands are split into two components and two multiplications are performed. The fields of the components alternate between zero fields and fields retained from the original operands.

A bit different method is to serially apply function f on reduced suffixes of the key. Namely, we want to find multipliers $a_0, a_1, \ldots, a_{q-2}$ such that the function

$$a_0 \cdot x[0]_\sigma + (\ldots + (a_{q-3} \cdot x[q-3]_\sigma + (a_{q-2} \cdot x[q-2]_\sigma + x[q-1]_\sigma)) \ldots) \quad (2)$$

is injective on S. The final function has a similar form as before, and it is again evaluated as a dot product. However the composition of the function is represented differently, and the process of parameter selection is different. The multiplier selection algorithm is applied $q-1$ times, as suggested by the expression in (2). We will call this way of combining function f as the *suffix reduction*. Here we may set $\sigma = O(\log n)$, and thus have a smaller range of the final function. Beside the cost of calling the multiplier selection procedure $q-1$ times, there is a preparation task of partitioning the keys so that the elements in the sets $\{x_1[0], x_2[0], \ldots, x_n[0]\}$, $\{x_1[1], \ldots, x_n[1]\}$, …, $\{x_1[q-1], \ldots, x_n[q-1]\}$ are

grouped together. The cost of this partitioning is proportional to $\lg q$ times the cost of reading the keys (they may be packed into words). The selection of multipliers is the dominant task. If the input set of keys is the output of the parallel reduction method, then $q = O(\log(w/\log n))$.

An injective function composed of the functions generated by the method of parallel reduction and the method of suffix reduction can be evaluated in constant time on a word RAM. One upper bound on time required to compute the description of such a function is $O(n + n\frac{\log^3 n}{w} \log^3 \frac{w}{\log n} + \log w \log n)$, assuming the the input is already sorted. It is apparent that for $\log^{3+\epsilon} n < w < 2^{\frac{n}{\log n}}$ the construction algorithm runs in linear time on sorted input. We may use fusion trees to cover the extreme case $w \geq 2^{\frac{n}{\log n}}$ efficiently. In this paper we show an improved complexity of dictionary construction in the case $w = \log^{O(1)} n$.

3.2 Speed-Up of the Suffix Reduction

The outlined method of making deterministic signatures produces perfect hash functions with ranges of polynomial size. The functions have rather succinct descriptions, and they might have an application outside of dictionary structures. Here we will give up the injectiveness requirement, and replace it with considerably weaker properties. These weaker functions will be useful only when combined with additional data structures, foremost a dictionary structure for universes of polynomial size. The variant of the suffix reduction method that we introduce is particularly efficient in the external memory models. Yet it also produces some useful results in the word RAM model.

The multiplier selection procedure is again called $q - 1$ times, but each time with an input set of size $O(n/(\log n)^2)$. There will be no limit on the number of collisions that the final function may cause. Yet the function will have some properties that will allow the initial searching problem to be reduced either to a problem over a universe of size $O(\sigma)$ bits, or to a problem over a set of size $O(\log^2 n)$. The high level idea is to look for clusters of elements that already piled up and will hash to equal values by the final function, and to prevent further collisions between already formed clusters. Sorting operations (over shorter keys) will dominate the running times. Suppose that values for the parameters $a_l, a_{l+1}, \ldots, a_{q-2}$ were selected. If two keys x and y share the prefix of length l and the function values on their suffixes of length $q - l$ collide, i.e. $a_l \cdot x[l]_\sigma + a_{l+1} \cdot x[l+1]_\sigma + \ldots + x[q-1]_\sigma = a_l \cdot y[l]_\sigma + a_{l+1} \cdot y[l+1]_\sigma + \ldots + y[q-1]_\sigma$, then x and y will certainly be mapped to the same value by the final function. On the other hand, if the length l prefixes of x and y differ, it does not matter what are the values of the partial function on their suffixes of length $q - l$, since the separation of their hash values will be decided at a later time. The construction algorithm will keep track of sufficiently large clusters of elements that are certain to collide given the already selected multipliers. Different clusters will be ensured to map to different values. However keys not yet belonging to any cluster are able to join existing clusters or form new ones. The time of joining a cluster for a given key, specified by a prefix length, is possible to determine quickly during lookups. Some pieces of information related to this

joining point will enable us to substantially reduce the search space. To be precise, the reduced search space will consist of keys of length $O(\sigma)$ bits. If we set $\sigma = \Theta(\log n)$ then the method can be composed with the structure from Section 2.

The computationally dominant process in the construction algorithm will usually be sorting. The procedure performs $O(q)$ sorting operations over sets of $O(n)$ keys of length $O(\sigma)$ bits. In the external memory models this amounts to $O(\mathrm{Sort}(n))$ I/Os (with the block size B expressed in terms of $\lg |U|$-bit items, where U is the universe of keys that are input to the method). Combining this with the result from Section 2 produces the result stated in Theorem 2. In the word RAM model, the total sorting time is in general $O(nq \log\log n)$, based on [13]. When $\sigma = \Theta(\log n)$ we may use radix sort and get a time bound of $O(nq)$, which explains Remark 3.

3.3 Speed-Up of the Parallel Reduction

This section provides a sketch of the proof of Theorem 3 for the remaining case $\omega(\log n \log\log n) < w < \log^{3+\epsilon} n$. The approach is conceptually very similar to the approach that led to the speed-up of the method of suffix reduction, but the details are different and the computation is more involved. Consider partially reduced keys, after some number of levels of the parallel reduction. We call a prefix value heavy if it is shared by at least $(\lg n)^2$ partially reduced keys. We ensure that the current level of reduction avoids any collisions between heavy prefixes. For this purpose, it is convenient to maintain the trie of the partially reduced set (see [17]). It is again possible to substantially reduce the search space for a given key by using information related the key's point of joining a cluster of piled up elements.

In order to determine the level at which a key joined a cluster in constant time, we need to evaluate partially reduced values of the key for all levels in constant time. This is possible if $\Omega(K^2)$ copies of a key can fit in a single machine word, where K is the number of reduction levels; for $w = \log^{O(1)} n$ we have that $K = O(\log\log n)$. To provide such a situation we use two levels of the searching problem reduction from Section 3.2, using the setting $q = O(\log\log n)$. Hence the incurred construction cost from these two reduction steps is $O(n(\log\log n)^2)$.

The construction of this version of the parallel reduction function has a time cost proportional to K times the sorting time. For $w = \log^{O(1)} n$ we again get a bound of $O(n(\log\log n)^2)$. Since through the method of parallel reduction we map the keys to a range of size $O(\log n \log \frac{w}{\log n})$ bits, at the bottom end we again employ the suffix reduction but this time paired with the dictionary for polynomial-size universes.

Acknowledgment. The author wishes to thank Mihai Pătraşcu for interesting and helpful discussions on this subject, and in particular for pointing out that dot product with a constant vector can be easily computed in $O(1)$ time.

References

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM 31(9), 1116–1127 (1988)
2. Albers, S., Hagerup, T.: Improved parallel integer sorting without concurrent writing. In: Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 463–472 (1992)
3. Alon, N., Naor, M.: Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. Algorithmica 16(4-5), 434–449 (1996)
4. Andersson, A.: Sublogarithmic searching without multiplications. In: Proc. 36th Symposium on Foundations of Computer Science (FOCS), pp. 655–663 (1995)
5. Brodal, G.S., Fagerberg, R.: On the limits of cache-obliviousness. In: Proc. 35th Annual ACM Symposium on Theory of Computing, pp. 307–315 (2003)
6. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. J. of Comput. Syst. Sci. 18(2), 143–154 (1979)
7. Dietzfelbinger, M., Karlin, A.R., Mehlhorn, K., auf der Heide, F.M., Rohnert, H., Tarjan, R.E.: Dynamic perfect hashing: Upper and lower bounds. SIAM J. Comput. 23(4), 738–761 (1994)
8. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. ACM 31(3), 538–544 (1984)
9. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. J. Comput. Syst. Sci. 47(3), 424–436 (1993)
10. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS), pp. 285–298 (1999)
11. Hagerup, T.: Sorting and searching on the word RAM. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 366–398. Springer, Heidelberg (1998)
12. Hagerup, T., Miltersen, P.B., Pagh, R.: Deterministic dictionaries. J. Algor. 41(1), 69–85 (2001)
13. Han, Y.: Deterministic sorting in O(n log log n) time and linear space. J. Algor. 50(1), 96–105 (2004)
14. Overmars, M.H., van Leeuwen, J.: Worst-case optimal insertion and deletion methods for decomposable searching problems. Inf. Proc. Lett. 12(4), 168–173 (1981)
15. Pagh, R.: A trade-off for worst-case efficient dictionaries. Nordic J. Comput. 7(3), 151–163 (2000)
16. Raman, R.: Priority queues: Small, monotone and trans-dichotomous. In: Díaz, J. (ed.) ESA 1996. LNCS, vol. 1136, pp. 121–137. Springer, Heidelberg (1996)
17. Ružić, M.: Making deterministic signatures quickly. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, pp. 900–909. ACM, SIAM (2007); Preliminary full version is available on the author's web page
18. Ružić, M.: Uniform deterministic dictionaries. ACM Transactions on Algorithms 4(1) (2008)
19. Tarjan, R.E., Yao, A.C.-C.: Storing a sparse table. Commun. ACM 22(11), 606–611 (1979)

On List Update with Locality of Reference*

Susanne Albers and Sonja Lauer

University of Freiburg, Georges Köhler Allee 79, 79110 Freiburg, Germany
{salbers,lauers}@informatik.uni-freiburg.de

Abstract. We present a comprehensive study of the list update problem with locality of reference. More specifically, we present a combined theoretical and experimental study in which the theoretically proven and experimentally observed performance guarantees of algorithms match or nearly match. In the first part of the paper we introduce a new model of locality of reference that is based on the natural concept of runs. Using this model we develop refined theoretical analyses of popular list update algorithms. The second part of the paper is devoted to an extensive experimental study in which we have tested the algorithms on traces from benchmark libraries. It shows that the theoretical and experimental bounds differ by just a few percent. Our new bounds are substantially lower than those provided by standard competitive analysis. Another result is that the elegant Move-To-Front strategy exhibits the best performance, which confirms that it is the method of choice in practice.

1 Introduction

The list update problem is one of the most extensively studied online problems, with a tremendous body of work over the past 40 years. The problem consists in maintaining a set of items as an unsorted linear list. More specifically, a linear linked list of items is given. A list update algorithm is presented with a sequence of *requests* that must be served in their order of occurrence. Each request specifies an item in the list. In order to serve a request, a list update algorithm must *access* the requested item, i.e. it has to start at the front of the list and search linearly through the items until the desired item is found. Accessing the i-th item in the list incurs a cost of i. Immediately after an access, the requested item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. All other exchanges of two consecutive items in the list cost 1 and are called *paid exchanges*. The goal is to serve the request sequence so that the total cost is as small as possible. We are interested in *online algorithms* that serve each request without knowledge of any future requests.

While early work on the list update problem evaluated online algorithms assuming that requests are generated according to probability distributions, research over the past 20 years has focused on *competitive analysis* [24]. Here an

* Work supported by the German Research Foundation, projects AL 464/4-2 and 5-1.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 96–107, 2008.

online algorithm is compared to an optimal offline algorithm. Given a request sequence σ, let $A(\sigma)$ denote the cost incurred by online algorithm A in serving σ, and let $OPT(\sigma)$ denote the optimum offline cost. Algorithm A is called c-competitive if there exists a constant α such that $A(\sigma) \leq c \cdot OPT(\sigma) + \alpha$ holds for all σ and all size lists.

In 1985 Sleator and Tarjan proved that the *Move-To-Front* algorithm is 2-competitive [24]. This elegant strategy simply moves an item to the front of the list whenever it is requested. Since then, algorithms with an improved competitiveness have been developed. While the competitive ratios are of course constant, there is a substantial gap between the theoretical bounds and the performance ratios of the algorithms observed in practice. More precisely, the ratios in practice are much smaller than their theoretical counterparts. The reason is that competitive analysis considers arbitrary request sequences, whereas sequences arising in practice have a special structure: They exhibit locality of reference, i.e. that at any point in time only a small set of items is referenced.

There has been considerable research interest in studying the paging problem with locality of reference, see e.g. [2,11,15,18,20] because, in paging, the gap between the theoretical and experimental performance values is even superconstant. However, hardly any work has been presented for the classical list update problem. In fact, references [8,16] point out that locality is an essential aspect in the list update problem and that a good model is required to properly evaluate the performance of algorithms.

Previous results: We focus on the results that have been developed in the framework of competitive analysis. As mentioned above Sleator and Tarjan [24] showed that *Move-To-Front* is 2-competitive. This is the best factor deterministic online algorithms can achieve [19]. Bachrach and El-Yaniv [7] devised deterministic *MRI* and *PRI* families of algorithms. These families attain competitive ratios of 2 and 3, respectively. We next turn to randomized algorithms. The first randomized strategy was presented by Irani [17]. Her *Split* algorithm is 1.9375-competitive. Reingold et al. [22] presented an elegant *BIT* algorithm that is 1.75-competitive. This factor is substantially below the deterministic bound of 2. The *BIT* algorithm can be generalized to a family of *Counter* strategies [22]. A *Timestamp* family of algorithms was developed in [1]. It achieves a competitiveness equal to the Golden Ratio $\Phi \approx 1.62$. The best randomized algorithm currently known is *COMB* which is 1.6-competitive [3]. Interestingly, *COMB* is a combination of *BIT* and a (deterministic) element of the *Timestamp* family. The factor of 1.6 is close to best lower bound of 1.50084 shown by Ambühl [5] on the performance of randomized list update algorithms.

Experimental studies for the list update problem have been presented by Rivest [23], Bentley and McGeoch [9] and Bachrach et al. [8]. They analyzed popular algorithms on request sequences generated by probability distributions and Markov sources, on sequences derived from text and Pascal files as well as on sequences extracted from the Calgary Corpus [13]. The results are not unanimous. A conclusion is that the ranking of algorithms depends on the degree of locality in the input.

The only previous paper addressing list update with locality of reference is a technical report by Angelopoulos et al. [6]. They adapt a locality model introduced in [2] for the paging problem and prove that *Move-To-Front* is superior to other algorithms.

Our contribution: We present a comprehensive study of the list update problem with locality of reference. The goal is to provide a refined analysis of the problem in which theoretical and empirical results match or nearly match. To this end our study integrates theoretical and experimental work.

First, in Section 2, we introduce a new model of locality of reference that is based on the natural concept of *runs*. A run is a sequence of requests to the same item. We define a number of parameters that characterize request sequences in terms of the occurrence of long runs. Using these parameters we will be able to accurately estimate the performance of list update algorithms. We also define a model of so-called λ-*locality* that characterizes classes of input sequences with respect to their degree of locality. Loosely speaking, the more long runs there are, the higher the locality. As we shall see, our new concepts properly capture locality of reference in the list update problem, both from a theoretical and practical point of view.

In Section 3 we present refined theoretical analyses of list update algorithms. We concentrate on the most popular strategies that have received the most attention recently, namely *Move-To-Front*, *BIT* and *COMB*. In order to be able to analyze *COMB*, we have also evaluated a member of the *Timestamp* family. Of course, we have also investigated an optimal offline strategy. For each algorithm we have analyzed the total service cost incurred on a request sequence, where cost is expressed in terms of our new locality parameters. Interestingly, for *Move-To-Front*, our cost analysis is exact. Furthermore, for each online algorithm, we have evaluated its performance relative to that of an optimal offline algorithm. Here *Move-To-Front* achieves an excellent performance ratio that even tends down to 1 as the degree of locality increases.

In Section 4, we present a comprehensive experimental study in which we have evaluated our list update algorithms on real-world traces from benchmark libraries. Obviously, the list update problem is a solution to the classical dictionary problem. In this context, in practice, requests are memory accesses. Secondly, list update has interesting applications in data compression, see e.g. [10,12]. Therefore, in our experiments we consider as input (a) memory access strings (47 traces) and (b) sequences arising in data compression routines (44 traces). In our tests we first analyze the traces with respect to their locality characteristics. It shows that the parameters introduced in Section 2 are indeed sensible.

Next, in the experiments, for each algorithm and each input sequence, we have computed the total service cost. Furthermore, for each online algorithm and each input, we have determined the *experimentally observed competitiveness*, which is the total service cost of the algorithm divided by the total cost incurred by an optimal offline strategy. In general, our theoretically proven and experimentally observed bounds are very close and differ by just a few percent. As for the total service cost, *Move-To-Front* exhibits an error of 0 because our theoretical

bound is exact. For the other three online algorithms *BIT*, *Timestamp* and *COMB*, the average relative error between the theoretical and experimentally observed values is 3–4% on the memory access traces and 7–9% on the data compression sequences. As far as performance ratios relative to the optimum are concerned, the average relative errors between our theoretical bounds and the experimentally observed competitive ratios are a bit higher. *Move-To-Front* exhibits average relative errors of 0.3% on the memory traces and of 0.7% on the data compression sequences. The other three strategies incur average errors of 3–4% on the memory traces and of 8–10% on the data compression sequences.

In our study, the theoretical and experimental performance ratios of the algorithms are much lower than the corresponding standard competitive ratios. In particular, *Move-To-Front* shows the best performance with ratios in the range of 1.2–1.3. This confirms that *Move-To-Front* is the method of choice in practice.

2 A New Model for Locality of Reference

Informally speaking, a request sequence exhibits locality of reference if, at any time, it references only a small set of items. If an item is requested, it is likely to be requested again soon. This description suggests the concept of *runs*, where a run is a subsequence of requests to the same item. In the best case, when there is a high degree of locality, an item is requested many times in a row before a different element is referenced. Unfortunately, real-world request sequences may contain only few of these pure long runs. However, long runs may occur if we focus on small item sets and, in particular, on item pairs: If, at any time, item x is more significant than y, then this relation is likely to hold also in the near future and we encounter several requests to x before the next reference to y arises. Thus long runs occur if we project request sequences to smaller item sets or item pairs. A request sequence exhibits a high degree of locality if a substantial portion of the requests belongs to long runs. In the following we introduce a formal model of locality of reference based on this generalized notion of runs.

Let L be the set of items in the list to be maintained. Consider a fixed request sequence σ. For any two items $x, y \in L$ with $x \neq y$, let σ_{xy} be the request sequence that is derived from σ when deleting all requests that are neither to x nor to y, i.e. only the requests to x and y survive. Any maximal subsequence of consecutive requests to the same item in σ_{xy} is called a *run*. Let $r(\sigma_{xy})$ be the number of runs in σ_{xy}. A run is *short* if it consists of one request only. A run consisting of at least two requests is *long*. Let $s(\sigma_{xy})$ and $l(\sigma_{xy})$ denote the number of short and long runs, respectively, in σ_{xy}. Then $s(\sigma_{xy}) + l(\sigma_{xy}) = r(\sigma_{xy})$.

Online algorithms typically perform well on long runs, relative to an optimal offline algorithm. In order to properly evaluate our algorithms, we need some further definitions. A long run ρ is called a *prefixed long run* if it is preceded by one or more short runs; otherwise ρ is called an *independent long run*. Again let $l_p(\sigma_{xy})$ and $l_i(\sigma_{xy})$ be the number of prefixed and independent long runs, respectively. We have $l_p(\sigma_{xy}) + l_i(\sigma_{xy}) = l(\sigma_{xy})$. Consider two long runs ρ' and ρ

such that ρ' occurs earlier than ρ in σ_{xy}. Run ρ' *precedes* ρ if the last request of ρ' is followed by the first request of ρ in σ_{xy} or if ρ' and ρ are separated by short runs only. A long run ρ which is not equal to the first long run in σ_{xy} represents a *long run change* if ρ and the preceding long run ρ' reference different items. The first long run ρ in σ_{xy} represents a long run change if ρ and the first request of σ_{xy} reference the same item (imagining that σ_{xy} was preceded by a long run to just the other item of $\{x, y\}$). Let $l_c(\sigma_{xy})$ be the number of long run changes in σ_{xy}. We have $l_c(\sigma_{xy}) \leq l(\sigma_{xy})$. Furthermore, $l_i(\sigma_{xy}) \leq l_c(\sigma_{xy})$ because each independent long run, except for possibly the first one, is immediately preceded by another long run, which references a different item. The number of long run changes will be particularly important in lower bounding the cost incurred by an optimal offline algorithm. Hence, it will allow us to derive good upper bounds on the relative performance ratios of online strategies.

We next introduce some definitions regarding the beginning and end of σ_{xy}. Let $f_b(\sigma_{xy})$ be equal to 1 if the item first requested in σ_{xy} precedes the other item of $\{x, y\}$ in the initial list; otherwise let $f_b(\sigma_{xy}) = 0$. Moreover, let $f'_b(\sigma_{xy})$ be equal to 1 if $f_b(\sigma_{xy}) = 1$ and σ_{xy} starts with a short run followed by a long run; otherwise $f'_b(\sigma_{xy}) = 0$. Finally, let $f_e(\sigma_{xy})$ be equal to 1 if (a) σ_{xy} consists of a single request and the referenced item is stored after the other item of $\{x, y\}$ in the initial list or if (b) σ_{xy} ends with a short run but is preceded by a long run. Otherwise $f_e(\sigma_{xy}) = 0$.

So far we have defined a number of values for a particular request sequence σ_{xy}. We now sum these values over all pairs of items x and y. For any pair $x, y \in L$ with $x \neq y$ and for any value $v \in \{r, s, l, l_i, l_p, l_c, f_b, f'_b, f_e\}$, let $v(\sigma) = \sum_{\{x,y\} \subseteq L, x \neq y} v(\sigma_{xy})$. For instance, $r(\sigma)$ is the total number of runs in σ, while $s(\sigma)$ and $l(\sigma)$ represent the total number of short and long runs, respectively.

All the definitions presented so far refer to a given request sequence σ and, using these definitions, we will be able to accurately evaluate the performance of list update algorithms on such a σ. Next, we introduce a model of locality of reference that applies to *classes* of request sequences which may be generated by a particular application. Intuitively, request sequences exhibit a high degree of locality if there are many long runs. However, in order to obtain meaningful results we have to work with a refined definition. Again, the number of long run changes is crucial. We say that a class Σ of request sequences exhibits λ-*locality*, for some $0 \leq \lambda \leq 1$, if for any $\sigma \in \Sigma$ inequality $l_c(\sigma)/r(\sigma) \geq \lambda$ holds, i.e. the number of long run changes represents at least a fraction of λ among all the runs. Note that, for a given request sequence, $l_c(\sigma)$ accounts for all the independent long runs and, depending on the input, for a smaller or larger fraction of the prefixed long runs. If a sequence consists of long runs only, we have $\lambda = 1$.

A reader may wonder why our locality model does not incorporate the length of long runs. This parameter is irrelevant for algorithms performance because after the second request of a long run competitive algorithms have moved the referenced item ahead of the other item in the list and no further cost is incurred on the run.

3 Analyzing Online and Offline Algorithms

We analyze the costs incurred by an optimal offline algorithm OPT and by online algorithms on a request sequence σ. Due to space limitations, the proofs of all lemmas, theorems and corollaries are omitted. They are presented in detail in the full version of the paper.

Lemma 1. *The cost incurred by OPT is at least $OPT(\sigma) \geq \frac{1}{2}(r(\sigma) + l_c(\sigma) + f_e(\sigma)) - f_b(\sigma) + |\sigma|$.*

We next turn to online algorithms and start with deterministic strategies.

Algorithm Move-To-Front (MTF): Move the requested item to the front of the list.

Lemma 2. *The cost incurred by MTF is $MTF(\sigma) = r(\sigma) - f_b(\sigma) + |\sigma|$.*

For any request sequence σ, let $\alpha(\sigma) = (|\sigma| - f_b(\sigma))/r(\sigma)$. Furthermore, let $\beta(\sigma) = l_c(\sigma)/r(\sigma)$ be the fraction of the long run changes relative to the total number of runs. The following theorem gives a refined bound on the performance ratio of MTF. It implies, in particular, that MTF is 2-competitive.

Theorem 1. *For any request sequence σ, the cost incurred by MTF is at most $\frac{2+2\alpha(\sigma)}{1+2\alpha(\sigma)+\beta(\sigma)}$ times that payed by OPT.*

An immediate consequence of the above theorem is the following corollary. It implies that MTF can achieve a competitiveness as low as 1 on request sequences that exhibit a high degree of locality, i.e. that satisfy λ-locality with values of λ close to 1.

Corollary 1. *On request sequences exhibiting λ-locality, MTF achieves a competitive ratio of at most $\frac{2}{1+\lambda}$.*

Algorithm Timestamp (TS): Insert the requested item, say x, immediately in front of the first item in the list that precedes x in the current list and was requested at most once since the last request to x. If there is no such item or if x is requested for the first time, do not change the position of x.

Lemma 3. *The cost incurred by TS is $TS(\sigma) \leq r(\sigma) + l_i(\sigma) - l_p(\sigma) + f'_b(\sigma) + |\sigma|$.*

We observe that TS is better than MTF if the number of prefixed long runs is larger than the number of independent long runs plus $f'_b(\sigma)$. Let $\alpha'(\sigma) = (f_b(\sigma) + f'_b(\sigma))/r(\sigma)$ and $\gamma(\sigma) = (l_i(\sigma) - l_p(\sigma))/r(\sigma)$.

Theorem 2. *For any request sequence σ, the cost incurred by TS is at most $\frac{2+2\alpha(\sigma)+2\alpha'(\sigma)+2\gamma(\sigma)}{1+2\alpha(\sigma)+\beta(\sigma)}$ times that payed by OPT.*

We have $l_i(\sigma) \leq l_c(\sigma)$ and $f'_b(\sigma) \leq f_b(\sigma) \leq |L|(|L| - 1)/2$. Hence the above theorem also yields that TS is 2-competitive. As $l_p(\sigma)$ can be 0 and $l_i(\sigma)$ can be as high as $l_c(\sigma)$, we observe that TS does not achieve improved competitive ratios on request sequences satisfying λ-locality.

Next we study randomized algorithms.

Algorithm BIT: Maintain a bit $b(x)$ for each item $x \in L$. These bits are initialized independently and uniformly at random to a value in $\{0, 1\}$. On a request, complement the bit of the referenced item. If the bit value changes to 1, move the item to the front of the list.

Lemma 4. *The expected cost incurred by BIT is* $BIT(\sigma) \leq \frac{3}{4}r(\sigma) + \frac{1}{4}l(\sigma) + \frac{1}{2}l_i(\sigma) + \frac{1}{4}f_e(\sigma) + |\sigma|$.

Define $\delta(\sigma) = (\frac{1}{2}l(\sigma) + l_i(\sigma) + 2f_b(\sigma))/r(\sigma)$.

Theorem 3. *For any request sequence* σ*, the expected cost incurred by BIT is at most* $\frac{1.5+2\alpha(\sigma)+\delta(\sigma)}{1+2\alpha(\sigma)+\beta(\sigma)}$ *times that payed by OPT.*

It is not hard to show that the above theorem also implies that BIT is 1.75-competitive. Taking into account that $l(\sigma) \leq r(\sigma)$ and $l_i(\sigma) \leq l_c(\sigma)$, we obtain the following corollary, which yields that BIT attains a competitiveness of 1.5 on request sequences with a high degree of locality, i.e. with values of λ close to 1.

Corollary 2. *On request sequences exhibiting* λ*-locality, BIT achieves a competitive ratio of* $\min\{1.75, \frac{2+\lambda}{1+\lambda}\}$.

Algorithm Combination (COMB): With probability 4/5 serve the request sequence using BIT, and with probability 1/5 serve the sequence using TS.

Lemma 5. *The expected cost incurred by COMB is* $COMB(\sigma) \leq \frac{1}{5}(4r(\sigma) + 4l_i(\sigma) + f_e(\sigma) + f'_b(\sigma)) + |\sigma|$.

Let $\zeta(\sigma) = (l_i(\sigma) + \frac{5}{4}f_b(\sigma) + \frac{1}{4}f'_b(\sigma))/r(\sigma)$.

Theorem 4. *For any request sequence* σ*, the expected cost incurred by COMB is at most* $\frac{1.6+2\alpha(\sigma)+1.6\zeta(\sigma)}{1+2\alpha(\sigma)+\beta(\sigma)}$ *times that payed by OPT.*

We have $l_i(\sigma) \leq l_c(\sigma)$ and $f'_b(\sigma) \leq f_b(\sigma) \leq |L|(|L| - 1)/2$. Hence the above theorem also implies that $COMB$ is 1.6-competitive. Since $l_i(\sigma)$ can be as high as $l_c(\sigma)$, algorithm $COMB$ does not achieve improved competitive ratios on request sequences satisfying λ-locality.

4 Experimental Study

In our experimental study we have implemented all the algorithms analyzed in Section 3. The main purpose of our study is to compare the experimentally observed performance of the algorithms to the bounds stated in Theorems 1–4 as well as Corollaries 1 and 2. In order to get meaningful results we have tested the algorithms on real-world request sequences from benchmark libraries. Clearly, self-organizing linear lists represent a solution to the classical dictionary problem. A second important application of self-organizing linear lists is data

compression. Since the latter application has gained considerable popularity and importance recently, we report on the corresponding results in this extended abstract and address the findings with respect to the dictionary application in the full version of the paper.

Data compression schemes: Bentley et al. [10] showed that self-organizing linear lists can be used to build locally adaptive data compression schemes. The best compression results are achieved when the scheme is combined with the famous Burrows-Wheeler transformation [12]. In fact the common open source data compression program `bzip2` consists of a Burrows-Wheeler transformation followed by Move-To-Front and Huffman encodings. We briefly describe the approach.

In data compression we are given a string S that shall be *compressed*, i.e. that shall be represented using fewer bits. The string S consists of *symbols*, where each symbol is element of an alphabet $X = \{x_1, \ldots, x_n\}$. The idea of data compression schemes using linear lists, proposed by Bentley et al., is to convert the string S of symbols into a string I of integers. An *encoder* maintains a linear list of symbols contained in X and reads the symbols in the string S. Whenever the symbol x_i has to be compressed, the encoder looks up the current position of x_i in the linear list, outputs this position and updates the list using a list update algorithm. Here one can use MTF or any other list update strategy. If symbols to be compressed are moved closer to the front of the list, then frequently occurring symbols can be encoded with small integers. Clearly, when the string I is actually stored or transmitted, each integer in the string should be coded again using a variable length prefix code.

The refined compression scheme by Burrows and Wheeler first applies a reversible transformation to the string S. The purpose of this transformation is to group together instances of a symbol x_i occurring in S so that the resulting string S' exhibits a high degree of locality of reference. Due to space limitation we refer the reader to [12] for details on the Burrows-Wheeler transformation and its efficient implementation. The transformed string S' is then encoded using the algorithm by Bentley et al. as described in the previous paragraph. Alphabet X, i.e. the set of entries in the self-organizing linear list, is the ASCII alphabet with its 256 different characters. The initial list is given by the initial numerical order of the ASCII characters. In the string S, each byte represents a symbol. For large files to be compressed, the Burrows-Wheeler transformation is applied not to the entire file but rather to blocks of uniform size; the block size may be chosen by a user.

Data sets and their locality characteristics: In our experiments we selected files available at the repository named *Canterbury Corpus* [14], which is the standard benchmark library for evaluating data compression algorithms. The corpus contains files of different types. In addition to text files such as books and papers there are source code files, pictures, office documents, object files, and many more. A description of the corpus can be found at [14].

In our tests we selected all the files from the Canterbury Corpus. We applied the Burrows-Wheeler transformation to each of these files. We chose a block size of $9 \cdot 10^5$ bytes, which is the default and also the maximum allowable block size in `bzip2`. If the file size exceeds the block size, then, as described above, the file is split into several blocks of the chosen size. The sequences obtained from the Burrows-Wheeler transformation are the request sequences on which the list update algorithms have to be evaluated. Recall that each byte of the sequence forms a request. We remark that we actually do not compress files; instead we evaluate list update algorithms on these realistic benchmark sequences.

The full version of the paper contains detailed tables presenting the characteristics of our (transformed) request sequences. Due to space limitations, in this extended abstract we report the main facts. The length of the request sequences differs vastly among the test instances. There are short sequences consisting of only 3721 requests and long sequences of up to $9 \cdot 10^5$ references, which occur when a file is split into several blocks. Moreover, the number of different bytes (ASCII characters) requested differs vastly. In text files typically about 80 to 90 different characters are requested. In object files all the 256 ASCII characters are referenced.

For each of the request sequences we have determined the values r, s, l, l_p, l_i, l_c as introduced in Section 2. We have also determined f_b, f'_b, f_e but, for brevity, will not discuss them. Instead of giving the absolute values of r, s, l, l_p, l_i, l_c we present the interesting relations. First, it shows that among all the runs of a request sequence, about 60% to 65% are long runs. The fraction can go as high as 80%, for some files. Next, we analyzed the distribution of the prefixed and independent long runs among all the long runs. Again, in most cases, the majority of the long runs are independent long runs (fractions can go as high as 80%), indicating that long runs are usually followed by long runs. Finally, we studied the number of long run changes relative to the number of long runs. Here the interesting observation is that the ratio l_c/l is typically 5% to 10% higher than l_i/l. This demonstrates that the definition of l_c was indeed sensible in Section 2 as it yields more expressive lower bounds on the cost of OPT, compared to l_i. A final remark is that, for files split into several blocks, all the numbers for the various blocks are consistent, i.e. the characteristics do not change within the file.

Performance results: We have executed the online algorithms analyzed in Section 3 on all the request sequences described above and recorded their cost. As for the randomized strategies BIT and $COMB$, they were executed 16 times on each sequences and, for any sequence, the average cost was taken. Since the offline version of the list update problem is NP-hard [4] and the best known offline algorithm takes $O(2^n n! m)$ time [21], where $n = |L|$ and $m = |\sigma|$, computing the true optimum offline cost is impossible for our request sequences. Therefore, we computed the *pairwise optimum*, which is equal to $\sum_{\{x,y\} \subseteq L, x \neq y} OPT(\sigma_{xy}) + |\sigma|$. Here $OPT(\sigma_{xy})$ denotes the cost incurred by an optimal offline algorithm OPT in serving σ_{xy} on a two-item list consisting of x and y only. This cost is easy to determine because an optimal strategy on a two-item list moves a referenced item to the front of the list whenever it is requested at least twice in a row. The

pairwise optimum is usually used as approximation of the optimum offline cost, even in standard competitive analysis.

In our performance analysis we have first evaluated the costs incurred by the algorithms on all the sequences. For each algorithm we compared the costs observed in the experiments to those implied by the theoretical bounds of Lemmas 1–5. For any $A \in \{MTF, TS, BIT, COMB, OPT\}$, we recorded the experimentally observed cost. Using the values $r, s, l, l_i, l_p, l_c, f_b, f'_b, f_e$ obtained for our request sequences, we derived theoretical bounds applying Lemmas 1–5. Instead of reporting these bounds, we focus on the relative errors. The relative error, for a given strategy A and request sequence σ, is the absolute value of the difference between the experimental and theoretical costs, divided by the experimental value.

The full paper contains a detailed description of the errors. Due to space limitations, in this extended abstract we summarize the results on the relative errors using box plots, which is a standard method to display numerical data. We refer the reader to the left chart of Figure 1. For each algorithm, the bold line within the box represents the median data point. The box includes 50% of the data points, where 25% is located above and 25% below the median. The upper (respectively lower) whisker is the maximum (respectively minimum) data point that can be found within a distance of 1.5 of the inter-quartile range. All other points are outliers. It shows that all the errors are very small, indicating that the bounds developed in Lemmas 1–5 very well approximate the experimentally observed cost. As for OPT, or the pairwise optimum, the average relative error is below 0.5%. The incurred error for MTF is 0 because the bound given in Lemma 2 is exact. For the other three online strategies, the average relative errors are between 7% and 8%; the medians are even below 2%.

For comparison with previous experimental studies [7,8] we have also analyzed the average service cost incurred by the algorithms on a single request. These average costs are all very small, typically in the range between 1.8 and 5, which confirms that the request sequences exhibit a high degree of locality and that the algorithms respond well to this property.

The main results of our experimental study is a careful comparison of the experimentally observed competitiveness to the theoretical bounds developed in Theorems 1–4. The comparision is done for each online algorithm and each of our request sequences. Again, the full paper contains a detailed presentation. Here we summarize the results by considering average values. The right chart of Figure 1 depicts, for each algorithm, the average performance values, over all files. The average experimentally observed competitiveness is shown as grey bars while the average theoretical bounds are shown in white. A first, very positive finding is that the experimentally observed and theoretical performance ratios are very close to each other. Hence our locality model and theoretical analyses are indeed sensible. The best results are achieved for MTF. Here the average relative error is below 0.7%. For almost all of the files, the actual error is substantially smaller because a few files contribute very high errors to the average value. For the other three online algorithms the average relative errors are higher, ranging

between 8% and 10%, but these values are still reasonable. Again, for many files we have very small errors; high contributions in the average relative errors just come from a few files.

A second important result is that the experimentally observed competitiveness as well as the performance ratios implied by Theorems 1–4 are much lower than the standard competitive ratios of the algorithms. Recall that MTF and TS are 2-competitive while BIT and $COMB$ achieve competitive ratios of 1.75 and 1.6, respectively. In our experiments, MTF shows the best behavior with performance ratios between 1.2 and 1.3. The other three algorithms TS, BIT and $COMB$ are slightly worse, with ratios that are typically in the range between 1.3 and 1.6. There is no clear winner among these three algorithms. For each strategy there are some sequences where this stategy outperforms the other two.

Finally, our experimental study analyzes the performance of MTF and BIT in terms of λ-locality. For both algorithms, the competitive performance under λ-locality is higher than the theoretical bounds of Theorems 1 and 3. This is not surprising because Corollaries 1 and 2 consider asymptotic algorithm performance, ignoring the request sequence length $|\sigma|$, i.e. an additive 1 per request, in both the online and offline cost. However, since the average service cost of the algorithms is small, the additive values of 1 make a difference in performance. Further details are given in the full version of the paper.

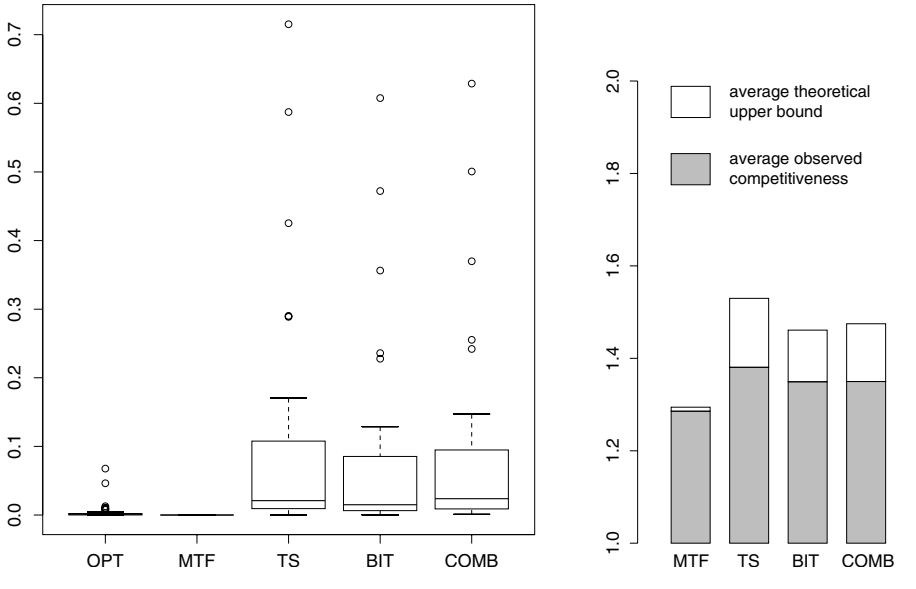


Fig. 1. Left chart: Relative errors of the upper bounds on the service costs given by Lemmas 1–5 when compared to the actual costs observed. Right chart: Average upper bounds on the performance ratios as implied by Theorems 1–4 (white bars) compared to the average experimentally observed competitiveness (grey bars).

References

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM Journal on Computing 27, 670–681 (1998)
2. Albers, S., Favrholdt, L.M., Giel, O.: On paging with locality of reference. Journal of Computer and System Sciences 70, 145–175 (2005)
3. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Information Processing Letters 56, 135–139 (1995)
4. Ambühl, C.: Offline list update is NP-hard. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 42–51. Springer, Heidelberg (2000)
5. Ambühl, C., Gärtner, B., von Stengel, B.: A new lower bound for the list update problem in the partial cost model. Theoretical Computer Science 268, 3–16 (2001)
6. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: List update with locality of reference: MTF outperforms all other algorithms. Technical Report CS-2006-46, School of Computer Science, University of Waterloo (2006)
7. Bachrach, R., El-Yaniv, R.: Online list accessing algorithms and their applications: Recent empirical evidence. In: Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 53–62 (1997)
8. Bachrach, R., El-Yaniv, R., Reinstädtler, M.: On the competitive theory and practice of online list accessing algorithms. Algorithmica 32, 201–245 (2002)
9. Bentley, J.L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Communication of the ACM 28, 404–411 (1985)
10. Bentley, J.L., Sleator, D.S., Tarjan, R.E., Wei, V.K.: A locally adaptive data compression scheme. Communications of the ACM 29, 320–330 (1986)
11. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. Journal of Computer and System Sciences 50, 244–258 (1995)
12. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. DEC SRC Research Report 124 (1994)
13. Calgary Corpus, `http://links.uwaterloo.ca/calgary.corpus.html`
14. The Canterbury Corpus, `http://corpus.canterbury.ac.nz/`
15. Fiat, A., Mendel, M.: Truly online paging with locality of reference. In: Proc. 38rd Annual Symposium on Foundations of Computer Science, pp. 326–335 (1997)
16. Hester, J.H., Hirschberg, D.S.: Self-organizing linear search. ACM Computing Surveys 17, 295–312 (1985)
17. Irani, S.: Two results on the list update problem. Information Processing Letters 38, 301–306 (1991)
18. Karlin, A., Phillips, S., Raghavan, P.: Markov paging. In: Proc. 33rd Annual Symposium on Foundations of Computer Science, pp. 24–27 (1992)
19. Karp, R., Raghavan, P.: Personal communication cited in [22] (1990)
20. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. In: Proc. 35th Annual Symposium on Foundations of Computer Science, pp. 394–400 (1994)
21. Reingold, N., Westbrook, J.: Optimum off-line algorithms for the list update problem. Technical Report YALEU/DCS/TR-805, Yale University (1990)
22. Reingold, N., Westbrook, J., Sleator, D.D.: Randomized competitive algorithms for the list update problem. Algorithmica 11, 15–32 (1994)
23. Rivest, R.: On self-organizing sequential search heuristics. Communications of the ACM 19, 63–67 (1976)
24. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28, 202–208 (1985)

A New Combinatorial Approach for Sparse Graph Problems

Guy E. Blelloch, Virginia Vassilevska, and Ryan Williams

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA
{guyb,virgi,ryanw}@cs.cmu.edu

Abstract. We give a new combinatorial data structure for representing arbitrary Boolean matrices. After a short preprocessing phase, the data structure can perform fast vector multiplications with a given matrix, where the runtime depends on the sparsity of the input vector. The data structure can also return *minimum witnesses* for the matrix-vector product. Our approach is simple and implementable: the data structure works by precomputing small problems and recombining them in a novel way. It can be easily plugged into existing algorithms, achieving an asymptotic speedup over previous results. As a consequence, we achieve new running time bounds for computing the transitive closure of a graph, all pairs shortest paths on unweighted undirected graphs, and finding a maximum node-weighted triangle. Furthermore, any asymptotic improvement on our algorithms would imply a $o(n^3/\log^2 n)$ combinatorial algorithm for Boolean matrix multiplication, a longstanding open problem in the area. We also use the data structure to give the first asymptotic improvement over $O(mn)$ for all pairs least common ancestors on directed acyclic graphs.

1 Introduction

A large collection of graph problems in the literature admit essentially two solutions: an *algebraic* approach and a *combinatorial* approach. Algebraic algorithms rely on the theoretical efficacy of fast matrix multiplication over a ring, and reduce the problem to a small number of matrix multiplications. These algorithms achieve unbelievably good theoretical guarantees, but can be impractical to implement given the large overhead of fast matrix multiplication. Combinatorial algorithms rely on the efficient preprocessing of small subproblems. Their theoretical guarantees are typically worse, but they usually lead to more practical improvements. Combinatorial approaches are also interesting in that they have the capability to tackle problems that seem to be currently out of the reach of fast matrix multiplication. For example, many sparse graph problems are not known be solvable quickly with fast matrix multiplication, but a combinatorial approach can give asymptotic improvements. (Examples are below.)

In this paper, we present a new combinatorial method for preprocessing an $n \times n$ dense Boolean matrix A in $O(n^{2+\varepsilon})$ time (for any $\varepsilon > 0$) so that sparse vector multiplications with A can be done faster, while matrix updates are not too expensive to handle. In particular,

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 108–120, 2008.

- for a vector v with t nonzeros, $A \cdot v$ can be computed in $O(\frac{n}{\log n}(t/\kappa + n/\ell))$ time, where ℓ and κ are parameters satisfying $\binom{\ell}{\kappa} \leq n^{\varepsilon}$, and
- row and/or column updates to the matrix can be performed in $O(n^{1+\varepsilon})$ time.

The matrix-vector multiplication can actually return a vector w of *minimum witnesses*; that is, $w[i] = k$ iff k is the minimum index satisfying $A[i,k] \cdot v[k] \neq 0$. The data structure is simple, does not use devious "word tricks" or hide any large constants, and can be implemented on a pointer machine.[1] We apply our data structure to four fundamental graph problems: transitive closure, all pairs shortest paths, minimum weight triangle, and all pairs least common ancestors. All four are known to be solvable in $n^{3-\delta}$ time for some $\delta > 0$, but the algorithms are algebraic and do not exploit the potential sparsity of graphs. With the right settings of the parameters ℓ and κ, our data structure can be applied to all the above four problems, giving the best runtime bounds for *sparse problems* to date.

Transitive Closure: We have a directed graph on n nodes and m edges, and wish to find all pairs of nodes (u, v) whether there is a path from u to v in the graph. Transitive closure has myriad applications and a long history of ideas. The best known theoretical algorithms use $O(M(n))$ time [10,13] where $M(n)$ is the complexity of $n \times n$ Boolean matrix product, and $O(mn/\log n + n^2)$ time [5,3]. Algebraic matrix multiplication implies an $O(n^{\omega})$ algorithm, where $\omega < 2.376$ [6], and combinatorial matrix multiplication gives an $O(n^3/\log^2 n)$ runtime [2,14,18]. Our data structure can be used to implement transitive closure in $O(mn(\log(\frac{n^2}{m})/\log^2 n) + n^2)$ time. This constitutes the first combinatorial improvement on the bounds of $O(n^3/\log^2 n)$ and $O(mn/\log n + n^2)$ that follow from Four Russians preprocessing, and it establishes the best known running time for general sparse graphs.

All Pairs Shortest Paths (APSP): We want to construct a representation of a given graph, so that for any pair of nodes, a shortest path between the pair can be efficiently obtained from the representation. The work on APSP is deep and vast; here we focus on the simplest case where the graph is *undirected* and *unweighted*. For this case, Galil and Margalit [11] and Seidel [15] gave $O(n^{\omega})$ time algorithms. These algorithms do not improve on the simple $O(mn + n^2)$ algorithm (using BFS) when $m = o(n^{\omega-1})$. The first improvement over $O(mn)$ was given by Feder and Motwani [9] who gave an $O(mn \log(\frac{n^2}{m})/\log n)$ time algorithm. Recently, Chan presented new algorithms that take $\hat{O}(mn/\log n)$ time.[2] We show that APSP on undirected unweighted graphs can be computed in $O(mn \log(\frac{n^2}{m})/\log^2 n)$ time. Our algorithm modifies Chan's $O(mn/\log n + n^2 \log n)$ time solution, implementing its most time-consuming procedure efficiently using our data structure.

[1] When implemented on the w-word RAM, the multiplication operation runs in $O(\frac{n}{w}(t/k + n/\ell))$. In fact, all of the combinatorial algorithms mentioned in this paper can be implemented on a w-word RAM in $O(T(n)(\log n)/w)$ time, where $T(n)$ is the runtime stated.

[2] The $\hat{O}$ notation suppresses $\mathrm{poly}(\log \log n)$ factors.

All Pairs Weighted Triangles: Here we have a directed graph with an arbitrary weight function w on the nodes. We wish to compute, for all pairs of nodes v_1 and v_2, a node v_3 that such that (v_1, v_3, v_2, v_1) is a cycle and $\sum_i w(v_i)$ is minimized or maximized. The problem has applications in data mining and pattern matching. Recent research has uncovered interesting algebraic algorithms for this problem [17], the current best being $O(n^{2.575})$, but again it is somewhat impractical, relying on fast rectangular matrix multiplication. (We note that the problem of finding a *single* minimum or maximum weight triangle has been shown to be solvable in $n^{\omega+o(1)}$ time [8].) The proof of the result in [17] also implies an $O(mn/\log n)$ algorithm. Our data structure lets us solve the problem in $O(mn\log(\frac{n^2}{m})/\log^2 n)$ time.

Least Common Ancestors on DAGs: Given a directed acyclic graph G on n nodes and m edges, fix a topological order on the nodes. For all pairs of nodes s and t we want to compute the highest node in topological order that still has a path to both s and t. Such a node is called a least common ancestor (LCA) of s and t. The all pairs LCA problem is to determine an LCA for every pair of vertices in a DAG. In terms of n, the best algebraic algorithm for finding all pairs LCAs uses the minimum witness product and runs in $O(n^{2.575})$ [12,7]. Czumaj, Kowaluk, and Lingas [12,7] gave an algorithm for finding all pairs LCAs in a *sparse* DAG in $O(mn)$ time. We improve this runtime to $O(mn\log(\frac{n^2}{m})/\log n)$.

1.1 On the Optimality of Our Algorithms

We have claimed that all the above problems (with the exception of the last one) can be solved in $O(mn\log(n^2/m)/\log^2 n)$ time. How does this new runtime expression compare to previous work? It is easier to see the impact when we let $m = n^2/s$ for a parameter s. Then

$$\frac{mn\log(n^2/m)}{\log^2 n} = \frac{n^3}{\log^2 n}\cdot\frac{\log s}{s} \tag{1}$$

Therefore, our algorithms yield asymptotic improvements on "medium density" graphs, where the number of edges is in the range $n^{2-o(1)} \leq m \leq o(n^2)$.

At first glance, such an improvement may appear small. We stress that our algorithms have essentially reached the best that one can do for these problems, without resorting to fast matrix multiplication or achieving a major breakthrough in combinatorial matrix algorithms. As all of the above problems can be used to simulate Boolean matrix multiplication, (1) implies that an $O\left(mn\frac{\log(n^2/m)}{f(n)\log^2 n}\right)$ algorithm for any of the above problems and any unbounded function $f(n)$ would entail an asymptotic improvement on combinatorial Boolean matrix multiplication, a longstanding open problem. Note that an $O\left(\frac{mn}{f(n)\log n}\right)$ combinatorial algorithm does not imply such a breakthrough: let $m = n^2/s$ and $f(n) = o(\log n)$ and observe $mn/(f(n)\log n) = n^3/(sf(n)\log n)$; such an algorithm is still slow on sufficiently dense matrices.

1.2 Related Work

Closely related to our work is Feder and Motwani's ingenious method for compressing sparse graphs, introduced in STOC'91. In the language of Boolean matrices, their method runs in $\tilde{O}(mn^{1-\varepsilon})$ time and decomposes an $n \times n$ matrix A with m nonzeros into an expression of the form $A = (A_1 * A_2) \vee A_3$, where $*$ and $\vee$ are matrix product and pointwise-OR, A_1 is $n \times mn^{\varepsilon}$, A_2 is $mn^{\varepsilon} \times n$, A_1 and A_2 have $O(\frac{m \log(n^2/m)}{\log n})$ nonzeros, and A_3 has $O(n^{1+\varepsilon})$ nonzeros. Such a decomposition has many applications to speeding up algorithms.

While a $(\log n)/\log(n^2/m)$ factor of savings crops up in both approaches, our approach is markedly different from graph compression; in fact it seems orthogonal. From the above viewpoint, Feder-Motwani's graph compression technique can be seen as a method for preprocessing a sparse Boolean matrix so that multiplications of it with arbitrary vectors can be done in $O\left(\frac{m \log(n^2/m)}{\log n}\right)$ time. In contrast, our method preprocesses an *arbitrary matrix* so that its products with *sparse vectors* are faster, and updates to the matrix are not prohibitive. This sort of preprocessing leads to a Feder-Motwani style improvement for a new set of problems. It is especially applicable to problems in which an initially sparse graph is augmented and becomes dense over time, such as transitive closure.

Our data structure is related to one given previously by the third author [18], who showed how to preprocess a matrix over a constant-sized semiring in $O(n^{2+\varepsilon})$ time so that subsequent vector multiplications can be performed in $O(n^2/\log^2 n)$ time. Ours is a significant improvement over this data structure in two ways: the runtime of multiplication now varies with the sparsity of the vector (and is never worse than $O(n^2/\log^2 n)$), and our data structure also returns minimum witnesses for the multiplication. Both of these are non-trivial augmentations that lead to new applications.

2 Preliminaries and Notation

Define $H(x) = x \log_2(1/x) + (1-x)\log_2(1/(1-x))$. H is often called the binary entropy function. All logarithms are assumed to be base two. Throughout this paper, when we consider a graph $G = (V, E)$ we let $m = |E|$ and $n = |V|$. G can be directed or undirected; when unspecified we assume it is directed. We define $\delta_G(s, v)$ to be the distance in G from s to v. We assume G is always weakly connected, so that $m \geq n-1$. We use the terms *vertex* and *node* interchangeably. For an integer ℓ, let $[\ell]$ refer to $\{1, \ldots, \ell\}$.

We denote the typical Boolean product of two matrices A and B by $A \cdot B$. For two Boolean vectors u and v, let $u \wedge v$ and $u \vee v$ denote the componentwise AND and OR of u and v respectively; let $\neg v$ be the componentwise NOT on v.

A *minimum witness* vector for the product of a Boolean matrix A with a Boolean vector v is a vector w such that $w[i] = 0$ if $\vee_j(A[i][j] \cdot v[j]) = 0$, and if $\vee_j(A[i][j] \cdot v[j]) = 1$, $w[i]$ is the minimum index j such that $A[i][j] \cdot v[j] = 1$.

3 Combinatorial Matrix Products with Sparse Vectors

We begin with a data structure which after preprocessing stores a matrix while allowing efficient matrix-vector product queries and column updates to the matrix. On a matrix-vector product query, the data structure not only returns the resulting product matrix, but also a minimum witness vector for the product.

Theorem 1. *Let B be a $d \times n$ Boolean matrix. Let $\kappa \geq 1$ and $\ell > \kappa$ be integer parameters. Then one can create a data structure with $O(\frac{dn\kappa}{\ell} \cdot \sum_{b=1}^{\kappa} \binom{\ell}{b})$ preprocessing time so that the following operations are supported:*

- *given any $n \times 1$ binary vector r, output $B \cdot r$ and a $d \times 1$ vector w of* minimum witnesses *for $B \cdot r$ in $O(d \log n + \frac{d}{\log n}\left(\frac{n}{\ell} + \frac{m_r}{\kappa}\right))$ time, where m_r is the number of nonzeros in r;*
- *replace any column of B by a new column in $O(d\kappa \sum_{b=1}^{\kappa} \binom{\ell}{b})$ time.*

The result can be made to work on a pointer machine as in [18]. Observe that the naïve algorithm for $B \cdot r$ that simulates $\Theta(\log n)$ word operations on a pointer machine would take $O(\frac{dm_r}{\log n})$ time, so the above runtime gives a factor of κ savings provided that ℓ is sufficiently large. The result can also be generalized to handle products over any fixed size semiring, similar to [18].

Proof of Theorem 1. Let $0 < \varepsilon < 1$ be a sufficiently small constant in the following. Set $d' = \lceil \frac{d}{\lceil \varepsilon \log n \rceil} \rceil$ and $n' = \lceil n/\ell \rceil$. To preprocess B, we divide it into block submatrices of at most $\lceil \varepsilon \log n \rceil$ rows and ℓ columns each, writing B_{ji} to denote the j, i submatrix, so

$$B = \begin{bmatrix} B_{11} & \dots & B_{1n'} \\ \vdots & \ddots & \vdots \\ B_{d'1} & \dots & B_{d'n'} \end{bmatrix}.$$

Note that entries from the kth column of B are contained in the submatrices $B_{j\lceil k/\ell \rceil}$ for $j = 1, \dots, d'$, and the entries from kth row are in $B_{\lceil k/\ell \rceil i}$ for $i = 1, \dots, n'$. For simplicity, from now on we omit the ceilings around $\varepsilon \log n$.

For every $j = 1, \dots, d'$, $i = 1, \dots, n'$ and every ℓ length vector v with at most κ nonzeros, precompute the product $B_{ji} \cdot v$, and a minimum witness vector w which is defined as: for all $k = 1, \dots \varepsilon \log n$,

$$w[k] = \begin{cases} 0 & \text{if } (B_{ji} \cdot v)[k] = 0 \\ (i-1)\ell + w' & \text{if } B_{ji}[k][w'] \cdot v[w'] = 1 \text{ and } \forall w'' < w', B_{ji}[k][w''] \cdot v[w''] = 0 \end{cases}$$

Store the results in a look-up table. Intuitively, w stores the minimum witnesses for $B_{ji} \cdot v$ with their indices in $[n]$, that is, as if B_{ji} is construed as an $n \times n$ matrix which is zero everywhere except in the (j, i) subblock which is equal to B_{ji}, and v is construed as a length n vector which is nonzero only in its ith block which equals v. This product and witness computation on B_{ji} and v takes $O(\kappa\varepsilon \log n)$ time. There are at most $\sum_{b=1}^{\kappa} \binom{\ell}{b}$ such vectors v, and hence this precomputation

takes $O(\kappa \log n \sum_{b=1}^{\kappa} \binom{\ell}{b})$ time for fixed B_{ji}. Over all j, i the preprocessing takes $O(\frac{dn}{\ell \log n} \cdot \kappa \log n \sum_{b=1}^{\kappa} \binom{\ell}{b}) = O(\frac{dn\kappa}{\ell} \sum_{b=1}^{\kappa} \binom{\ell}{b})$ time.

Suppose we want to modify column k of B in this representation. This requires recomputing $B_{j\lceil k/\ell \rceil} \cdot v$ and the witness vector for this product, for all $j = 1, \ldots, n'$ and for all length ℓ vectors v with at most κ nonzeros. Thus a column update takes only $O(d\kappa \sum_{b=1}^{\kappa} \binom{\ell}{b})$ time.

Now we describe how to compute $B \cdot r$ and its minimum witnesses. Let m_r be the number of nonzeros in r. We write $r = [r_1 \ \cdots \ r_{n'}]^T$ where each r_i is a vector of length ℓ. Let m_{ri} be the number of nonzeros in r_i.

For each $i = 1, \ldots, n'$, we decompose r_i into a union of at most $\lceil m_{ri}/\kappa \rceil$ disjoint vectors r_{ip} of length ℓ and at most κ nonzeros, so that r_{i1} contains the first κ nonzeros of r_i, r_{i2} the next κ, and so on, and $r_i = \vee_p r_{ip}$. Then, for each $p = 1, \ldots, \lceil m_{ri}/\kappa \rceil$, r_{ip} has nonzeros with larger indices than all $r_{ip'}$ with $p' < p$, *i.e.* if $r_{ip}[q] = 1$ for some q, then for all $p' < p$ and $q' \geq q$, $r_{ip'}[q'] = 0$.

For $j = 1, \ldots, d'$, let $B^j = [B_{j1} \ldots B_{jn'}]$. We shall compute $v_j = B^j \cdot r$ separately for each j and then combine the results as $v = [v_1, \ldots, v_{d'}]^T$. Fix $j \in [d']$. Initially, set v_j and w_j to be the all-zeros vector of length $\varepsilon \log n$. The vector w_j shall contain minimum witnesses for $B^j \cdot r$.

For each $i = 1, \ldots, n'$ in increasing order, consider r_i. In increasing order for each $p = 1, \ldots, \lceil m_{ri}/\kappa \rceil$, process r_{ip} as follows. Look up $v = B_{ji} \cdot r_{ip}$ and its witness vector w. Compute $y = v \wedge \neg v_j$ and then set $v_j = v \vee v_j$. This takes $O(1)$ time. Vector y has nonzeros in exactly those coordinates h for which the minimum witness of $(B^j \cdot r)[h]$ is a minimum witness of $(B_{ji} \cdot r_{ip})[h]$; since over all i' and p' the nonzeros of $r_{i'p'}$ partition the nonzeros of r, this minimum witness is not a minimum witness of any $(B_{ji'} \cdot r_{i'p'})[h]$ with $i' \neq i$ or $p' \neq p$. In this situation we say that the minimum witness is in r_{ip}. In particular, if $y \neq 0$, r_{ip} contains some minimum witnesses for $B^j \cdot r$ and the witness vector w_j for $B^j \cdot r$ needs to be updated. Then, for each $q = 1, \ldots, \varepsilon \log n$, if $y[q] = 1$, set $w_j[q] = w[q]$. This ensures that after all i, p iterations, $v = \bigvee_{i,p} (B_{ji} \cdot r_{ip}) = B^j \cdot r$ and w_j is the product's minimum witness vector. Finally, we output $B \cdot r = [v_1 \cdots v_{d'}]^T$ and $w = [w_1 \ldots w_{d'}]$. Updating w_j can happen at most $\varepsilon \log n$ times, because each $w_j[q]$ is set at most once for $q = 1, \ldots, \varepsilon \log n$. Each individual update takes $O(\log n)$ time. Hence, for each j, the updates to w_j take $O(\log^2 n)$ time, and over all j, the minimum witness computation takes $O(d \log n)$. Computing $v_j = B^j \cdot r$ for a fixed j takes $O(\sum_{i=1}^{n'} \lceil m_{ri}/\kappa \rceil) \leq O(\sum_{i=1}^{n'} (1 + m_{ri}/\kappa))$ time. Over all $j = 1, \ldots, d/(\varepsilon \log n)$, the computation of $B \cdot r$ takes asymptotically

$$\frac{d}{\log n} \sum_{i=1}^{n/\ell} \left(1 + \frac{m_{ri}}{\kappa}\right) \leq \frac{d}{\log n} \left(\frac{n}{\ell} + \frac{m_r}{\kappa}\right).$$

In total, the running time is $O(d \log n + \frac{d}{\log n}(\frac{n}{\ell} + \frac{m_r}{\kappa}))$. □

Let us demonstrate what the data structure performance looks like with a particular setting of the parameters ℓ and k. From Jensen's inequality we have:

Fact 1. *$H(a/b) \leq 2a/b \log(b/a)$, for $b \geq 2a$.*

Corollary 1. *Given a parameter m and $0 < \varepsilon < 1$, any $d \times n$ Boolean matrix A can be preprocessed in $O(dn^{1+\varepsilon})$ time, so that every subsequent computation of AB can be determined in $O(de \log n + \frac{md \log(ne/m)}{\log^2 n})$ time, for any $n \times e$ Boolean matrix B with at most m nonzeros.*

Proof. When $m \geq \frac{en}{2}$, the runtime in the theorem is $\Omega(end/\log^2 n)$, and can be achieved via Four Russians processing. If $m \leq en^{1-\varepsilon}$ then $\varepsilon \log n \leq \log \frac{en}{m}$ and running a standard sparse matrix multiplication with $\log n$ bit operations in $O(1)$ time achieves $O(md/\log n) \leq O(md\frac{\log(en/m)}{\log^2 n})$ time. If $en^{1-\varepsilon} < m < \frac{en}{2}$, apply Theorem 1 with $\ell = \varepsilon \frac{en}{m} \cdot (\log n)/\log(\frac{en}{m})$, and $\kappa = \varepsilon(\log n)/\log(\frac{en}{m}) \geq 1$. Then by Fact 1 and $m < \frac{en}{2}$, we can show that $\binom{\ell}{\kappa} \leq n^{\varepsilon}$. Hence the preprocessing step takes $O(dn^{1+\varepsilon}m/(ne)) = O(dn^{1+\varepsilon})$ time. Matrix-vector multiplication with a vector of m_i nonzeros takes $O\left(d \log n + \frac{d}{\log n}\left(\frac{m \log(en/m)}{e \log n} + \frac{m_i \log(en/m)}{\log n}\right)\right)$ time. If m_i is the number of nonzeros in the ith column of B, as $\sum_i m_i = m$, the full matrix product $A \cdot B$ can be done in $O\left(de \log n + \frac{md \log(en/m)}{\log^2 n}\right)$ time. □

It follows that Boolean matrix multiplication for $n \times n$ matrices can be done in $O(n^{2+\varepsilon} + mn \log \frac{n^2}{m}/\log^2 n)$ time, provided that at least one of the matrices has only m nonzeros. We note that such a result could also be obtained by construing the sparse matrix as a bipartite graph, applying Feder-Motwani's graph compression to represent the sparse matrix as a product of two sparser matrices (plus a matrix with $n^{2-\delta}$ nonzeros), then using an $O(mn/\log n)$ Boolean matrix multiplication algorithm on the resulting matrices. However, given the complexity of this procedure (especially the graph compression, which is involved) we believe that our method is more practical. Theorem 1 also leads to two other new results almost immediately.

Minimum Witnesses for Matrix Multiplication. The *minimum witness* product of two $n \times n$ Boolean matrices A and B is the $n \times n$ matrix C defined as $C[i][j] = \min_{k=1}^{n}\{k \mid A[i][k] \cdot B[k][j] = 1\}$, for every $i, j \in [n]$. This product has applications to several graph algorithms. It has been used to compute all pairs least common ancestors in DAGs [12,7], to find minimum weight triangles in node-weighted graphs [17], and to compute all pairs bottleneck paths in a node weighted graph [16]. The best known algorithm for the minimum witness product is by Czumaj, Kowaluk and Lingas [12,7] and runs in $O(n^{2.575})$ time. However, when one of the matrices has at most $m = o(n^{1.575})$ nonzeros, nothing better than $O(mn)$ was known (in terms of m) for combinatorially computing the minimum witness product. As an immediate corollary of Theorem 1, we obtain the first asymptotic improvement for sparse matrices: an algorithm with $O(mn \log(n^2/m)/\log^2 n)$ running time.

Corollary 2. *Given $n \times n$ Boolean matrices A and B, where B has at most m nonzero entries, all pairs minimum witnesses of $A \cdot B$ can be computed in $O(n^2 + mn \log(n^2/m)/\log^2 n)$ time.*

Minimum Weighted Triangles. The problem of computing for all pairs of nodes in a node-weighted graph a triangle (if any) of minimum weight passing through both nodes can be reduced to finding minimum witnesses as follows ([17]). Sort the nodes in order of their weight in $O(n \log n)$ time. Create the adjacency matrix A of the graph so that the rows and columns are in the sorted order of the vertices. Compute the minimum witness product C of A. Then, for every edge $(i, j) \in G$, the minimum weight triangle passing through i and j is $(i, j, C[i][j])$. From this reduction and Corollary 2 we obtain the following.

Corollary 3. *Given a graph G with m edges and n nodes with arbitrary node weights, there is an algorithm which finds for all pairs of vertices i, j, a triangle of minimum weight sum going through i, j in $O(n^2 + mn \log(\frac{n^2}{m})/\log^2 n)$ time.*

4 Transitive Closure

The transitive closure matrix of an n node graph G is the $n \times n$ Boolean matrix A for which $A[i][j] = 1$ if and only if node i is reachable from node j in G. In terms of n, the complexity of computing the transitive closure of an n node graph is equivalent to that of multiplying two Boolean $n \times n$ matrices [1], thus the best known algorithm in terms of n is $O(n^\omega)$. However, when the sparsity of G is taken into account, it is unclear how to use an algorithm for sparse matrix multiplication to solve transitive closure in the same runtime. While Feder and Motwani's [9] graph compression implies an $O(mn \log(\frac{n^2}{m})/\log^2 n)$ algorithm for sparse matrix product, this result gives little insight on how to compute the transitive closure of a sparse graph—since the number of edges in the transitive closure is independent of m in general, maintaining a graph compression will not suffice. In contrast, the data structure of Theorem 1 is perfectly suited for use with a dynamic programming algorithm for transitive closure.

Theorem 2. *Transitive closure can be computed in $O(n^2 + mn \log(\frac{n^2}{m})/\log^2 n)$ time on graphs with n nodes and m edges.*

Proof. We first compute the strongly connected components of G in $O(m + n)$ time. We then contract them in linear time to obtain a DAG G'. Clearly, if we have the transitive closure matrix of G', we can recover the transitive closure matrix of G with an extra $O(n^2)$ additive overhead: for every edge (u, v) in the transitive closure graph of G', go through all pairs of vertices (i, j) such that i is in u and j is in v and add 1 to the transitive closure matrix of G. Hence it suffices for us to compute the transitive closure of a DAG G'.

First, we topologically sort G' in $O(m + n)$ time. Our transitive closure algorithm is based on a dynamic programming formulation of the problem given by Cheriyan and Mehlhorn [5], later also mentioned by Chan [3]. The algorithm proceeds in n iterations; after the kth iteration, we have computed the transitive closure of the last k nodes in the topological order. At every point, the current transitive closure is maintained in a Boolean matrix R, such that $R[u][v] = 1$ iff u is reachable from v. Let $R[\cdot][v]$ denote column v of R. Let p be the $(k + 1)$st

node in reverse topological order. We wish to compute $R[\cdot][p]$, given all the vectors $R[\cdot][u]$ for nodes u after p in the topological order. To do this, we compute the componentwise OR of all vectors $R[\cdot][u]$ for the neighbors u of p.

Suppose R is a matrix containing columns $R[\cdot][u]$ for all u processed so far, and zeros otherwise. Let v_p be the *outgoing neighborhood* vector of p: $v_p[u] = 1$ iff there is an arc from p to u. Construing v_p as a column vector, we want to compute $R \cdot v_p$. Since all outgoing neighbors of p are after p in the topological order, and we process nodes in reverse order, correctness follows.

We now show how to implement the algorithm using the data structure of Theorem 1. Let R be an $n \times n$ matrix such that at the end of the algorithm R is the transitive closure of G'. We begin with R set to the all zero matrix. Let $\kappa \geq 1$, and $\ell > \kappa$ be parameters to be set later. After $O((n^2\kappa/\ell)\sum_{b=1}^{\kappa}\binom{\ell}{b})$ preprocessing time, the data structure for R from Theorem 1 is complete.

Consider a fixed iteration k. Let p be the kth node in reverse topological order. As before, v_p is the neighborhood column vector of p, so that v_p has $outdeg(p)$ nonzero entries. Let $0 < \varepsilon < 1$ be a constant. We use the data structure to compute $r_p = R \cdot v_p$ in $O(n^{1+\varepsilon} + n^2/(\ell \log n) + outdeg(p) \cdot n/(\kappa \log n))$ time. Then we set $r_p[p] = 1$, and $R[\cdot][p] := r_p$, by performing a column update on R in $O(n\kappa\sum_{b=1}^{\kappa}\binom{\ell}{b})$ time. This completes iteration k. Since there are n iterations and since $\sum_p outdeg(p) = m$, the overall running time is asymptotic to

$$\frac{n^2\kappa}{\ell}\sum_{b=1}^{\kappa}\binom{\ell}{b} + n^{2+\varepsilon} + \frac{n^3}{\ell \log n} + \frac{mn}{\kappa \log n} + n^2\kappa\sum_{b=1}^{\kappa}\binom{\ell}{b}.$$

If for some $\varepsilon' > 0$, $m \leq n^{2-\varepsilon'}$, then we can ignore the $O(n^{2+\varepsilon})$ preprocessing step and execute the original algorithm, in $O(mn/\log n) \leq O(mn \log \frac{n^2}{m}/\log^2 n)$ time. Otherwise, we set ℓ and κ to minimize the running time. Similar to Corollary 1, we set $\frac{n^3}{\ell \log n} = \frac{mn}{\kappa \log n}$, (implying $\ell = \kappa n^2/m$), and $n^{\varepsilon} = \sum_{b=1}^{\ell m/n^2}\binom{\ell}{b}$. For $m \leq n^2/2$, $\sum_{b=1}^{m\ell/n^2}\binom{\ell}{b} = O(2^{\ell H(m/n^2)}) = O(2^{\ell \frac{m}{n^2}\log\frac{n^2}{m}})$. Hence we want $\ell\frac{m}{n^2}\log\frac{n^2}{m} = \log n^{\varepsilon}$, and $\ell = \varepsilon'\frac{n^2\log n}{m\log(n^2/m)}$, $\kappa = \varepsilon'\frac{\log n}{\log(n^2/m)}$ for $\varepsilon' < \varepsilon$ suffices. Since for all $\varepsilon' > 0$, $m \geq n^{2-\varepsilon'}$, we can pick any $\varepsilon' < \varepsilon$ and we will have $\kappa \geq 1$. For sufficiently small ε' the runtime is $\Omega(n^{2+\varepsilon})$, and the final runtime is $O(n^2 + mn\log(n^2/m)/\log^2 n)$. □

5 APSP on Unweighted Undirected Graphs

Our data structure can also be applied to solve all pairs shortest paths (APSP) on unweighted undirected graphs, yielding the fastest algorithm for sparse graphs to date. Chan [4] gave two algorithms for this problem, which constituted the first major improvement over the $O(mn\frac{\log(n^2/m)}{\log n} + n^2)$ obtained by Feder and Motwani's graph compression [9]. The first algorithm is deterministic running in $O(n^2 \log n + mn/\log n)$ time, and the second one is Las Vegas running in $O(n^2 \log^2 \log n/\log n + mn/\log n)$ time on the RAM. To prove the following Theorem 3, we implement Chan's first algorithm, along with some modifications.

Theorem 3. *The All Pairs Shortest Paths problem in unweighted undirected graphs can be solved in* $O(n^{2+\varepsilon} + mn\log(n^2/m)/\log^2 n)$ *time, for any* $\varepsilon > 0$.

By running our algorithm when $m = \Omega(n^{1+\varepsilon}\log^2 n)$ for some $\varepsilon > 0$ and Chan's second algorithm otherwise, we obtain the following result.

Theorem 4. *On the probabilistic RAM, the APSP problem on unweighted undirected graphs can be solved in* $O(n^2\log^2\log n/\log n + mn\log(n^2/m)/\log^2 n)$ *time, with high probability.*

To be able to prove Theorem 3, let us focus on a particular part of Chan's first algorithm that produces the bottleneck in its runtime. The algorithm contains a subprocedure $P(d)$, parameterized by an integer $d < n$. The input to $P(d)$ is graph $G = (V, E)$, a collection of n/d vertices $\{s_1, \ldots, s_{n/d}\}$, and a collection of n/d *disjoint* vertex subsets $\{S^1, \ldots, S^{n/d}\}$, such that $\forall\ i \in [n/d]$, every $s \in S^i$ has distance at most 2 from s_i. $P(d)$ outputs the $|\cup_i S^i| \times n$ matrix D, such that for every $s \in \cup_i S^i$ and $v \in V$, $D[s][v] = \delta_G(s, v)$. Notice, $|\cup_i S^i| \leq n$. This procedure $P(d)$ is significant for the following reason:

Lemma 1 (Implicit in Chan [4]). *If* $P(d)$ *can be implemented in* $O(T(P(d)))$ *time, then APSP on unweighted undirected graphs is in* $O(T(P(d)) + n^2 \cdot d)$ *time.*

Setting $d = n^{\varepsilon}$, Theorem 3 is obtained from the following lemma.

Lemma 2. $P(d)$ *is implementable in* $O(n^{2+\varepsilon} + \frac{mn}{d} + \frac{mn\log(n^2/m)}{\log^2 n})$ *time,* $\forall \varepsilon > 0$.

Proof. First, we modify Chan's original algorithm slightly. As in Chan's algorithm, we first do breadth-first search from every s_i in $O(mn/d)$ time overall. When doing this, for each distance $\ell = 0, \ldots, n-1$, we compute the sets $A^i_\ell = \{v \in V \mid \delta_G(s_i, v) = \ell\}$. Suppose that the rows (columns) of a matrix M are in one-to-one correspondence with the elements of some set U. Then, for every $u \in U$, let $\bar{u}$ be the index of the row (column) of M corresponding to u.

Consider each (s_i, S^i) pair separately. Let $k = |S^i|$. For $\ell = 0, \ldots, n-1$, let $B_\ell = \cup_{j=\ell-2}^{\ell+2} A^i_\ell$, where $A^i_j = \{\}$ when $j < 0$, or $j > n-1$.

The algorithm proceeds in n iterations. Each iteration $\ell = 0, \ldots, n-1$ produces a $k \times |B_\ell|$ matrix D_ℓ, and a $k \times n$ matrix OLD (both Boolean), such that for all $s \in S^i$, $u \in B_\ell$ and $v \in V$,

$$D_\ell[\bar{s}][\bar{u}] = 1 \text{ iff } \delta_G(s, u) = \ell \text{ and } \mathrm{OLD}[\bar{s}][\bar{v}] = 1 \text{ iff } \delta_G(s, v) \leq \ell.$$

At the end of the last iteration, the matrices D_ℓ are combined in a $k \times n$ matrix D^i such that $D[\bar{s}][\bar{v}] = \ell$ iff $\delta_G(s, v) = \ell$, for every $s \in S^i$, $v \in V$. At the end of the algorithm, the matrices D^i are concatenated to create the output D.

In iteration 0, create a $k \times |B_0|$ matrix D_0, so that for every $s \in S^i$, $D_0[\bar{s}][\bar{s}] = 1$, and all 0 otherwise. Let OLD be the $k \times n$ matrix with $\mathrm{OLD}[\bar{s}][\bar{v}] = 1$ iff $v = s$.

Consider a fixed iteration ℓ. We use the output ($D_{\ell-1}$, OLD) of iteration $\ell - 1$. Create a $|B_{\ell-1}| \times |B_\ell|$ matrix N_ℓ, such that $\forall u \in B_{\ell-1}, v \in B_\ell$, $N_\ell[\bar{u}][\bar{v}] = 1$ iff v is a neighbor of u. Let N_ℓ have m_ℓ nonzeros. If there are b_ℓ edges going out of B_ℓ, one can create N_ℓ in $O(b_\ell)$ time: Reuse an $n \times n$ zero matrix N, so that N_ℓ

will begin at the top left corner. For each $v \in B_\ell$ and edge (v,u), if $u \in B_{\ell-1}$, add 1 to $N[\bar{u}][\bar{v}]$. When iteration ℓ ends, zero out each $N[\bar{u}][\bar{v}]$.

Multiply $D_{\ell-1}$ by N_ℓ in $O(k \cdot |B_{\ell-1}|^{1+\varepsilon} + m_\ell k \log(\frac{n^2}{m})/\log^2 n)$ time (applying Corollary 1). This gives a $k \times |B_\ell|$ matrix A such that for all $s \in S^i$ and $v \in B_\ell$, $A[\bar{s}][\bar{v}] = 1$ iff there is a length-ℓ path between s and v. Compute D_ℓ by replacing $A[\bar{s}][\bar{v}]$ by 0 iff $\mathrm{OLD}[\bar{s}][\bar{v}] = 1$, for each $s \in S^i, v \in B_\ell$. If $D_\ell[\bar{s}][\bar{v}] = 1$, set $\mathrm{OLD}[\bar{s}][\bar{v}] = 1$.

Computing the distances from all $s \in S^i$ to all $v \in V$ can be done in $O(m + \sum_{\ell=1}^{n-1} \left(k \cdot |B_{\ell-1}|^{1+\varepsilon} + m_\ell k \log(n^2/m)/\log^2 n + b_\ell\right))$ time. However, since every node appears in at most $O(1)$ sets B_ℓ, $\sum_{\ell=0}^{n-1} |B_\ell| \leq O(n)$, $\sum_\ell b_\ell \leq O(m)$ and $\sum_\ell m_\ell \leq O(m)$. The runtime becomes $O(kn^{1+\varepsilon} + mk\log(n^2/m)/\log^2 n + m)$. When we sum up the runtimes for each (s_i, S^i) pair, since the sets S^i are disjoint, we obtain asymptotically

$$\sum_{i=1}^{n/d} \left(m + |S^i| \cdot \left(n^{1+\varepsilon} + \frac{m\log(n^2/m)}{\log^2 n}\right)\right) = \frac{mn}{d} + n^{2+\varepsilon} + \frac{mn\log(n^2/m)}{\log^2 n}.$$

As the data structure returns witnesses, we can also return predecessors. □

6 All Pairs Least Common Ancestors in a DAG

To our knowledge, the best algorithm in terms of m and n for finding all pairs least common ancestors (LCAs) in a DAG, is a dynamic programming algorithm by Czumaj, Kowaluk and Lingas [12,7] which runs in $O(mn)$ time. We improve the runtime of this algorithm to $O(mn\log(n^2/m)/\log n)$ using the following generalization of Theorem 1, the proof of which is omitted. Below, for an $n \times n$ real matrix B and $n \times 1$ Boolean vector r, $B \odot r$ is the vector c with $c[i] = \max_{k=1,\ldots,n}(A[i][k] \cdot r[k])$ for each $i \in [n]$. This product clearly generalizes the Boolean matrix-vector product.

Theorem 5. *Let B be a $d \times n$ matrix with β-bit entries. Let $0 < \varepsilon < 1$ be constant, and let $\kappa \geq 1$ and $\ell > \kappa$ be integer parameters. Then one can create a data structure with $O(\frac{dn\kappa}{\ell} \cdot \lceil\frac{\beta}{\log n}\rceil \cdot \sum_{b=1}^{\kappa}\binom{\ell}{b})$ preprocessing time so that the following operations are supported on a pointer machine:*

- *given any $n \times 1$ binary vector r, output $B \odot r$ in $O(dn^\varepsilon + \frac{d\beta}{\log n}\left(\frac{n}{\ell} + \frac{m_r}{\kappa}\right))$ time, where m_r is the number of nonzeros in r;*
- *replace any column of B by a new column in $O(d\kappa\lceil\frac{\beta}{\log n}\rceil\sum_{b=1}^{\kappa}\binom{\ell}{b})$ time.*

Due to space limitations, the proof of the following is also omitted.

Theorem 6. *The all pairs least common ancestors problem on n node and m edge DAGs can be solved in $O(mn\log(n^2/m)/\log n)$ time.*

7 Conclusion

We have introduced a new combinatorial data structure for performing matrix-vector multiplications. Its power lies in its ability to compute sparse vector products quickly and tolerate updates to the matrix. Using the data structure, we gave new running time bounds for four fundamental graph problems: transitive closure, all pairs shortest paths on unweighted graphs, maximum weight triangle, and all pairs least common ancestors.

Acknowledgments. This research was supported in part by NSF ITR grant CCR-0122581 (The Aladdin Center).

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.: The design and analysis of computer algorithms. Addison-Wesley Longman Publishing Co., Boston (1974)
2. Arlazarov, V.L., Dinic, E.A., Kronrod, M.A., Faradzev, I.A.: On economical construction of the transitive closure of an oriented graph. Soviet Math. Dokl. 11, 1209–1210 (1970)
3. Chan, T.M.: All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 318–324. Springer, Heidelberg (2005)
4. Chan, T.M.: All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. In: Proc. SODA, pp. 514–523 (2006)
5. Cheriyan, J., Mehlhorn, K.: Algorithms for dense graphs and networks on the random access computer. Algorithmica 15(6), 521–549 (1996)
6. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symbolic Computation 9(3), 251–280 (1990)
7. Czumaj, A., Kowaluk, M., Lingas, A.: Faster algorithms for finding lowest common ancestors in directed acyclic graphs. TCS 380(1–2), 37–46 (2007)
8. Czumaj, A., Lingas, A.: Finding a heaviest triangle is not harder than matrix multiplication. In: Proc. SODA, pp. 986–994 (2007)
9. Feder, T., Motwani, R.: Clique partitions, graph compression and speeding-up algorithms. In: Proc. STOC, pp. 123–133 (1991)
10. Fischer, M.J., Meyer, A.R.: Boolean matrix multiplication and transitive closure. In: Proc. FOCS, pp. 129–131 (1971)
11. Galil, Z., Margalit, O.: All pairs shortest paths for graphs with small integer length edges. JCSS 54, 243–254 (1997)
12. Kowaluk, M., Lingas, A.: LCA Queries in Directed Acyclic Graphs. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 241–248. Springer, Heidelberg (2005)
13. Munro, J.I.: Efficient determination of the transitive closure of a directed graph. Inf. Process. Lett. 1(2), 56–58 (1971)
14. Rytter, W.: Fast recognition of pushdown automaton and context-free languages. Information and Control 67(1–3), 12–22 (1985)
15. Seidel, R.: On the all-pairs-shortest-path problem in unweighted undirected graphs. JCSS 51, 400–403 (1995)

16. Shapira, A., Yuster, R., Zwick, U.: All-pairs bottleneck paths in vertex weighted graphs. In: Proc. SODA, pp. 978–985 (2007)
17. Vassilevska, V., Williams, R., Yuster, R.: Finding the Smallest H-Subgraph in Real Weighted Graphs and Related Problems. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 262–273. Springer, Heidelberg (2006)
18. Williams, R.: Matrix-vector multiplication in sub-quadratic time (some preprocessing required). In: Proc. SODA, pp. 995–1001 (2007)

How to Explore a Fast-Changing World
(Cover Time of a Simple Random Walk on Evolving Graphs)

Chen Avin[1], Michal Koucký[2], and Zvi Lotker[1]

[1] Communication Systems Engineering, Ben Gurion University of the Negev, Israel
{avin,zvilo}@cse.bgu.ac.il
[2] Institute of Mathematics, Academy of Sciences of the Czech Republic
koucky@math.cas.cz

Abstract. Motivated by real world networks and use of algorithms based on random walks on these networks we study the simple random walks on *dynamic* undirected graphs with fixed underlying vertex set, i.e., graphs which are modified by inserting or deleting edges at every step of the walk. We are interested in the expected time needed to visit all the vertices of such a dynamic graph, the *cover time*, under the assumption that the graph is being modified by an oblivious adversary. It is well known that on connected *static* undirected graphs the cover time is polynomial in the size of the graph. On the contrary and somewhat counter-intuitively, we show that there are adversary strategies which force the expected cover time of a simple random walk on connected dynamic graphs to be exponential. We relate this result to the cover time of static directed graphs. In addition we provide a simple strategy, the *lazy* random walk, that guarantees polynomial cover time regardless of the changes made by the adversary.

1 Introduction

A random walk on a graph is a simple process of visiting the nodes of the graph in some random sequential order. The walk starts at some fixed node, and at each step it moves to a neighbor of the current node chosen at random. The random walk is called *simple* when the next node is chosen uniformly at random from the set of neighbors. In the context of communication networks (e.g., Internet, wireless ad-hoc networks and sensor networks) and information networks (e.g., peer-to-peer file sharing networks and distributed databases), a random walk on a network (graph) will result when messages are sent at random from device to device.

Since this process presents locality, simplicity, low memory-overhead and robustness to changes in the network structure applications based on random-walk techniques are becoming more and more popular in the networking community. In recent years, different authors have proposed the use of random walk for a large variety of tasks and networks; to name but a few: querying in sensor and ad-hoc networks [18,5,1], searching in peer-to-peer networks [12], gossiping [16], PageRank and search engines on the web [13].

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 121–132, 2008.

One of the main reasons that random walk techniques are so appealing for networking application is their robustness to dynamics. Many communication networks are subject to dramatic structural changes created by mobility, sleep modes, channel fluctuations, device failures, nodes joining or leaving the system and other factors. Topology-driven algorithms are at a disadvantage for such networks, since they incur high overhead to maintain up-to-date topology and routing information such as routing tables, clusters and spanning trees. In contrast, algorithms that require no knowledge of network topology, such as the random walk, are at an advantage.

While at first glance, the process of a token wandering randomly in the network may seem overly simplistic and highly inefficient, many encouraging results prove that it is comparable to other approaches that have been used over the years. One important property of random walks on graphs that needs to be evaluated to study the efficiency of the approach is the *cover time* [2]. The *cover time* C_G of a graph G is the expected time (measured by number of steps or in our case by the number of messages) taken by a simple random walk to visit all the nodes in G. Methods on bounding the cover time of graphs have been thoroughly investigated with the major result being that cover time is always at most polynomial for undirected graphs. More precisely, it has been shown by Aleliunas *et al.* in their seminal work [3] that C_G is always $O(mn)$, where m is the number of edges in the graph and n is the number of nodes. Tighter bounds for many classes of graphs have been established and they can be found in the extensive literature on the subject.

Since real-world networks change over time researches have recently started to study random walks on such dynamic graphs. Motivated by robotic exploration of the Web, Cooper and Frieze [7] studied the question of covering a graph that grows over time. They considered a particular model of so-called *web graphs* and showed that a simple random walk on the graph fails to visit a constant fraction of nodes if a new node appears and is connected to the graph after every constant number of steps of the walk.

Motivated by sensor networks we consider a similar question on a different model of dynamic graphs. We consider dynamic graphs with fixed number of nodes where connections between the nodes appear and disappear over the time. The question that we study is the cover time of such graphs.

1.1 Overview of Our Results

We show that somewhat counter-intuitively, there are dynamic graphs of the type given above that have exponential cover time when explored by a simple random walk. (For the sake of clarity let us say that our examples are deterministic but oblivious to the actual random walk.) Moreover, we show that a random walk on any directed graph G can be simulated (in a way we define later) by a random walk on an undirected dynamic graph that we construct from G; this gives yet another justification to our previous claim. Our examples are also valid when we allow the random walk to make more than a single step between each graph change. Indeed, we can allow up-to $n^{1-\epsilon}$ steps before making each change and

still obtain an exponential cover time. Although one could question whether our graphs could appear in a real-word scenario we do not consider these graphs to be far-fetched: for example a particular implementation of sensor networks with links (network interfaces) going to sleep periodically or nodes switching communication frequencies could exhibit such behavior.

In addition to these negative results we also show several positive results. Most importantly we show that a *lazy* random walk (also known as a *max-degree* random walk in literature [15]) does not suffer from these issues. We define as a lazy random walk a walk that picks each adjacent edge with probability $1/d_{\max}$, where $d_{\max}$ is the maximum degree of the graph, and with the remaining probability it stays at the current vertex. We show that a lazy random walk covers any connected dynamic graph in time polynomial in the size of the graph. Furthermore, we also show that when the dynamic graph itself is obtained by sampling from a certain probability distribution, a simple random walk will also cover such a graph in expected polynomial time.

2 Models and Preliminaries

2.1 Random Walks on Graphs

Let $G(V, E)$ be an undirected graph, with V the set of nodes and E the set of edges. Let $n = |V|$ and $m = |E|$. For $v \in V$, let $N(v) = \{u \in V \mid (v, u) \in E\}$ be the set of neighbors of v and $d(v) = |N(v)|$ the degree of v. A d-regular graph is a graph in which the degree of all the nodes is d.

The *simple random walk* on a graph G is a walk on G where the next node is chosen uniformly at random from the set of neighbors of the current node, i.e., when the walk is at node v, the probability to move in the next step to u is $P(v, u) = \frac{1}{d(v)}$ for $(v, u) \in E$ and 0 otherwise.

The *hitting time*, H_{uv}, is the expected time for a random walk starting at u to arrive at v for the first time, and the *commute time*, C_{uv}, is the expected time for a random walk starting at u to first arrive at v and then return to u. Let $H_{\max}$ be the maximum hitting time over all the pairs of nodes in G.

The *cover time* C_G of a graph G is the expected time taken by a simple random walk on G to visit all the nodes in G. Formally, for $v \in V$, let C_v be the expected number of steps needed for the simple random walk starting at v to visit all the nodes in G, and the cover time of G is $C_G = \max_v C_v$. The *cover time* of graphs and methods of bounding it have been extensively investigated [17]. Results for the cover time of specific graphs vary from the *optimal cover time* of $\Theta(n \log n)$ associated with the complete graph to the worst case of $\Theta(n^3)$ associated with the lollipop graph [9,8].

2.2 Evolving Graphs Model

The most general model to describe a dynamic network is called the Evolving Graph model. We will use a similar definition as in [14,11,10].

Definition 1 (Evolving Graphs). *Let $\mathcal{G} = G_1, G_2, \ldots$ be an infinite sequence of graphs on the same vertex set V. We call this sequence an* evolving graph. *We say that $\mathcal{G}$ has the graph property X if every graph G_i in the sequence has the property X.*

In simple words at time i the structure of the evolving graph $\mathcal{G}$ is G_i. For an integer $\tau \geq 1$, an evolving graph $\mathcal{G}$ is *evolving with rate* $\frac{1}{\tau}$ if for all $i \geq 1$, $G_i \neq G_{i+1}$ implies, $G_{i+1} = G_{i+1+j}$ for all $j \in \{0, \ldots, \tau - 1\}$.

A simple random walk on evolving graph $\mathcal{G}$ is defined as follows: assume that at time i the walker is at node $v \in V$, and let $N(v)$ be the set of neighbors of v in G_i, then the walker moves to one of its neighbors from $N(v)$ uniformly at random.

The strength and the weakness of the above model have the same origin, its generality. On the positive side, it captures many interesting scenarios of dynamic networks, but on the other hand, most natural problems are NP-complete such as finding strongly connected components and the equivalence of minimum spanning tree [10].

2.3 Constructive Evolving Graphs Model

Evolving graphs do not capture the underlying mechanism of how (or why) the graph evolves. In many situations the evolving graph itself is a product of some random process. (For example this is the case of web graphs considered in [7].) We will use the following definition to capture the underlying process in the case it is a Markov chain. A special case of such graphs is considered in Section 5.

Definition 2 (Markovian Evolving Graphs). *Let the* space set $\mathbf{G}$ *be a set of graphs with the same set V of nodes, let $G_1 \in \mathbf{G}$ and let P be a probability transition matrix on $\mathbf{G}$. A* Markovian evolving graph $\mathcal{M} = (\mathbf{G}, G_1, P)$ *is an evolving graph $\mathcal{M}_1, \mathcal{M}_2, \ldots$ obtained by the Markov chain given by P with the initial state G_1. Thus for any sequence of graphs $G_2, G_3, \cdots \in \mathbf{G}$ and any $t > 1$, $\Pr[\mathcal{M}_t = G_t \mid \mathcal{M}_{t-1} = G_{t-1}]$ is given by the appropriate entry of P.*

It is clear that a Markovian evolving graph is a random variable. By a random walk on a Markovian evolving graph we understand a random walk on the outcome of the random variable. When considering the expected cover time of a random walk on a Markovian evolving graph we consider the expectation over the choice of both the evolving graph and the random walk.

3 Exponential Hitting Time of Evolving Graphs

In this section we address the cover time of the simple random walk on evolving graphs by studying the maximum hitting time. Clearly the cover time must be at least as large as the maximum hitting time. First, we mention some technical issues. On static graphs the cover time is finite only for connected graphs. This is not the case for evolving graphs as we will see in Section 5. For simplicity though

we restrict our discussion mostly to evolving graphs in which every graph in the sequence $\mathcal{G}$ is connected. (In the Markovian model we require that every graph in the set $\mathbf{G}$ is connected, and call $\mathbf{G}$ *connected* if that is the case). Moreover we require that all graphs have a self-loop for each of the nodes. This is simply a technical condition to avoid pathological cases such as the walk switching forever between two nodes. In the case of static graphs this is a standard way of enforcing ergodicity. An evolving graph $\mathcal{G}$ that has the above properties we call an *explorable* evolving graph.

Now one can easily claim the following for explorable evolving graphs (a similar claim can be made for Markovian evolving graphs):

Claim 3. *Let $\mathcal{G}$ be an explorable evolving graph, then the cover time of $\mathcal{G}$ is bounded by $n^{O(n)}$.*

We outline the argument here. Let V be the set of vertices of $\mathcal{G}$. Fix two vertices u and v from V. For $i \geq 1$, let V_i be the set of vertices that could be visited within first i steps of a simple random walk on $\mathcal{G}$ starting from u. Since $\mathcal{G}$ is connected it must be the case that for each i, $V_i \subsetneq V_{i+1}$ unless $V_i = V$. In particular, V_n must contain all vertices of $\mathcal{G}$ and in particular v. Thus, the probability of reaching v starting from u is at least n^{-n}. Indeed, this is true for any two vertices u and v and starting from any time t. A standard argument now implies that the cover time is at most $n^{O(n)}$.

Requiring connectivity at each step of the evolving graph may look like a very strong condition that should imply polynomial cover time and maximum hitting time. Surprisingly we show that this is not the case.

Theorem 4. *There exists an explorable evolving graph $\mathcal{G}$, such that the maximum hitting time of the simple random walk on $\mathcal{G}$ is $\Omega(2^n)$.*

One can think of this result in the following way: consider a random walk on an evolving graph that is controlled by an oblivious *adversary* that is deciding what will be the next graph at each time step. In such a case the adversary, although unaware of the random walk location, can force the walk to step exponential number of steps before exploring the whole graph. We give below the basic details of the proof.

Proof of Theorem 4. Let G_1 be the star of size n (with the addition of a self-loop at each node) where nodes called $1, 2, \ldots, n-2$ and n are always the leafs and node called $n-1$ is always the center. The random walk starts at the node originally called 1 and we will bound the hitting time to the node called n. The adversary is the following *deterministic* process: At each time step vertices $1, \ldots, n-1$ will trade their places, i.e., the adversary changes the edges by changing the names of the nodes. The adversary uses the following renaming strategy: for $1 \leq i \leq n-1$, node i changes its name to $i + 1 (\mathrm{mod}\, n-1)$. Note that node n does not change its name, nodes $1, \ldots, n-2$ increase their name by one, node $n-1$ becomes 1.

The only way to reach node n is through the center. By induction on $i = 2, \ldots, n-2$ one can see the following. Unless we have already reached the center

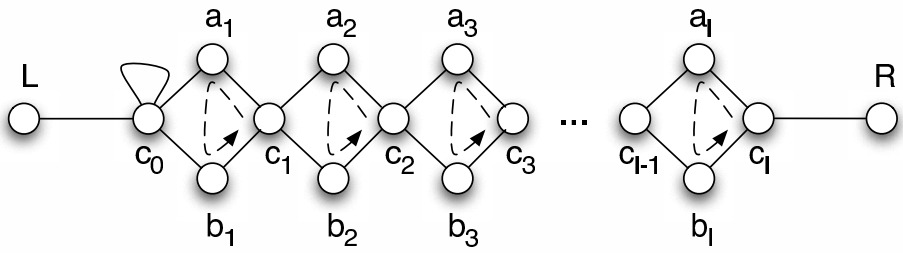


Fig. 1. The gadget $\mathcal{H}_\ell$ (dashed lines show node transformations)

of the star the only way to be at the leaf named i after the adversary move is to be at the leaf named $i-1$ before the adversary renaming. That implies we must have used a self-loop at that random step. Hence, to get to the leaf named $n-2$ we must have had a sequence of $n-3$ random steps all taking a self-loop. To get to the center we have to stay at leaf $n-2$. All in all to be at the center after the adversary move the random walk must have made a sequence of $n-2$ consecutive self-loop steps. That happens with probability 2^{-n+2} in a sequence of $n-2$ consecutive steps. Therefore the expected time before we observe the random walk to make such a sequence of steps is $\Omega(2^n)$. □

We would like to point out that all graphs in $\mathcal{G}$ are isomorphic and rapidly mixing (the cover time of each of them is in fact $O(n \log n)$). This fact shows that common tools like spectral analysis cannot be applied naïvely to dynamic graphs.

3.1 Simulating Directed Graphs

One way to understand the results of the previous section is by relating random walks on explorable evolving graphs to random walks on static directed graphs. In fact we can simulate a simple random walk on a directed graph G by a careful choice of evolving graph $\mathcal{G}$. We will use the following gadget $\mathcal{H}$ to replace every directed edge of G. For $\ell > 0$, the gadget $\mathcal{H}_\ell$ is a sequence of graphs $H_\ell^0, H_\ell^1, H_\ell^2, H_\ell^0, H_\ell^1, H_\ell^2, H_\ell^0, \ldots$ with vertices L, R, s_0 and $s_{i,j}$, for $i = 1, \ldots, \ell$ and $j = 0, 1, 2$. The graph H_ℓ^k is obtained from the graph in Fig. **??** by mapping vertices $L \to L$, $R \to R$, $s_0 \to c_0$ and $s_{i,k} \to c_i$, $s_{i,k+1 \bmod 3} \to b_i$, $s_{i,k+2 \bmod 3} \to a_i$, $i = 1, \ldots, \ell$. (We deviate here from our convention of having self-loops at every node for the sake of simplicity of the analysis. As it will be clear in the next section with minor modification of bounds our claims would be true even if we would add a self-loop to every node.) The main property of a simple random walk on $\mathcal{H}$ is summarized in the following lemma.

Lemma 5. *Let $\ell > 0$, $\mathcal{H}_\ell = H_\ell^0, H_\ell^1, H_\ell^2, H_\ell^0, H_\ell^1, H_\ell^2, H_\ell^0, \ldots$ and $\epsilon = \ell(1/2)^\ell + (3/4)^\ell$. Consider a simple random walk on $\mathcal{H}_\ell$. If the walk starts at vertex L then the probability of returning to L before visiting R is at least $1 - \epsilon$. Moreover if the walk starts at vertex R then the probability of returning to R before visiting L is at most ϵ.*

We omit the proof of this lemma due to space constraints. Thus the gadget $\mathcal{H}_\ell$ has essentially the same effect for a simple random walk as a directed edge from R to L with a self-loop at L. Given a directed graph G with a self-loop at every vertex we can replace all its directed edges between different vertices by a copy of $\mathcal{H}_\ell$ to obtain a sequence of graphs $\mathcal{G}$ on which a simple random walk will simulate a simple random walk on G (up-to some error ϵ). Of course, replacing several edges incoming to a vertex by the gadget will introduce several self-loops to that vertex. To avoid that we can collapse the vertices c_0 from these gadgets into one thus obtaining an equivalent of one self-loop. (This collapse will affect ϵ slightly but no more than by a factor polynomial in the number of replaced edges.) We also remove the original self-loops from the graph G.

If we perform a simple random walk on $\mathcal{G}$ and we restrict ourselves to observing only visits to the vertices of the original graph G we will observe essentially the same probability distribution as of a simple random walk on G. In particular, if we choose $\ell = n^{k+1}$, for $k > 1$ and n being the size of G, then the probability of observing an edge being traversed in the opposite direction in the first 2^{n^k} steps is at most $2^{-O(n^{k+1})}$. Since for example the maximal hitting time on any strongly connected directed graph is bounded by $2^{O(n \log n)}$ this error is negligible.

4 Slowly Evolving Graphs

The previous section has shown that there are evolving graphs for which a simple random walk essentially fails as a means of exploring it. All our examples so far considered graphs that evolve at rate one. This would not really be a typical case in a real-world application. The rate at which graphs evolve is usually slower compared to unit operations such as sending a packet. So could it be the case that a simple random walk covers in polynomial time all graphs evolving at lower rate? In this section we show that this is not the case. Namely for any constant $0 < \epsilon < 1$ and an integer n large enough, we provide an example of an evolving graph on $O(n)$ vertices that evolves at rate $\frac{1}{n^{1-\epsilon}}$ so that a simple random walk needs expected time $2^{\Omega(n^\epsilon)}$ to cover the graph. Indeed the graph is essentially the gadget from the previous section with the speed of evolution slowed down.

Let F^i_ℓ be the graph H^i_ℓ from the previous section modified by adding possibly several self-loops to each vertex so that the probability of staying at the same vertex is precisely one half. (So in particular vertices of degree two will receive two self-loops and vertices of degree four will receive four of them. We remark that our claim would be true even without these self-loops but in some cases for trivial reasons. So to capture the most general situation we introduce the loops.)

For $0 < \epsilon < 1$ and an integer $n \geq 2^{1/(1-\epsilon)}$, we define an evolving graph $\mathcal{G}^\epsilon_n$ to consist of repeated sequence $F^0_{2n}, F^0_{2n}, \ldots, F^0_{2n}, F^1_{2n}, F^1_{2n}, \ldots, F^1_{2n}, F^2_{2n}, \ldots, F^2_{2n}$, where each block of consecutive F^i_{2n}'s consists of $n^{1-\epsilon}$ copies of F^i_{2n}. Clearly $\mathcal{G}^\epsilon_n$ evolves at rate $\frac{1}{n^{1-\epsilon}}$. We claim:

Theorem 6. *The cover time of $\mathcal{G}^\epsilon_n$ is $2^{\Omega(n^\epsilon)}$.*

In order to prove the theorem we analyze a concept that we call a *random walk on a line with a drift*. A random walk on a line with a drift is a simple random walk on a line of size n where each $\ell = n^{1-\epsilon}$ steps, a step biased towards the same direction is taken. We show that such a walk requires in expectation an exponential number of steps to traverse the line in the direction opposite to the bias. Due to space limitations we omit the detailed analysis from this version (see [6] for the proof).

5 Polynomial Cover Time of Dynamic Graphs

We turn our attention to cases where the cover time of evolving graphs is "good", i.e., polynomial. Our first example is of a simple Markovian case.

Definition 7 (Bernoulli evolving graph). *Let* $\mathbf{G}$ *be a set of graphs with the same set* V *of nodes and let* $\bar{P}$ *be a probability distribution over* $\mathbf{G}$*. A Bernoulli evolving graph* $\mathcal{B} = (\mathbf{G}, \bar{P})$ *is a Markovian evolving graph in which the rows of the transition matrix* P *are identical and equal to* $\bar{P}$ *and the initial graph* G_1 *is taken at random according to* $\bar{P}$*, i.e., the random graphs* G_i*, are i.i.d.*

We show that the bound for the cover time of the simple random walk on Bernoulli evolving graphs is very similar to the bound of static graphs; essentially when the process is time invariant and the graph is always connected then the bound of Aleliunas *et al.* [3] can be extended to dynamic graphs.

Theorem 8. *For any explorable Bernoulli evolving graph,* $\mathcal{B} = (\mathbf{G}, \bar{P})$*, the cover time of the simple random walk on* $\mathcal{B}$ *is* $O(n^3 \log n)$ *and the maximum hitting time is* $O(n^3)$*.*

The property that $\mathbf{G}$ is *connected* is not necessary to obtain a polynomial bound on the cover time as the following statement shows. (We omit proofs of both of these theorems due to space limitations.)

Theorem 9. *Let* $\mathcal{B} = (\mathbf{G}, \bar{P})$ *be a Bernoulli evolving graph,* $\mathbf{G}$ *be the set of all maximum matching of the complete graph (any such graph is disconnected) and* $\bar{P}$ *is the uniform distribution over* $\mathbf{G}$*. The cover time of the simple random walk on* $\mathcal{B}$ *is the same as the cover time of the complete graph,* $n \log n(1 + o(1))$*.*

5.1 *d*-Regular Dynamic Graphs

It is known that simple random walks on regular, connected, non-bipartite static graph have cover time of $O(n^2)$ [17]. Interestingly, it turns out that a similar result holds true for regular, connected, non-bipartite evolving graphs.

Theorem 10. *For any d-regular connected non-bipartite evolving graph* $\mathcal{G}$ *the cover time of the simple random walk on* $\mathcal{G}$ *is* $O(d^2 n^3 \ln^2 n)$*.*

We will need the following lemma proof of which omitted is due to space limitations:

Lemma 11. *Let G be an undirected d-regular (multi)graph on n vertices and $p = (p_1, \ldots, p_n)$ be a probability distribution on its vertices. Let A_G be the transition matrix of a simple random walk on G. Then:*

1. $$\left\| pA_G - \frac{\mathbb{I}}{n} \right\|_2^2 \leq \left\| p - \frac{\mathbb{I}}{n} \right\|_2^2 .$$
2. *If G is connected non-bipartite*
$$\left\| pA_G - \frac{\mathbb{I}}{n} \right\|_2^2 \leq \left(1 - \frac{1}{d^2 n^2}\right) \left\| p - \frac{\mathbb{I}}{n} \right\|_2^2 .$$

Here $\mathbb{I}$ stands for a vector of ones of an appropriate dimension.

As an immediate corollary to the previous lemma we obtain:

Corollary 12. *Let $\mathcal{G} = G_1, G_2, \ldots$ be a sequence of d-regular graphs on the same vertex set $V = \{1, \ldots, n\}$. For integers $0 \leq \ell \leq t$ let at least ℓ of the graphs $G_1, \ldots, G_t$ be non-bipartite connected. If p_0 is the initial probability distribution on V and we perform a simple random walk on $\mathcal{G}$ starting from p_0, then the probability distribution p_t of the walk after t steps satisfies:*

$$\left\| p_t - \frac{\mathbb{I}}{n} \right\|_2^2 \leq \left(1 - \frac{1}{d^2 n^2}\right)^{\ell} \left\| p_0 - \frac{\mathbb{I}}{n} \right\|_2^2 .$$

A technique similar to [4] gives the following lemma.

Lemma 13. *Let $Y_0, Y_1, Y_2, \ldots$ be a sequence of random variables with range $V = \{1, \ldots, n\}$ satisfying for all $u, v \in V$ and $i > 0$, $\Pr[Y_i = u | Y_{i-1} = v] \geq 1/2n$. If $t = \min\{i;\ \{Y_0, Y_1, \ldots, Y_i\} = V\}$ then the expectation $E[t] \leq 3n \ln n + O(\sqrt{n} \ln n)$.*

Proof. For every $\ell > 0$ and every $v \in V$, $\Pr[v \notin \{Y_{\ell+1}, \ldots, Y_{\ell+3n \ln n}\}] < (1 - 1/2n)^{3n \ln n} < e^{-(3/2) \ln n} = n^{-3/2}$. Thus, $\Pr[\exists v \in V;\ v \notin \{Y_{\ell+1}, \ldots, Y_{\ell+3n \ln n}\}] < n \cdot n^{-3/2} = 1/\sqrt{n}$. For each integer $k \geq 0$, if we set $\ell = k \cdot 3n \ln n$ then the probability that $Y_{\ell+1}, \ldots, Y_{\ell+3n \ln n}$ does not cover whole V is at most $1/\sqrt{n}$. Thus the expected k before V is covered is at most $1/(1 - 1/\sqrt{n}) = 1 + O(1/\sqrt{n})$. Hence the expected cover time of V is bounded by $E[t] \leq 3n \ln n + O(\sqrt{n} \ln n)$. □

Now, we can prove Theorem 10.

Proof of Theorem 10. Let $X_0, X_1, \ldots$ be a random walk on $\mathcal{G}$. For an integer $i \geq 0$, define $Y_i = X_{i \cdot 4d^2 n^2 \ln n}$. Pick $u, v \in V$. For $i > 1$, let p_i be the probability distribution of Y_i conditioned on $Y_{i-1} = v$. By Corollary 12, $\left\| p_i - \frac{\mathbb{I}}{n} \right\|_2^2 \leq \left(1 - \frac{1}{d^2 n^2}\right)^{4d^2 n^2 \ln n} < n^{-4}$. Hence, all coordinates of the vector $(p_i - \frac{\mathbb{I}}{n})$ are in absolute value smaller than $1/n^2$. Thus $\Pr[Y_i = u | Y_{i-1} = v] \geq \frac{1}{n} - \frac{1}{n^2} \geq 1/2n$, provided that $n \geq 2$. Applying Lemma 13 yields the result. □

6 Random Walk Strategy

Consequently to the previous section the following simple strategy for the random walk guarantees that an evolving graph will be covered in expected polynomial time:

Definition 14 (Lazy Random Walk). *At each step of the walk pick a vertex v from $V(G)$ uniformly at random and if there is an edge from the current vertex to the vertex v then we move to v otherwise we stay at the current vertex.*

In effect what this strategy does is that it makes the graph n-regular; every edge adjacent to the current vertex is picked with the probability $1/n$ and with the remaining probability we use one of many self-loops. If we have an a priori upper bound $d_{\max}$ on the maximum degree of the dynamic graph we can achieve a slightly faster cover time. In that case we can reformulate the strategy as follows:

> At each step of the walk with probability $1 - (d(u)/(d_{\max} + 1))$ stay at the current vertex u and with the remaining probability pick uniformly at random one of the neighbors v of the current vertex and move to v.

We call this strategy $d_{\max}$-*lazy random walk*. (In literature such a walk is sometimes called *max-degree* random walk [15].) If the only upper bound on the maximum degree that we have is n then this strategy becomes the previous one. We claim the following as an immediate corollary of Theorem 10:

Theorem 15. *For any connected evolving graph $\mathcal{G}$ with maximum degree $d_{\max}$ the cover time of the $d_{\max}$-lazy random walk on $\mathcal{G}$ is $O(d_{\max}^2 n^3 \ln^2 n)$.*

Indeed these strategies do not even require the dynamic graph to be connected at each step. By Corollary 12 and Lemma 13 as long as the dynamic graph is connected for polynomial fraction of the time, the cover time of a random walk using our strategy will still be polynomial. In that case we can obtain the following generalization of Theorem 15.

Theorem 16. *Let $\mathcal{G} = G_1, G_2, \ldots$ be an evolving graph with maximum degree $d_{\max}$. Let $\epsilon > 0$ be such that for every integer ℓ, at least $\epsilon\ell$ graphs among $G_1, G_2, \ldots, G_\ell$ are connected. Then the cover time of the $d_{\max}$-lazy random walk on $\mathcal{G}$ is $O(\epsilon^{-1} d_{\max}^2 n^3 \ln^2 n)$.*

The constant in the big-O is a universal constant that is independent of $\mathcal{G}$.

We point out that the strategy can be modified so that the a priori knowledge of $d_{\max}$ and n is not necessary. First, we can assume that $d_{\max} = n$. Second, we can try different estimates for n as follows. We start with the estimate $n = 10$. Then we always walk for $n^5 \ln^2 n$ steps as in the n-lazy random walk, where n is the current estimate, and after that we double our estimate of n. One can show that this strategy will provide $O(n^5 \ln^2 n)$ expected cover time of the random walk on a connected evolving graph with n vertices.

7 Conclusions

In this paper we demonstrate that the cover time of the simple random walk on dynamic graphs is significantly different from the case of static graphs. While the latter was well known to be polynomial, the former is shown here to be exponential on some evolving graphs. Moreover, we show that even if the random walk takes many steps before the graph evolves the cover time can still be exponential.

We prove that in order to accelerate the cover time one can use a *lazy* random walk and reduce the cover time to polynomial. This approach has been used previously on static graphs in order to sample nodes uniformly at random, but contrary to our situation, it can be shown that it cannot accelerate the cover time for static graphs.

To summarize, the main results presented here provide theoretical justification to the wide use of random-walk-techniques in dynamic networks. Nevertheless, one must pay careful attention to the network dynamics when choosing the implementation of the random walk.

Acknowledgements. Part of this work was done while Michal Koucký was visiting Ben Gurion University. Michal Koucký was supported in part by grant GA ČR 201/07/P276, project No. 1M0021620808 of MŠMT ČR and Institutional Research Plan No. AV0Z10190503.

References

1. Alanyali, M., Saligrama, V., Sava, O.: A random-walk model for distributed computation in energy-limited network. In: Proc. of 1st Workshop on Information Theory and its Application, San Diego (2006)
2. Aldous, D., Fill, J.: Reversible Markov Chains and Random Walks on Graphs (unpublished, 1999), http://stat-www.berkeley.edu/users/aldous/RWG/book.html
3. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: 20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, pp. 218–223. IEEE, New York (1979)
4. Alon, N., Avin, C., Koucký, M., Kozma, G., Lotker, Z., Tuttle, M.R.: Many random walks are faster than one. In: 20th ACM Symposium on Parallelism in Algorithms and Architectures (to appear, 2008)
5. Avin, C., Brito, C.: Efficient and robust query processing in dynamic environments using random walk techniques. In: Proc. of the Third International Symposium on Information Processing in Sensor Networks, pp. 277–286 (2004)
6. Avin, C., Koucký, M., Lotker, Z.: How to explore a fast-changing world. Tech. Rep. pre-116, Institute of Mathematics of the Academy of Sciences of the Czech Republic (2007), http://www.math.cas.cz/preprint/pre-116.pdf
7. Cooper, C., Frieze, A.: Crawling on simple models of web graphs. Internet Mathematics 1, 57–90 (2003)
8. Feige, U.: A tight lower bound on the cover time for random walks on graphs. Random Structures and Algorithms 6(4), 433–438 (1995)

9. Feige, U.: A tight upper bound on the cover time for random walks on graphs. Random Structures and Algorithms 6(1), 51–54 (1995)
10. Ferreira, A.: Building a reference combinatorial model for manets. Network, IEEE 18(5), 24–29 (2004)
11. Ferreira, A., Goldman, A., Monteiro, J.: On the evaluation of shortest journeys in dynamic networks. In: Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007), pp. 3–10 (2007)
12. Gkantsidis, C., Mihail, M., Saberi, A.: Random walks in peer-to-peer networks. In: INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 1, p. 130 (2004)
13. Henzinger, M., Heydon, A., Mitzenmacher, M., Najork, M.: Measuring index quality using random walks on the Web. WWW8 / Computer Networks 31(11-16), 1291–1303 (1999)
14. Jarry, A., Lotker, Z.: Connectivity in evolving graph with geometric properties. In: DIALM-POMC 2004: Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing, NY, USA, pp. 24–30. ACM Press, New York (2004)
15. Jerrum, M., Sinclair, A.: Approximating the permanent. SIAM Journal on Computing 18(6), 1149–1178 (1989)
16. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 482–491 (2003)
17. Lovász, L.: Random walks on graphs: A survey. In: Combinatorics, Paul Erdos is eighty, Keszthely, vol. 2 (1993); Bolyai Soc. Math. Stud. János Bolyai Math. Soc., Budapest, 2, 353–397 (1996)
18. Sadagopan, N., Krishnamachari, B., Helmy, A.: Active query forwarding in sensor networks (acquire). Journal of Ad Hoc Networks 3(1), 91–113 (2005)

Networks Become Navigable as Nodes Move and Forget

Augustin Chaintreau[1], Pierre Fraigniaud[2,⋆], and Emmanuelle Lebhar[2,⋆⋆]

[1] Thomson, Paris, France
Augustin.Chaintreau@thomson.net
[2] CNRS and University Paris Diderot, France
Pierre.Fraigniaud@liafa.jussieu.fr, Emmanuelle.Lebhar@liafa.jussieu.fr

Abstract. We propose a dynamic process for network evolution, aiming at explaining the emergence of the small world phenomenon, i.e., the statistical observation that any pair of individuals are linked by a short chain of acquaintances computable by a simple decentralized routing algorithm, known as greedy routing. Our model is based on the combination of two dynamics: a random walk (spatial) process, and an harmonic forgetting (temporal) process. Both processes reflect natural behaviors of the individuals, viewed as nodes in the network of inter-individual acquaintances. We prove that, in k-dimensional lattices, the combination of these two processes generates long-range links mutually independently distributed as a k-harmonic distribution. We analyze the performances of greedy routing at the stationary regime of our process, and prove that the expected number of steps for routing from any source to any target in any multidimensional lattice is a polylogarithmic function of the distance between the two nodes in the lattice. Up to our knowledge, these results are the first formal proof that navigability in small worlds can emerge from a dynamic process for network evolution. Our dynamica process can find practical applications to the design of spatial gossip and resource location protocols.

Keywords: Small world phenomenon, dynamic process, routing, spatial gossip, resource location, random walks.

1 Introduction

Models relating geography and social-network friendship enable a good understanding of the small world phenomenon, a.k.a., six degrees of separation between individuals [13,31]. In these models, the probability of befriending a particular person is assumed to be inversely proportional to the number of closer people, fitting with what was observed experimentally (cf. [30]). Under this assumption, it was proved that, using ad hoc probability distributions, many classes of

⋆ Additional supports from the ANR projects ALADDIN and ALPAGE, and from the COST Action 295 DYNAMO.

⋆⋆ Additional supports from the ANR project ALADDIN, and from the COST Action 295 DYNAMO.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 133–144, 2008.

graphs are navigable, that is, a simple decentralized routing procedure enables efficient routing from any source to any target. (By efficient, we mean, as it is standard in this framework, that routing from any source s to any target t takes a polylogarithmic expected number of steps). For instance, such a navigability property is satisfied in multi-dimensional meshes [26], in graphs of bounded ball growth [15], and more generally in graphs of bounded doubling dimension [35]. In all these cases, a graph G, that may not only represent geography but also other proximity measures like professional activities, religious beliefs, etc., is enhanced with additional links chosen at random. More precisely, every node is given some *long-range links* pointing at other nodes in the graph. For each long-range link added at a node u, the probability that the head of this link is v is inversely proportional to the size of the ball of radius $\text{dist}_G(u, v)$ centered at u in G, hence depending on the density of G around u. This setting applies to weighted graphs too [28], and to infinite graphs as well [15]. For instance, in the k-dimensional lattice $\mathbb{Z}^k$, the probability that u has a long-range link pointing at v is essentially proportional to $1/d^k$ where d is the distance between u and v in the lattice. This choice of the long-range links enables greedy routing[1] to perform in polylogarithmic expected number of steps (as a function of the distance in the lattice between the source and the target), and it was shown to be necessary.

1.1 Navigability as an Emerging Property

In [27] (Problem 7), Jon Kleinberg asks about "what kinds of growth processes or selective pressures might exist to cause networks to become more efficiently searchable". Many attempts have been made to explain how the density-based distribution of the long-range links can emerge with time from the evolution of a network. Inspired by the world wide web or by P2P file-sharing systems, all the models we are aware of have considered the augmentation process (or rewiring) of a static graph used by its nodes for searching information. Our work uses a different approach, starting from the following observations. On the one hand, individuals usually communicate with people they have met in the past. We thus assume that long-range connections are between remote people who have met once in the past. In other words, long-range links are emerging from nodes mobility, that we model by random walks in this paper. On the other hand people tend to forget some of their former acquaintance with time. This forgetting mechanism represents the well understood fact that one cannot maintain close relationships with an explosive number of people. Thus we couple the random walk process with a forgetting process, and prove that this ideal setting is sufficient to insure polylogarithmic navigability with simply one long-range connection per node.

[1] Greedy routing [26] aims at modeling the routing strategy performed by the individuals in Milgram experiment. In a graph G enhanced with long-range links, a node u handling a message of destination t selects among all its neighbors, including its long-range contact(s), the one that is the closest to the target t according to the distance in the base graph G, and forwards the message to that node.

The main contribution of this paper is therefore to show that navigability emerges from the combination of two separate processes: a spatial process and a temporal process. Our model also provides a setting for the analysis of dynamic social networks (see e.g. [7] for an example of application to spatial gossip).

1.2 Rewiring Processes

Clauset and Moore [11] proposed the following rewiring process for the multidimensional lattice, inspired by the actions of surfers on the web. While routing from a source s to a target t, if the target is not reached after τ steps, then the long-range link of s is rewired to point at the current node x. The threshold τ is set based on the distance (in the lattice) between s and t, and on the expected time of greedy routing from s to t when the k-dimensional lattice is augmented using the k-harmonic distribution [26]. The simulation results presented in [11] show that the distribution f of the link lengths converges to the k-harmonic distribution. Sandberg and Clarke [33] proposed a different rewiring process, based on Freenet feedback mechanisms [10]. This iterative process selects, at each phase, two nodes s and t uniformly at random, and constructs the greedy path $s = x_0, x_1, \ldots, x_{k-1}, x_k = t$ from s to t. For every $i \in \{0, 1, \ldots, k\}$, the long-range link of x_i is rewired with probability p, to point at t. The $k + 1$ decisions (rewiring or not) are taken mutually independently. This process is analyzed in [32]. It is proved that, under some hypotheses, the process converges. Moreover, the stationary distribution f of the link lengths can be fully characterized. In the k-dimensional lattice, it is close to the k-harmonic distribution when p is set appropriately, and simulations show that greedy routing in rings and meshes enhanced using the stationary distribution f performs as efficiently as when these networks are enhanced using the 1- and 2-harmonic distributions, respectively.

For both [11] and [33], the complete formal analysis of the process remains open (even the formal characterization of the stationary distribution of the processes described in [11] remains open). The difficulty of the analysis is due to the dependencies between the long-range links generated by the processes. In particular, the computation of the greedy routing performances is a challenge when the long-rank links are not mutually independent. So, building further theory upon these two models looks quite difficult.

In this paper, we propose a dynamic network model based on the combination of two simple processes: a random walk process, and a harmonic forgetting process.We prove that the combination of these two processes generates long-range links mutually independently distributed as a distribution that resembles the density-based distribution, and from which navigability provably emerges.

1.3 Sketch of Our Network Evolution Process

In our network evolution process, called move-and-forget, or M&F for short, individuals are modeled by tokens moving from node to node in the k-dimensional lattice $\mathbb{Z}^k$, for some fixed integer $k \geq 1$ (the dimension of the lattice may be related to the number of proximity criteria used by the individuals for routing).

Initially, each node is occupied by exactly one token. These tokens are moved mutually independently during the execution of the dynamic process, according to a random walk.

Tokens are attached to the heads of the long-range links, whose tails are the nodes from where the tokens initially started their random walks. Using the analogy of individuals moving in the geographical world, each long-range link indicates an acquaintance between an individual located at a fixed geographical point (where the token initially stood) and some individual located at some geographical coordinates (where the token currently stands).

The random walk process is coupled with another dynamic: nodes may forget their contacts through their long-range links. The motivation for our forgetting process is that individuals may loose touch with former friends, but they meet new people among which some may become friends. Since older acquaintances indicate stronger relationships, they have less probability to be forgotten than recent ones. We assume that whenever an acquaintance has been known for twice longer, she or he is twice less likely to be forgotten now. Therefore a long-range link of age a, that is a long-range link that survived a steps of the forgetting process, is forgotten with probability $\phi(a) \propto 1/a$. When a long-range link is forgotten by a node, it is rewired to point at this node (hence creating a self-loop). The token at the head of the forgotten link is removed, and a new token is launched at the node. (A new local relationship replaces an old remote relationship).

Note that M&F is defined independently from the dimension k of the lattice: tokens execute random walks, and they are forgotten with a probability that depends only of their ages.

1.4 Our Results

We prove that, for any fixed integer $k \geq 1$, the M&F rewiring process sketched above converges in the k-dimensional lattice to a stationary regime where the distribution f of the link lengths that resembles the k-harmonic distribution. Precisely, we prove that there exists $d_0 \geq 0$ and two positive constants c and c', such that, for any $\mathbf{d} = (d_1, \ldots, d_k) \in \mathbb{Z}^k$ with $|d_i| \geq d_0$ for all $i \in \{1, \ldots, k\}$, we have

$$\frac{c}{\| \mathbf{d} \|^k \cdot \ln^{1+\varepsilon} \| \mathbf{d} \|} \leq f(\mathbf{d}) \leq \frac{c' \ln^{k/2} \| \mathbf{d} \|}{\| \mathbf{d} \|^k \cdot \ln^{1+\varepsilon} \| \mathbf{d} \|}$$

where $\varepsilon > 0$ is a fixed (arbitrary small) parameter of M&F, and $\| \cdot \|$ denotes the ℓ_∞ norm.

Moreover, M&F guarantees the mutual independence of the long-range links. As a consequence, the performances of greedy routing in the lattice enhanced using the distribution f can be analyzed formally. We prove that the expected number of steps of greedy routing from any source s to any target t at distance d in the k-dimensional lattice satisfies

$$\mathbb{E}[X_{s,t}] \leq O(\ln^{2+\varepsilon} d).$$

Table 1. Properties of known network evolution processes compared to M&F

	Convergence	Navigability
A. Clauset and C. Moore (2003)	Simulations	Simulations
O. Sandberg and I. Clarke (2007)	**Proof**	Simulations
Move-and-forget (M&F)	**Proof**	**Proof**

Therefore, greedy routing performs polylogarithmically as a function of the distance between the source and the target. In particular, the performances of greedy routing are essentially the same as the ones obtained by Kleinberg [26] using the ad hoc k-harmonic distribution [26].

Up to our knowledge, these results are the first formal proof that navigability in small worlds can emerge from a dynamic process for network evolution (see Table 1). Moreover, M&F is simple (by just coupling two simple dynamics), naturally distributed (each node takes care of just its token), robust (the loss of one token simply requires to launch a new token), and scalable (by direct adaptations of the infinite lattice setting to square toroidal meshes of arbitrary sizes).

Last but not least, M&F can find practical applications, including the design of distributed spatial gossip and resource location protocols.

The omitted proofs of the results in this extended abstract can be found in [8].

1.5 Related Works

The search for a network evolution process that could explain the emergence of the small world phenomenon in social networks started with the pioneering work of Watts and Strogatz [36] who proposed a rewiring process in the cycle, generating networks possessing small diameter and large clustering coefficient, simultaneously. Adding random matchings to cycles, as in [5], yields graphs with small diameter, but non necessarily with large clustering coefficient. As far as navigability is concerned, these networks do not support efficient decentralized routing mechanisms [26]. Preferential attachment model [2,34] enables the design of efficient search procedures under specific circumstances (see [18] and the references therein), the recent lower bounds in [14] show that polylogarithmic routing cannot be achieved in general in such networks. As far as we know, the only network evolution models from which polylogarithmic routing emerges are the aforementioned ones [11,33].

Following up the seminal work of Kleinberg [26], a large literature has been dedicated to the analysis of greedy routing in graphs enhanced by long-range links set according to various kinds of probability distributions (see, e.g., [1, 15,16,17,35]). These papers proved that several large classes of graphs can be enhanced by long-range links so that greedy routing performs in polylogarithmic expected number of steps. A lower bound of $\Omega(n^{1/\sqrt{\log n}})$ expected number of steps for greedy routing in arbitrary graphs has been proved in [20], and an upper bound of $O(n^{1/3})$ has been proved in [19]. Lower bounds for the cycle can be found in [3,4,21].

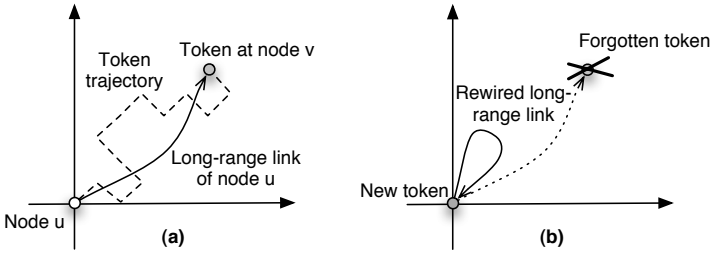


Fig. 1. Dynamic of the long-range links in M&F

2 The Move-and-Forget (M&F) Rewiring Process

2.1 Process Description

Random walks. Let $k \geq 1$ be an integer. The rewiring process move-and-forget (M&F for short) assumes that each node in the k-dimensional lattice $\mathbb{Z}^k$ is initially occupied by exactly one token. These tokens move mutually independently according to random walks. That is, each token is given a set of k fair coins c_i, $i = 1, \ldots, k$. At each step of its walk, each token flips its k coins, and moves in the ith dimension of the lattice in the positive direction if c_i is head, and in the negative direction if it is tail. More precisely, let $X(t) \in \mathbb{Z}^k$ denotes the position of a token in the lattice after t steps of M&F, assuming that the token initially started at node $(u_1, \ldots, u_k) \in \mathbb{Z}^k$. We have $X(0) = (u_1, \ldots, u_k)$, and, for $t \geq 1$, $X(t) = (X_1(t), \ldots, X_k(t))$ satisfies

$$X_i(t) = \begin{cases} X_i(t-1) + 1 \text{ with probability } 1/2; \\ X_i(t-1) - 1 \text{ with probability } 1/2. \end{cases} \quad (1)$$

Setting of the long-range links. Tokens are attached to the heads of the long-range links, whose tails are the nodes from where the tokens initially started their random walks (see Figure 1(a)). The head of a long-range link is called the long-range *contact* of the tail of this link. Hence the long-range contact of a node u is the node v currently occupied by the token launched by node u.

Forgetting process. Nodes may forget their contacts through their long-range links. More precisely, a long-range link of age $a \geq 0$, that is a long-range link that survived a steps of the forgetting process, is forgotten with probability $\phi(a)$. When a long-range link is forgotten by a node, it is rewired to point at this node (see Figure 1(b)). The token at the head of the forgotten link is removed, and a new token is launched at the node. This new token starts another random walk in $\mathbb{Z}^k$. Hence, if $A(t) \in \mathbb{N}$ denotes the age of the long-range link of some node u, that is the number of steps between time t and the last time this link was rewired during the execution of M&F, and if $C(t)$ denotes the long-range contact of node u at step t, then we have $C(t) = X(A(t))$.

The forgetting function ϕ has a huge impact on the distribution of the long-range link lengths. In this paper, we will consider $\phi(a) \propto 1/a$. The precise setting

of ϕ will appear more complex for technical reasons only[2] (series convergence for infinite lattices, normalization, etc.). In fact, its behavior essentially reflects a decreasing of the forgetting probability that is inversely proportional to the age of the relationships. The precise setting of ϕ is described in the next section which explains the connections between the random walk X, the forgetting function ϕ, and the distribution f of the long-range link lengths.

2.2 Setting of the Forgetting Function

We first prove that the age of the long-range link resulting from the execution of M&F at a node has a stationary distribution.

Lemma 1. *For any function ϕ in $[0,1)$ such that the series of general term $\Pi_{i=1}^{j}(1-\phi(i))$ is finite, $(A(t))_{t\geq 0}$ is a Markov chain which is irreducible, aperiodic, and positive recurrent, with stationary probability distribution π where*

$$\pi(a) = \frac{\Pi_{i=1}^{a}(1-\phi(i))}{\sum_{j\geq 0}\Pi_{i=1}^{j}(1-\phi(i))},$$

for all $a \geq 0$.

Definition 1. *We define the forgetting probability ϕ as the following function:*

$$\phi(a) = \begin{cases} 0 & \text{if } a = 0,\ 1,\ \text{or } 2; \\ 1 - \frac{a-1}{a}\left(\frac{\ln(a-1)}{\ln a}\right)^{1+\epsilon} & \text{if } a \geq 3; \end{cases} \tag{2}$$

where $\epsilon > 0$ is arbitrary small.

Note that $\phi(a) = \frac{1}{a} + o\left(\frac{1}{a}\right)$. Indeed,

$$\left(\frac{\ln(a-1)}{\ln a}\right)^{1+\epsilon} = \left(1 + \frac{\ln(1-1/a)}{\ln a}\right)^{1+\epsilon} = 1 - \frac{1+\epsilon}{a\ln a} + o\Big(\frac{1}{a\ln a}\Big)$$

If ϕ is defined according to Eq. (2), then Lemma 1 enables to give a close formula for π.

Lemma 2. *If ϕ is defined according to Eq. 2, then there exists a constant $c > 0$ such that $\pi(0) = \pi(1) = \pi(2) = c$ and for any $a \geq 3$,*

$$\pi(a) = \frac{c}{a\ \ln^{1+\epsilon} a}.$$

[2] For instance, one needs $\sum_{a\geq 0}\phi(a)$ to diverge since otherwise the Markov chain $A(t)$ would be transient, and links could survive infinitely with positive probability. However, on the one hand, just setting $\phi(a) = 1/a$ would make $A(t)$ recurrent null (and thus for any a we would have $\Pr\{A(t) = a\}$ converging to 0 as t goes to infinity), but, on the other hand, setting $\phi(a) = 1/a^{\alpha}$ with $\alpha < 1$ would not yield navigability.

Finally, the relationship between the stationary distribution of the long-range link ages and the stationary distribution of the long-range link lengths is made explicit in the following lemma.

Lemma 3. *The distribution of the long-range links converges to the distribution f satisfying, for any $\mathbf{d} \in \mathbb{Z}^k$,*

$$f(\mathbf{d}) = \sum_{a \geq 0} \pi(a) \cdot \Pr\{X(a) = \mathbf{d}\}.$$

3 Analysis of the Dynamic Process M&F

In this section, we analyze the stationary distribution of the long-range link lengths in the k-dimensional lattice, and prove that this distribution resembles the k-harmonic distribution.

Theorem 1. *There exist $d_0 \geq 0$ and two positive constants c and c' such that, for any $\mathbf{d} = (d_1, \ldots, d_k) \in \mathbb{Z}^k$ with $|d_i| \geq d_0$ for all $i \in \{1, \ldots, k\}$, we have*

$$\frac{c}{\| \mathbf{d} \|^k \cdot \ln^{1+\epsilon} \| \mathbf{d} \|} \leq f(\mathbf{d}) \leq \frac{c' \ln^{k/2} \| \mathbf{d} \|}{\| \mathbf{d} \|^k \cdot \ln^{1+\epsilon} \| \mathbf{d} \|}$$

where $\epsilon > 0$ is the fixed parameter of M&F, and $\| \cdot \|$ denotes the ℓ_∞ norm.

To prove the theorem, we first prove that, for large distances d, a random walk of age a cannot be of length d unless $a \geq \Omega(d^2)$. More precisely, we establish an exponentially small upper bound for the probability for a long-range link to be of length d at age $a = o(d^2)$. Second, we prove that if the age a is sufficiently large, then the chance for a random walk to reach a given distance d at age a is proportional to $\frac{1}{\sqrt{a}}$. Summing this probability over all values of a larger than d^2 allows us to conclude that the transform of the age distribution π described in Lemma 3 is approaching the k-harmonic distribution.

We compute an estimation of $\Pr\{X(a) = d\}$ when a is sufficiently large. We will use the following asymptotic equivalent of the binomial coefficient, that can be derived by application of the Stirling formula. Let n_i and m_i be two sequences of positive integers such that $n_i \to \infty$, $m_i \to \infty$, and $n_i - m_i \to \infty$ when i grows to infinity. Then, as i grows to infinity

$$\binom{n_i}{m_i} \sim \frac{1}{\sqrt{2\pi}} \cdot \sqrt{\frac{n_i}{m_i \cdot (n_i - m_i)}} \cdot \frac{n_i^{n_i}}{m_i^{m_i} \cdot (n_i - m_i)^{n_i - m_i}}. \tag{3}$$

Lemma 4. *Let X be a random walk in $\mathbb{Z}$. For any $C > 0$, $\zeta > 0$, there exists $d_0 > 1$ such that, for any $|d| \geq d_0$ and $a \geq \frac{d^2}{C \cdot \ln |d|}$, we have*

$$(1 - \zeta) \cdot \sqrt{\frac{2}{\pi \cdot a}} \exp\left(-\frac{3d^2}{4a}\right) \leq \Pr\{X(a) = d\} \leq (1 + \zeta) \cdot \sqrt{\frac{2}{\pi \cdot a}} \exp\left(-\frac{d^2}{4a}\right).$$

We are now ready to prove of the lower bound of Theorem 1. For the sake of simplicity, let us first assume that the dimension of the lattice is 1. In this case, one can apply the results from the previous section directly. For any $a \geq \frac{3}{4}d^2$ we have

$$\exp\left(-\frac{3d^2}{4a}\right) \geq 1/e\,.$$

Therefore, for any $\zeta > 0$, there exists d_0 large enough and $a \geq \frac{3}{4}d^2$ such that Lemma 4 yields:

$$\Pr\{X(a) = d\} \geq \frac{1-\zeta}{e}\sqrt{\frac{2}{\pi}}\frac{1}{\sqrt{a}}.$$

Thus :

$$f(d) \;=\; \sum_{a\geq 0}\Pr\{X(a)=d\}\,\pi(a) \;\geq\; \frac{1-\zeta}{e}\sqrt{\frac{2}{\pi}}\sum_{a\geq\frac{3}{4}d^2}\frac{1}{a^{3/2}\cdot\ln^{1+\epsilon}(a)}.$$

More generally, in the k-dimensional lattice, let us denote the position of the random walk by $X(a) = (X_1(a), \cdots, X_k(a))$. From the setting of M&F, each X_i is an unbiased random walk in dimension 1, and the X_is are mutually independent. We can thus apply all the results from the previous section independently for each coordinate of $\mathbf{d} = (d_1, \ldots, d_k)$. Assuming that

$$|d_i| \geq d_0 \text{ for all } i \in \{1, \ldots, k\},$$

we can apply Lemma 4 to every dimension. We get:

$$a \geq \frac{3}{4}\parallel \mathbf{d} \parallel^2 \Longrightarrow \forall i \in \{1,\ldots,k\}\,,\, \Pr\{X_i(a)=d_i\} \geq \frac{1-\zeta}{e}\sqrt{\frac{2}{\pi}}\frac{1}{\sqrt{a}}\exp\left(-\frac{3d_i^2}{4a}\right).$$

For $a \geq \frac{3}{4}\parallel \mathbf{d} \parallel^2$, we have,

$$\frac{3}{4}\frac{d_i^2}{a} \leq \frac{3}{4}\frac{\parallel \mathbf{d} \parallel^2}{a} \leq 1 \;\text{ and thus }\; \exp\left(-\frac{3d_i^2}{4a}\right) \geq 1/e.$$

As a consequence, by Lemma 4,

$$\Pr\{X(a) = \mathbf{d}\} = \Pr\{X_1(a) = d_1, \ldots, X_k(a) = d_k\} \geq \left(\frac{1-\zeta}{e}\sqrt{\frac{2}{\pi}}\frac{1}{\sqrt{a}}\right)^k$$

hence

$$f(\mathbf{d}) = \sum_{a\geq 0}\Pr\{X(a)=\mathbf{d}\}\,\pi_A(a) \geq \left(\frac{1-\zeta}{e}\sqrt{\frac{2}{\pi}}\right)^k \sum_{a\geq\frac{3}{4}\|\mathbf{d}\|^2}\frac{c}{a^{1+(k/2)}\cdot\ln^{1+\epsilon}(a)}.$$

The lower bound is then a direct consequence of the following result with $N = \frac{3\|\mathbf{d}\|^2}{4}$.

Lemma 5. *For any $\epsilon > 0$, and any $N \geq e^{2(1+\epsilon)}$, we have*

$$\frac{2/(k+1)}{N^{k/2}\ln^{1+\epsilon} N} \leq \sum_{a \geq N} \frac{1}{a^{1+(k/2)}\ln^{1+\epsilon} a} \leq \frac{2/k}{(N-1)^{k/2}\ln^{1+\epsilon}(N-1)}. \quad (4)$$

Due to lack of space, the proof of the upper bound of Theorem 1 is omitted. It uses similar arguments for large values of a and uses a Chernoff bound for small values of a, given by the following lemma.

Lemma 6. *Let X be a random walk in $\mathbb{Z}$. Then, for any age $a > 0$ and any distance $d \in \mathbb{Z}$, we have* $\Pr\{X(a) = d\} \leq 2 \cdot \exp\left(-\frac{d^2}{32 \cdot a}\right)$.

4 Applications

In the previous section, we have shown that the distribution f of the long-range link lengths is provably converging to a distribution that resembles the k-harmonic distribution. In this section, we show that greedy routing can be formally analyzed at the stationary state of this distribution. Greedy routing can be formally analyzed for two reasons: (1) The distribution f of the long-range links constructed by M&F can be bounded formally (cf. Theorem 1); (2) The long-range links resulting from M&F are mutually independent. Based on these two facts, we can establish the theorem below.

Theorem 2. *In the k-dimensional lattice augmented with the long-range links at the stationary distribution of the dynamic process* M&F*, the expected number of steps of greedy routing from any source node s to any target node t at distance d is $O(\ln^{2+\epsilon} d)$.*

In the rest of the section, we discuss how M&F can find practical applications to the design of spatial gossip and resource location protocols.

Gossip-based protocols, a.k.a., epidemic algorithms [12], have been introduced as a methodology for designing robust and scalable communication schemes in distributed systems. Roughly, in each step, each node u chooses some other node v, and sends a message to it. By applying such scheme at each node, an information originated at some source s will eventually reach its target(s). This methodology can be adapted to various problems, including information spreading, resource location, etc. In [24], Kempe et al. introduced *spatial gossip*, which allowed them to derive efficient solutions for many communication problems. In spatial gossip, nodes are arranged with uniform density in the k-dimensional Euclidean space, and, at each step of the gossip protocol, node u chooses node v with probability $\propto 1/d^{\varrho k}$ where $\varrho > 0$ is a fixed parameter, and d is the distance between u and v. In particular, it is shown that, for $\varrho \in (1, 2)$, spatial gossip enables to propagate information at distance d in time polylogarithmic in d. In [25], Kempe and Kleinberg showed that spatial gossip enables to solve larger classes of problems, including MST construction and permutation routing. In particular, they prove that permutation routing using spatial gossip with $\varrho = 1$ performs in polylogarithmic expected number of steps.

In [7], we show how the M&F process could facilitate the implementation of the protocols in [24,25] for networks that take advantage of node mobility, as in, e.g., [9,22,23].

References

1. Abraham, I., Gavoille, C.: Object Location Using Path Separators. In: 25th ACM Symp. on Principles of Distributed Computing (PODC), pp. 188–197 (2006)
2. Albert, R., Barabási, A.-L.: Statistical mechanics of complex networks. Review of modern physics 74, 47–97 (2002)
3. Aspnes, J., Diamadi, Z., Shah, G.: Fault-tolerant routing in peer-to-peer systems. In: 21st ACM Symp. on Principles of Distributed Computing (PODC), pp. 223–232 (2002)
4. Barrière, L., Fraigniaud, P., Kranakis, E., Krizanc, D.: Efficient Routing in Networks with Long Range Contacts. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 270–284. Springer, Heidelberg (2001)
5. Bollobás, B., Chung, F.: The diameter of a cycle plus a random matching. SIAM J. Discrete Math. 1(3), 328–333 (1988)
6. Bremaud, P.: Markov Chains, Gibbs Field, Monte Carlo Simulation and Queues. Springer, Heidelberg (1999)
7. Chaintreau, A., Fraigniaud, P., Lebhar, E.: Opportunistic spatial gossip over mobile social networks. In: 1st ACM SIGCOMM Workshop on Online Social Net (WOSN) (to appear, 2008)
8. Chaintreau, A., Fraigniaud, P., Lebhar, E.: Networks Become Navigable as Nodes Move and Forget. Technical Report, arXiv:0803.0248v1 (2008)
9. Chaintreau, A., Hui, P., Crowcroft, J., Diot, C., Scott, J., Gass, R.: Impact of Human Mobility on Opportunistic Forwarding Algorithms. IEEE Trans. Mob. Comp. 6(6), 606–620 (2007)
10. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 46–66. Springer, Heidelberg (2001)
11. Clauset, A., Moore, C.: How Do Networks Become Navigable? Technical Report, arXiv:0309.415v2 (2003)
12. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic Algorithms for Replicated Database Maintenance. Operating Systems Review 22(1), 8–32 (1988)
13. Dodds, P., Muhamad, R., Watts, D.: An experimental study of search in global social networks. Science 301(5634), 827–829 (2003)
14. Duchon, P., Eggeman, N., Hanusse, N.: Non-Searchability of Random Power Law Graphs. In: Tovar, E., Tsigas, P., Fouchal, H. (eds.) OPODIS 2007. LNCS, vol. 4878. Springer, Heidelberg (2007)
15. Duchon, P., Hanusse, N., Lebhar, E., Schabanel, N.: Could any graph be turned into a small-world? Theoretical Computer Science 355(1), 96–103 (2006)
16. Flammini, M., Moscardelli, L., Navarra, A., Perennes, S.: Asymptotically Optimal Solutions for Small World Graphs. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 414–428. Springer, Heidelberg (2005)
17. Fraigniaud, P.: Greedy routing in tree-decomposed graphs: a new perspective on the small-world phenomenon. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 791–802. Springer, Heidelberg (2005)

18. Fraigniaud, P., Gauron, P., Latapy, M.: Combining the Use of Clustering and Scale-Free Nature of User Exchanges into a Simple and Efficient P2P System. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1163–1172. Springer, Heidelberg (2005)
19. Fraigniaud, P., Gavoille, C., Kosowski, A., Lebhar, E., Lotker, Z.: Universal Augmentation Schemes for Network Navigability: Overcoming the $\sqrt{n}$-Barrier. In: 19th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 1–9 (2007)
20. Fraigniaud, P., Lebhar, E., Lotker, Z.: A Doubling Dimension Threshold $\Theta(\log \log n)$ for Augmented Graph Navigability. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 376–386. Springer, Heidelberg (2006)
21. Giakkoupis, G., Hadzilacos, V.: On the complexity of greedy routing in ring-based peer-to-peer networks. In: 26th ACM Symp. on Princ. of Dist. Comp. (PODC) (2007)
22. Grossglauser, M., Tse, D.: Mobility Increases the Capacity of Ad Hoc Wireless Networks. IEEE/ACM Trans. on Net. 10(4), 477–486 (2002)
23. Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: Proc. ACM SIGCOMM (2004)
24. Kempe, D., Kleinberg, J., Demers, A.: Spatial gossip and resource location protocols. In: 33rd ACM Symposium on Theory of Computing, pp. 163–172 (2001)
25. Kempe, D., Kleinberg, J.: Protocols and impossibility results for gossip-based communication mechanisms. In: 43st IEEE Symp. on Foundations of Computer Science, pp. 471–480 (2002)
26. Kleinberg, J.: The small-world phenomenon: an algorithmic perspective. In: 32nd ACM Symp. on Theory of Computing (STOC), pp. 163–170 (2000)
27. Kleinberg, J.: Complex networks and decentralized search algorithm. In: International Congress of Mathematicians (ICM), Madrid (2006)
28. Kumar, R., Liben-Nowell, D., Tomkins, A.: Navigating Low-Dimensional and Hierarchical Population Networks. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168. Springer, Heidelberg (2006)
29. Liben-Nowell, D., Kleinberg, J.: The link prediction problem for social networks. Journal of the American society for information science and technology 58(7), 1019–1031 (2007)
30. Liben-Nowell, D., Novak, J., Kumar, R., Raghavan, P., Tomkins, A.: Geographic routing in social networks. In: Proc. of the Natl. Academy of Sciences of the USA, vol. 102/3, pp. 11623–11628 (2005)
31. Milgram, S.: The Small-World Problem. Psychology Today, pp. 60–67 (1967)
32. Sandberg, O.: Neighbor Selection and Hitting Probability in Small-World Graphs. Annals of Applied Probability (to appear, 2008)
33. Sandberg, O., Clarke, I.: The evolution of navigable small-world networks. Tech. Report 2007:14, Chalmers University of Technology (2007)
34. Simon, H.: On a class of skew distribution functions. Biometrika 42(3/4), 425–440 (1955)
35. Slivkins, A.: Distance estimation and object location via rings of neighbors. In: 24th Annual ACM Symp. on Princ. of Dist. Comp. (PODC), pp. 41–50 (2005)
36. Watts, D., Strogatz, S.: Collective Dynamics of Small-World Networks. Nature 393, 440–442 (1998)

Fast Distributed Computation of Cuts Via Random Circulations

David Pritchard*

Department of Combinatorics and Optimization, University of Waterloo
dagpritchard@math.uwaterloo.ca

Abstract. We describe a new *circulation*-based method to determine cuts in an undirected graph. A circulation is an oriented labeling of edges with integers so that at each vertex, the sum of the in-labels equals the sum of out-labels. For an integer k, our approach is based on simple algorithms for sampling a circulation (mod k) uniformly at random. We prove that with high probability, certain dependencies in the random circulation correspond to cuts in the graph. This leads to simple new linear-time sequential algorithms for finding all cut edges and *cut pairs* (a set of 2 edges that form a cut) of a graph, and hence 2-edge-connected and 3-edge-connected components.

In the model of *distributed computing* in a graph $G = (V, E)$ with $O(\log |V|)$-bit messages, our approach yields faster algorithms for several problems. The diameter of G is denoted by $\mathcal{D}$. Previously, Thurimella [J. Algorithms, 1997] gave a $O(\mathcal{D}+\sqrt{|V|}\log^* |V|)$-time algorithm to identify all cut vertices, 2-edge-connected components, and cut edges, and Tsin [Int. J. Found. Comput. Sci., 2006] gave a $O(|V| + \mathcal{D}^2)$-time algorithm to identify all cut pairs and 3-edge-connected components.

We obtain simple $O(\mathcal{D})$-time distributed algorithms to find all cut edges, 2-edge-connected components, and cut pairs, matching or improving previous time bounds on all graphs. Under certain assumptions these new algorithms are *universally optimal*, due to a $\Omega(\mathcal{D})$-time lower bound on every graph. These results yield the first distributed algorithms with *sub-linear* time for cut pairs and 3-edge-connected components. Let Δ denote the maximum degree. We obtain a $O(\mathcal{D} + \Delta/\log |V|)$-time distributed algorithm for finding cut vertices; this is faster than Thurimella's algorithm on all graphs with $\Delta, \mathcal{D} = O(\sqrt{|V|})$. The basic distributed algorithms are Monte Carlo, but can be made Las Vegas without increasing the asymptotic complexity.

1 Introduction

Let $G = (V, E)$ be a connected undirected graph. A part of G is said to be a *cut* if, after deleting it from G, the remaining graph is disconnected. Define a *cut*

* Funded in part by a NSERC PGS-D scholarship. The author wishes to thank Ramakrishna Thurimella, Graeme Kemkes, Carlos Hoppen, Jochen Könemann and Glencora Borradaile for comments on earlier versions of the paper.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 145–160, 2008.

vertex to be a vertex v such that $\{v\}$ is a cut; define a *cut edge* to be an edge e such that $\{e\}$ is a cut (i.e., a bridge); and define a *cut pair* to be a cut consisting of two edges e, f such that neither e nor f is a cut edge. For brevity we call all of these objects *small cuts*. In a network the small cuts are relevant because they represent the critical points where local failures can cause global disruption. Our primary motivation is to efficiently find all small cuts of an undirected graph; we consider the sequential, distributed, and parallel models of computation.

The fundamentally new idea in this paper is to use *random circulations* to find small cuts. Informally, in a circulation we transport quantities of a commodity along the edges of a graph, so that the net accumulation at each vertex is zero. When the shipment quantities are taken modulo some integer k, there are only finitely many possible circulations, and our first contribution is the observation that it is easy to sample *uniformly* from the family of all circulations on a fixed graph.

For $S \subset V$, let $\delta(S)$ denote the edges with exactly one end in S. An *induced edge cut* is a set of the form $\delta(S)$ for some S; we observe that cut edges and cut pairs are induced edge cuts. A well-known principle behind our method, made precise in Proposition 1, is that the net flow of any circulation across any induced edge cut is 0. At a high level, our algorithms depend on the near-converse: for certain edge sets F that are *not* induced edge cuts, the net flow of a uniformly random circulation on F is uniformly random, hence nonzero with high probability.

The Distributed Model. Our approach improves known time bounds in the *distributed* computing model with *congestion*. This model, denoted $\mathcal{CONGEST}$ (e.g. by Peleg [1, §2.3]), works as follows. The computation takes place in the graph $G = (V, E)$ where each vertex is a computer and each edge is a bidirectional communication link; i.e., we study the problem of having a network compute the small cuts of its own topology. There is no globally shared memory, only local memory at each vertex. Initially only local topology is known: each vertex knows its ID value, which is unique, and its neighbours' IDs. Time elapses in discrete *rounds*. In each round, every vertex performs local computations and may send one message to each of its neighbors, to be received at the start of the next round. The *time complexity* of a distributed algorithm is the number of rounds that elapse, and the *message complexity* is the total number of messages that are sent.

In the $\mathcal{CONGEST}$ model, every message must be at most $O(\log V)$ bits long. The model does not bound the memory capacity or computational power of the vertices, although our algorithms use time and space polynomial in $|V|$ at each vertex. Let $\mathcal{D}$ denote the diameter of (V, E), i.e. $\mathcal{D} := \max_{u,v \in V} \text{dist}_G(u, v)$. The message size bound, in addition to making the algorithms more practical, affects what is possible in the model, as the following example from Lotker, Patt-Shamir & Peleg [2] shows. On the one hand, if messages are allowed to be arbitrarily long, any graph property whatsoever can be trivially computed in $\mathcal{D}$ time[1]. On

[1] In $\mathcal{D}$ rounds each vertex broadcasts its local topology to all other vertices, then each vertex deduces the global topology and solves the problem with a local computation.

the other hand, Lotker et al. gave a family of graphs with $\mathcal{D} = 3$, such that in $\mathcal{CONGEST}$ on this family, a $\Omega(\sqrt[4]{|V|}/\sqrt{\log |V|})$-time lower bound holds to find the minimum spanning tree (MST).

Determining whether a task in this model can be accomplished in $O(\mathcal{D}) + o(|V|)$ time, or better yet $O(\mathcal{D})$ time, is a fundamental problem. For finding all cut edges and cut pairs of a graph, we give new affirmative answers by providing $O(\mathcal{D})$-time algorithms.

1.1 Existing Results

Our results apply to three common models of computation: sequential, distributed, and parallel. Abusing notation for readability, we sometimes abbreviate $|V|$ to V and $|E|$ to E.

Sequential. In the usual sequential (RAM) model of computing, Tarjan in the 1970s was the first to obtain linear-time ($O(V + E)$-time) algorithms to find all cut vertices [3], cut edges [3], and cut vertex-pairs (cuts $C \subset V$ with $|C| = 2$) [4]. These algorithms are based on depth-first search (DFS). Galil & Italiano, in 1991, gave the first linear-time algorithm to compute all cut pairs, by reducing to the cut vertex-pair problem.

Distributed. Here we only mention results valid in $\mathcal{CONGEST}$, ignoring results with $\Omega(n)$ message size such as one of Chang [5]. **Cut Edges/Vertices.** Two early distributed algorithms for cut edges and vertices, by Ahuja & Zhu [6] and Hohberg [7], use DFS. The smallest time complexity of any known distributed DFS algorithm is $\Theta(V)$; as such, the algorithms of Ahuja & Zhu and Hohberg have $\Omega(V)$ time complexity. Huang [8] gave a non-DFS-based algorithm with $\Theta(V)$ time complexity. A breakthrough by Thurimella [9] was an algorithm that is asymptotically faster than $\Theta(V)$ on some graphs (a so-called *sub-linear* algorithm). Precisely, Thurimella obtained time complexity[2] $O(\mathcal{D} + \sqrt{V} \log^* V)$ for cut edges and cut vertices, using a sub-linear MST subroutine. **Cut Pairs.** For cut pairs, Jennings and Motyckova [10] gave a distributed algorithm with worst-case time and message complexity $\Theta(n^3)$, and Tsin [11] recently obtained a DFS-based algorithm with improved time complexity $O(\mathcal{D}^2 + V)$.

Distributed Optimality. Distributed $\Theta(V)$-time algorithms for cut edges are optimal (up to a constant factor) on some graphs: e.g. it is straightforward to see, even guaranteed that G is either a $|V|$-cycle or a $|V|$-path, not all edges can determine if they are cut edges in less than $|V|/2 - 2$ rounds. One term for this property is *existentially optimal*, due to Garay, Kutten and Peleg [12]. However, as Thurimella's algorithm [9] showed, there are some graphs on which $\Theta(V)$ time is not asymptotically optimal. The stronger term *universally optimal* [12] applies to algorithms which, on *every* graph, have running time within a constant factor of the minimum possible.

[2] $\log^* x$ is the number of times which log must be iteratively applied to x before obtaining a number less than 1.

Parallel. In the PRAM model, optimal $O(\log V)$-time and $O(V+E)$-work Las Vegas algorithms have been given by Tarjan & Vishkin [13] for cut edges and cut vertices, and Fussell, Ramachandran & Thurimella [14] (using the reduction of Galil & Italiano [15]) for cut pairs. These algorithms require optimal spanning forest subroutines of Halperin & Zwick [16].

1.2 Our Contributions

Since our algorithms are randomized, we differentiate between two types of algorithms: *Monte Carlo* ones have deterministically bounded running time but may be incorrect with probability $1/V$ and *Las Vegas* ones are always correct and have bounded *expected* running time[3]. (Note, a Las Vegas algorithm can always be converted to Monte Carlo, so Las Vegas is generally better).

Sequential. The random circulation approach yields *new linear-time algorithms to compute all cut edges and cut pairs* of the Las Vegas type. As far as we are aware, our linear-time cut pair algorithm is the first one that does not rely on either DFS (e.g., see references in Tsin [17]) or open ear decomposition (e.g., see references in Fussell et al. [14]).

Distributed. We remark that all existing distributed algorithms mentioned for finding small cuts are deterministic. The random circulation approach yields *faster distributed algorithms for small cuts* of the Las Vegas type. For cut edges and pairs, we obtain $O(\mathcal{D})$-time algorithms. Compared to the previous best time of $O(\mathcal{D}+\sqrt{V}\log^* V)$ for cut edges, we remove the dependence on $|V|$. Compared to the previous best time of $O(\mathcal{D}^2+V)$ for cut pairs, we obtain a quadratic speedup on every graph. For cut vertices, we obtain a $O(\mathcal{D}+\Delta/\log V)$-time algorithm where Δ is the maximum degree. Compared to the previous best time of $O(\mathcal{D}+\sqrt{V}\log^* V)$ for cut vertices, this is faster on graphs with $\Delta, \mathcal{D} = O(\sqrt{V})$. We also obtain the first sub-linear distributed algorithm for 3-edge-connected components, using a connected components subroutine of Thurimella [9]. In Table 1 we depict our main results and earlier work, showing both time and message complexity.

Universal Optimality. If we assume distributed algorithms must act globally in a natural sense — either by initiating at a single vertex, or by reporting termination — then a $\Omega(\mathcal{D})$-time lower bound holds for the problems of finding cut edges or cut pairs, on any graph. Hence under natural conditions, our $O(\mathcal{D})$-time algorithms for cut edges and cut pairs are universally optimal.

Parallel. In the PRAM model, we obtain new optimal $O(\log V)$-time and $O(V+E)$-work Las Vegas algorithms for finding cut pairs and cut edges. Our algorithms require spanning forest subroutines of Halperin & Zwick [16]. These results are deferred to the full version of the paper.

[3] More generally, our algorithms can obtain error probability $\leq 1/V^c$ for any constant c without changing the asymptotic complexity.

Table 1. Comparison of our three main distributed results (denoted by †) to the best previously known algorithms

		Cuts Found	Time	Messages
[6]	'89	Vertices & Edges	$O(V)$	$O(E)$
[9]	'95	Vertices & Edges	$O(\mathcal{D} + \sqrt{V}\log^* V)$	$O(E \cdot (\mathcal{D} + \sqrt{V}\log^* V))$
[11]	'06	Pairs	$O(V + \mathcal{D}^2)$	$O(E + V \cdot \mathcal{D})$
Theorem 7†		Edges	$O(\mathcal{D})$	$O(E)$
Theorem 10†		Pairs	$O(\mathcal{D})$	$O(\min\{V^2, E \cdot \mathcal{D}\})$
Theorem 8†		Vertices	$O(\mathcal{D} + \Delta/\log V)$	$O(E(1 + \Delta/\log V))$

1.3 Other Related Work

Circulations have applications in diverse fields. Hoffman's Circulation Theorem [18] is a min-max relation for circulations from which many other min-max relations can be derived. In planar graphs, *nowhere-zero* circulations modulo k correspond to vertex k-colourings of the dual graph (e.g. see [19] and its references). Circulations also appear in the most efficient flow algorithm for planar directed graphs, due to Borradaile & Klein [20]. We defer further discussion to the full version.

Our usage of uniformly random circulations appears to be novel. The most similar work is a method of Benjamini & Lovász [21] to compute the genus of an embedded graph G while "observing" part of it. The similarity is that they use random perturbation and balancing steps to compute a "near-circulation" on G and the *dual graph* of G. Their computational model is significantly different, e.g. they allow a face to modify the values of all its incident edges in a single time step.

1.4 Organization of the Paper

Section 2 contains definitions and basic results pertaining to circulations. In Section 3 we define random circulations and show how to construct them efficiently. In Section 4 we show how random circulations yield algorithms for small cuts, and give sequential implementations. In Section 5 we precisely define our distributed model and give our distributed results. In Section 5.2 we introduce a useful technique, *fundamental cycle-cast*, which may be of independent interest. For space reasons we defer the following to the full version of the paper: $\Omega(\mathcal{D})$-time lower bounds assuming a single initiator or termination confirmation; some details of the distributed implementation; k-connected-components (e.g. distributed 3-edge-connected components in $O(\mathcal{D} + \sqrt{V}\log^* V)$ time); and Las Vegas implementation (we only discuss Monte Carlo versions here). See the preprint version `http://arXiv.org/abs/cs/0702113` for some of these details.

2 Preliminaries

In this paper the set notation $\{u, v\}$ denotes an undirected edge of $G = (V, E)$, which can be oriented in two ways, denoted (u, v) and (v, u). For a set F of edges

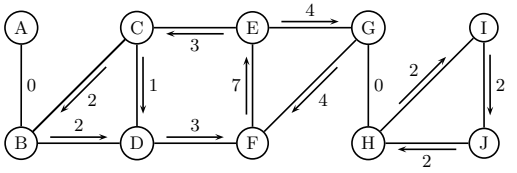


Fig. 1. A circulation ϕ. We have labeled edges in the direction of positive flow. Note that the net flow at each vertex is 0, i.e. conservation holds.

let $\overrightarrow{F}$ denote the $2|F|$ orientations of edges in F. An *F-orientation* is a subset of $\overrightarrow{F}$ consisting of exactly 1 orientation of each $e \in F$. Let $\mathbb{Z}_k$ denote the integers modulo k. For $v \in V$ the notation $\Gamma(v)$ denotes the set of neighbours of v.

Definition 1. *A* circulation on G *is a function* $\phi : \overrightarrow{E} \to \mathbb{R}$ *with the following two properties.*

Antisymmetry: $\phi(u,v) = -\phi(v,u)$ *for all* $\{u,v\} \in E$.
Conservation: $\sum_{v \in \Gamma(u)} \phi(u,v) = 0$ *holds for all vertices* u.

A k-circulation *is a function* $\phi : \overrightarrow{E} \to \mathbb{Z}_k$ *that satisfies* **Antisymmetry** *and* **Conservation** *when equality is replaced by equivalence modulo* k.

We illustrate a circulation in Figure 1. Although our notation is chosen for brevity, there are alternatives, e.g. we could equivalently define, for an E-orientation E', a circulation as a function $\phi : E' \to \mathbb{R}$ so that for each $u \in V$, $\sum_{v:(v,u) \in E'} \phi(v,u) = \sum_{v:(u,v) \in E'} \phi(u,v)$ is satisfied.

For $U \subsetneq V$, the *induced directed edge cut* $\delta^+(U)$ is the set of directed edges (u,v) with $u \in U, v \notin U$. The following result is folklore but see, e.g., [19, p. 7] for a proof.

Proposition 1 (Circulations vanish across induced cuts). *Let* ϕ *be a* k*-circulation on* G*, and let* $U \subset V$*. Then*

$$\sum_{(u,v) \in \delta^+(U)} \phi(u,v) = 0.$$

We immediately obtain the following corollary (see also [19, p. 8]).

Corollary 1. *If* $\{u,v\}$ *is a cut edge of* G *and* ϕ *is a circulation on* G *then* $\phi(u,v) = \phi(v,u) = 0$.

Proof. Let U be the connected component of $G \backslash \{\{u,v\}\}$ containing u. Now apply Proposition 1; the only member of $\delta^+(U)$ is (u,v) and so we obtain $\phi(u,v) = 0$. By antisymmetry, $\phi(v,u) = 0$. □

We now explain a tool which allows one to construct circulations; it appears implicitly in the book of Bondy & Murty [22, Ex. 12.1.1]. The idea is that for any spanning tree, we can choose any circulation values on the non-tree edges, and then there is a unique extension to a circulation on the whole graph.

Proposition 2. *Let T be any spanning tree of G and let $\phi_0 : \overrightarrow{E \backslash E(T)} \to \mathbb{Z}_k$ be antisymmetric. There is a unique circulation ϕ on G such that $\phi(u,v) = \phi_0(u,v)$ for all $\{u,v\} \in E \backslash E(T)$.*

Proof (Sketch). For a leaf node v incident to $\{u,v\} \in E(T)$, the value of $\phi(u,v)$ must equal $-\sum_t \phi_0(t,v)$ to satisfy conservation at v. The idea is to then delete $\{u,v\}$ from T and repeat. We give pseudocode in Algorithm 1 but the formal proof is deferred to the full version. □

Algorithm 1. Input: tree T, antisymmetric ϕ_0 on $E \backslash E(T)$. Output: circulation ϕ extending ϕ_0.

1: Initialize $\phi := \phi_0, S := T$. ▷ S is the subtree of T where ϕ is not yet defined
2: Root T at an arbitrary vertex r.
3: **while** S has any edges **do**
4: Let v be any leaf of S with $v \neq r$ and let u be the unique neighbor of v in S.
5: Define $\phi(v,u) := -\sum_{w \in \Gamma(v) \backslash \{u\}} \phi(v,w)$. ▷ Satisfy conservation at v
6: Define $\phi(u,v) := -\phi(v,u)$. ▷ Satisfy antisymmetry
7: Delete $\{u,v\}$ from S.
8: Output ϕ.

3 Random Circulations

We begin this section by showing that it is easy to uniformly sample from the set of all k-circulations. The basic idea is to feed a "random" ϕ_0 in to Algorithm 1. More precisely, we pick the values of ϕ_0 randomly and independently (up to antisymmetry) from $\mathbb{Z}_k$. We denote this algorithm by Rand-k-Circ(T) and illustrate it in Algorithm 2.

Algorithm 2. Input: a connected graph G. Output: the cut edges of G.

1: **procedure** Rand-k-Circ(T)
2: **for** each edge $\{u,v\} \in E \backslash E(T)$ **do**
3: Pick $x \in \mathbb{Z}_k$ uniformly and independently at random and set $\phi_0(u,v) = x, \phi_0(v,u) = -x$.
4: Return the unique circulation that extends ϕ_0 by calling Algorithm 1.

Theorem 1. *Let ϕ^* be a circulation on G and T be a spanning tree of G. Let ϕ be the output of* Rand-k-Circ(T)*. Then*

$$\Pr[\phi = \phi^*] = 1/k^{|E|-|V|+1}, \tag{1}$$

and the distribution produced by Rand-k-Circ(T) *over all k-circulations is uniform.*

Proof. The tree T has $|V|-1$ edges, so $|E\backslash E(T)| = |E|-|V|+1$. Clearly, the probability that ϕ_0 agrees with ϕ^* on $E\backslash E(T)$ is exactly $1/k^{|E|-|V|+1}$. But furthermore, by Proposition 2, $\phi = \phi^*$ if and only if ϕ_0 and ϕ^* agree on $E\backslash E(T)$, hence we obtain Equation (1). Since $1/k^{|E|-|V|+1}$ does not depend on ϕ^*, uniformity follows. □

Note, Theorem 1 implies that different choices of T have no effect on the output of Rand-k-Circ(T). Because of this we later refer to a "random k-circulation," meaning to sample a k-circulation uniformly at random by calling Rand-k-Circ with any spanning tree. We will make repeated use of the following corollary to show that ϕ "behaves randomly" on certain edge sets.

Corollary 2. *Let $D \subset E$ be such that $G\backslash D$ is connected, and D' be a D-orientation. Let ϕ be a random k-circulation. The values of ϕ on D' are uniformly and independently distributed over $\mathbb{Z}_k$.*

Proof. Since $G\backslash D$ is connected, it contains a spanning tree T of G. By Theorem 1, the distribution of ϕ is the same as if ϕ were generated by running Rand-k-Circ with this choice of T. When calling Rand-k-Circ on this T, each $e \in E\backslash E(T) \supset D$ incurs a uniform, independent sample $x \in \mathbb{Z}_k$ on line 3. (Notice that if x is uniformly distributed over $\mathbb{Z}_k$, so is $-x$). The result then follows. □

Note that Corollary 2 holds regardless of what spanning tree was actually used to generate ϕ.

4 Sequential Algorithms

In this section we show how to use random circulations to probabilistically determine the cut edges, cut pairs, and cut vertices of a graph. These are the Monte Carlo versions of the algorithms.

4.1 Finding All Cut Edges

Proposition 3. *Let $\{u,v\} \in E$ and ϕ be a random k-circulation on G. Then $\Pr[\phi(u,v) = 0]$ is 1 if $\{u,v\}$ is a cut edge and $1/k$ otherwise.*

Proof. If $\{u,v\}$ is a cut edge then Corollary 1 applies. Otherwise by Corollary 2 the value $\phi(u,v)$ is a uniformly random element of $\mathbb{Z}_k$, since $G\backslash\{\{u,v\}\}$ is connected. □

Thus, provided we pick k large enough, it is likely that the cut edges are exactly $\{\{u,v\} \mid \phi(u,v) = 0\}$. We provide pseudocode in Algorithm 3 and prove its correctness.

Theorem 2. *Algorithm 3 correctly determines the cut edges with probability $1-1/V$ and can be implemented in $O(E)$ sequential time.*

Proof. The algorithm chooses $k = |V||E|$. A union bound, in conjunction with Proposition 3, shows that the probability of error is at most $|E|/k = 1/|V|$. As is standard, we assume the machine word size is $\Omega(\log V)$. The subroutine

Algorithm 3. Input: a connected graph G. Output: the cut edges of G.

1: Let $k = |V||E|$ and let ϕ be a random k-circulation on G.
2: Output all edges $\{u, v\}$ for which $\phi(u, v) = 0$.

RAND-k-CIRC performs $O(E)$ random choices and arithmetic operations, each of which take $O(1)$ time since k is $O(\log V)$ bits long. □

4.2 Finding All Cut Pairs and Cut Classes

For cut pairs and cut vertices we work with circulations only modulo $k = 2$. This is convenient because $x = -x$ for all $x \in \mathbb{Z}_2$, and hence we can unambiguously refer to $\phi(e)$ for an edge e without specifying an orientation. The *cycle space* of an undirected graph is the family of subsets of E with even degree at each vertex, see e.g. Bondy & Murty [22, §12.1]. We remark that 2-circulations are the same as characteristic vectors of members of the cycle space.

Proposition 4, whose proof we omit, leads to our approach for finding cut pairs.

Proposition 4 (Cut pairs are induced). *If $\{e, f\}$ is a cut pair then $\{e, f\} = \delta(U)$ for some $U \subset V$.*

Proposition 5. *Let e, f be two distinct edges that are not cut edges. If ϕ is a random 2-circulation on G, then $\Pr[\phi(e) = \phi(f)] = 1$ if $\{e, f\}$ is a cut pair, and $1/2$ otherwise.*

Proof. If these two edges form a cut pair, using Proposition 4 and Proposition 1, we know that $\phi(e) + \phi(f) \equiv 0 \pmod 2$ and so $\phi(e) = \phi(f)$. Now suppose otherwise, that $\{e, f\}$ is not a cut pair. Then $G \backslash \{e, f\}$ is connected, and by Corollary 2 the values of ϕ on e and f are independent and uniform over $\mathbb{Z}_2$ whence $\Pr[\phi(e) = \phi(f)] = 1/2$. □

Proposition 5 gives us a probabilistic proof of the following fact.

Corollary 3 (Transitivity of cut pairs). *If $\{e, f\}$ and $\{f, g\}$ are cut pairs, then so is $\{e, g\}$.*

Proof. Note that e, f, g are not cut edges. Let ϕ be a random 2-circulation on G. By Proposition 5, $\phi(e) = \phi(f)$ and $\phi(f) = \phi(g)$. So $\phi(e) = \phi(g)$ with probability 1. By Proposition 5, $\{e, g\}$ must be a cut pair. □

Definition 2. *A* cut class *is an inclusion-maximal subset K of E such that $|K| > 1$ and every pair $\{e, f\} \subseteq K$ is a cut pair.*

Corollary 3 implies that any two distinct cut classes are disjoint. Hence, even though there may be many cut pairs, we can describe them all compactly — e.g. in $O(E)$ space in the sequential model — by listing all cut classes of the graph.

Let $\mathbb{Z}_2^b$ denote the set of b-bit binary strings. For $\phi : E \to \mathbb{Z}_2^b$, let $\phi_i(e)$ denote the ith bit of $\phi(e)$.

Definition 3. *A* b-bit circulation *is obtained by concatenating* b 2*-circulations* $\{\phi_i\}_{i=1}^b$. *A* (uniformly) random b-bit circulation *is obtained by concatenating* b *independent uniformly random* 2*-circulations* $\{\phi_i\}_{i=1}^b$.

Let $\oplus$ denote the bitwise xor operation. Notice that $\phi : E \to \mathbb{Z}_2^b$ is a b-bit circulation if and only if $\bigoplus_{e \in \delta(u)} \phi(e) = \mathbf{0}$ holds for each $u \in V$. The results of Sections 2 and 3 apply to b-bit circulations; for example, we can obtain Rand-b-Bit-Circ, a modified version of Rand-k-Circ that generates a uniformly random b-bit circulation, by replacing $\sum$ in Line 5 of Algorithm 1 by $\bigoplus$, replacing $\mathbb{Z}_k$ in Line 3 of Algorithm 2 by $\mathbb{Z}_2^b$, and ignoring occurrences of the unary $-$ operator. Propositions 3 and 5 give probability bounds of $1/2^b$ in place of $1/2$ when use random b-bit circulations instead of random 2-circulations, due to the independence of each bit.

We now give our simple linear-time algorithm to find all cut classes. The idea is to compute a random b-bit circulation for large enough b that $\phi(e) = \mathbf{0}$ only for cut edges, and so that ϕ labels the cut classes of other edges. Pseudocode is given in Algorithm 4.

Algorithm 4. Input: a connected graph G. Output: the cut classes of G.

1: Let $b = \lceil \log_2(|V||E|^2) \rceil$ and let ϕ be a random b-bit circulation on G.
2: For each $x \in \mathbb{Z}_2^b \backslash \{\mathbf{0}\}$, if $|\{e \in E \mid \phi(e) = x\}| \geq 2$, then output the cut class $\{e \in E \mid \phi(e) = x\}$.

Theorem 3. *Algorithm 4 correctly determines the cut pairs with probability* $1 - 1/V$ *and can be implemented in* $O(E)$ *sequential time.*

Proof. There are $|E|$ edges and Proposition 3 shows that $\Pr[\phi(e) = \mathbf{0}] \leq 1/2^b$ for each non-cut edge e. There are at most $\binom{E}{2}$ pairs $\{e, f\}$ of non-cut edges that are not cut pairs and Proposition 5 shows that $\Pr[\phi(e) = \phi(f)] \leq 1/2^b$ for each such pair. Hence, by a union bound and our choice $b = \lceil \log_2(|V||E|^2) \rceil$, the total probability of error is at most $|E|/2^b + \binom{E}{2}/2^b \leq 1/V$.

The subroutine Rand-b-Bit-Circ performs $O(E)$ random choices and xor operations on b-bit binary strings, each of which take $O(1)$ time since $b = O(\log V)$. To implement Line 2 in Algorithm 4 we sort all edges e according to the key $\phi(e)$. In particular, we use a three-pass *radix sort* (i.e., we consider each value in $\mathbb{Z}_2^b$ as a three-digit number in base $2^{b/3} = O(E)$ — see Cormen, Leiserson & Rivest [23, §9.3]), which runs in time $O(E)$. □

4.3 Finding All Cut Vertices

As we show in this section, the cut $\delta(v)$ properly contains smaller induced edge cuts iff v is a cut vertex. The essential idea behind our approach is to detect these induced edge cuts, in order to determine the cut vertices. We detect induced edge cuts via Proposition 1, which says that circulations vanish across induced edge cuts. To do so efficiently, we rephrase the detection problem as one of finding linearly dependent rows of a binary matrix. Hence we need the following fact, when $\mathbb{Z}_2$ is viewed as a field.

Fact 1. *In a matrix over $\mathbb{Z}_2$, a set C of columns is linearly dependent if and only if some nonempty subset of C sums to the zero column vector* $(\mathrm{mod}\ 2)$.

Our approach works as follows. We generate a random b-bit circulation ϕ for some suitably large b; denote the ith bit of $\phi(e)$ by $\phi_i(e)$. Let $d(v) := |\delta(v)|$, the *degree* of v. Let Δ denote the maximum degree. For each vertex v, let $M^{[v]}$ be a matrix with b rows indexed $1, \ldots, b$, and $d(v)$ columns indexed by $\delta(v)$; then fill the entries of $M^{[v]}$ according to $M^{[v]}_{ie} = \phi_i(e)$. The following two complementary claims, whose proofs we defer to the full version, underlie our approach.

Claim 1. *If v is a cut vertex then* $\mathrm{rank}(M^{[v]}) \leq d(v) - 2$.

Claim 2. *Let $v \in V$ and assume that v is not a cut vertex. Let $\emptyset \subsetneq D \subsetneq \delta(v)$. The probability that the columns of $M^{[v]}$ indexed by D sum to the zero vector* $(\mathrm{mod}\ 2)$ *is* 2^{-b}.

Next we show that for $b = \lceil \Delta + 2\log_2 |V| \rceil$, it is very likely that $\mathrm{rank}(M^{[v]}) < d(v) - 1$ iff v is a cut vertex. Thus our approach, with pseudocode given in Algorithm 5, is correct with high probability. It is not very efficient in the sequential model, but still runs in $\mathrm{poly}(V)$ time.

Algorithm 5. Input: a connected graph G. Output: the cut vertices of G.

1: Let $b = \lceil \Delta + 2\log_2 |V| \rceil$ and let ϕ be a random b-bit circulation on G.
2: for each vertex v of G, if $\mathrm{rank}(M^{[v]}) < d(v) - 1$ then output v.

Theorem 4. *Algorithm 5 correctly determines the cut vertices with probability* $1 - 1/V$.

Proof. Claim 1 shows that all cut vertices are output. Consider a vertex v that is not a cut vertex and let D be a subset of $\delta(v)$ of size $d(v) - 1$. By Claim 2, Fact 1, and a union bound, the probability that the columns of $M^{[v]}$ corresponding to D are linearly dependent is at most $2^{d(v)-1}2^{-b} \leq 1/|V|^2$; so with probability $1 - |V|^{-2}$, we have $\mathrm{rank}(M^{[v]}) \geq |D| = d(v) - 1$ and v is not output. By another union bound, the probability that any vertex is misclassified by Algorithm 5 is at most $|V|/|V|^2 = 1/|V|$. □

5 Distributed Implementation

Our algorithms make the following three assumptions: first, the network is synchronous; second, there is a distinguished *leader* vertex at the start of computation; third, every node begins with a unique $O(\log V)$-bit ID. These assumptions are standard in the sense that they are made by the best previous distributed algorithms [6,9,11] for small cuts. Nonetheless, these assumptions can be removed at a cost if desired, e.g. using the synchronizer of Awerbuch and Peleg [24] at a $\mathrm{polylog}(V)$ factor increase in complexity, Peleg's [25] $O(\mathcal{D})$-time leader election

algorithm, or by randomly assigning IDs in the range $\{1, \ldots, |V|^3\}$ (resulting in additional failure probability at most $\binom{V}{2}/|V|^3$ due to ID collisions).

Although only vertices can store data in the distributed model, we maintain data for each edge e (e.g., to represent a tree) by having both endpoints of e store the data. At the end of the algorithm, we require that the correct result is known locally, so each node stores a boolean variable indicating whether it is a cut node, and similarly for edges. To indicate cut pairs, each edge must know whether it is in any cut pair, and in addition we must give every cut class a distinct label. Previous work also essentially uses these representations.

When stating distributed algorithms, the assumptions of a leader, synchrony, unique IDs, and $O(\log V)$-bit messages are implicit. Our algorithms use a breadth-first search (BFS) tree with a root r as the basis for communication. One reason that BFS trees are useful is that they can be constructed quickly (e.g., see Peleg [11, §5.1]), as follows.

Proposition 6. *There is a distributed algorithm to construct a BFS tree in $O(\mathcal{D})$ time and $O(E)$ messages.*

For a tree T, the *level* $l(v)$ of $v \in V$ is the distance in T between v and r. The *height* $h(T)$ of tree T is the maximum vertex level in T. Any BFS tree T has $h(T) \leq \mathcal{D}$ and this is important because several fundamental algorithms based on passing information up or down the tree take $O(h(T))$ time. The *parent* of u is denoted $p(u)$. The *level of tree edge* $\{u, p(u)\}$ is the level of u.

5.1 Random Circulations, Cut Edges, and Cut Vertices

When we construct a random circulation, we require at termination that each v knows $\phi(v, u)$ for each $u \in \Gamma(v)$.

Theorem 5. *There is a distributed algorithm to sample a uniformly random k-circulation in $O(\mathcal{D})$ time and $O(E)$ messages, when $k = \text{poly}(V)$.*

Proof. We implement Rand-k-Circ distributively. Since $k = \text{poly}(V)$, any value in $\mathbb{Z}_k$ can be sent in a single $O(\log V)$-bit message. We compute a BFS tree T, using Proposition 6. Then for each non-tree edge $\{u, v\}$ in parallel, the endpoint with the higher ID (say, u) sets $\phi(u, v)$ to a random value in $\mathbb{Z}_k$, sends the value $\phi(u, v)$ to v, and then v sets $\phi(v, u) := -\phi(u, v)$. In the following $h(T)$ rounds, for $i = h(T)$ down to 1, for all level-i tree edges $\{v, p(v)\}$ in parallel, vertex v assigns $\phi(v, p(v))$ a value so that conservation is satisfied at v (like in Algorithm 1), notifies $p(v)$ of this value with a message, and then $p(v)$ sets $\phi(p(v), v) := -\phi(v, p(v))$. Termination occurs after r computes its incident ϕ values. This takes $O(\mathcal{D} + h(T)) = O(\mathcal{D})$ time and $O(E)$ messages, as claimed. □

Theorem 6. *There is a distributed algorithm to sample a uniformly random b-bit circulation in $O(\mathcal{D})$ time and $O(E)$ messages, when $b = O(\log V)$.*

Proof. Any value in $\mathbb{Z}_2^b$ can be sent in a single $O(\log V)$-bit message. Thus, analogous to the proof of Theorem 5, we implement Rand-b-Bit-Circ distributively. □

Theorem 5 yields our distributed cut edge algorithm.

Theorem 7. *There is a distributed algorithm to compute all cut edges with probability* $1 - 1/V$ *in* $O(\mathcal{D})$ *time and using* $O(E)$ *messages.*

Proof. We implement Algorithm 3 distributively, obtaining the required correctness probability by Theorem 2. For $k = |V||E|$, we use Theorem 5 to compute a random k-circulation in the required complexity bounds. Then we identify $\{u, v\}$ as a cut edge if $\phi(u, v) = 0$. □

A straightforward implementation of Algorithm 5 results in our cut vertex algorithm, as follows.

Theorem 8. *There is a distributed algorithm to compute all cut vertices with probability* $1 - 1/V$ *in* $O(\mathcal{D} + \Delta/\log V)$ *time and using* $O(E(1 + \Delta/\log V))$ *messages.*

Proof (Sketch). Using Theorem 6 and a *pipelining* technique — which we defer to the full version — we can sample a random b-bit circulation in $O(\mathcal{D} + b/\log V)$ time using $O(E(1 + b/\log V))$ messages. We take $b = \lceil \Delta + 2\log_2 |V| \rceil$ as in Algorithm 5. Since local computations are free in the distributed model, each vertex v can immediately compute $\text{rank}(M^{[v]})$. □

5.2 Fundamental Cycle-Cast (fc-cast)

We now define a new distributed technique. A non-tree edge is an edge $e \in E \backslash E(T)$. For a spanning tree T and non-tree edge e, the unique cycle in $T \cup \{e\}$ is called *the fundamental cycle of* T *and* e, and we denote it by C_e. We call our new technique *fundamental cycle-cast*, or *fc-cast* for short, and informally it allows simultaneous processing on all fundamental cycles. Let each vertex v store some data $\mathtt{d}[v]$ of length $O(\log V)$ bits. WOLOG $\mathtt{d}[v]$ includes the ID, level, and parent ID of v. At the end of the fc-cast, each non-tree edge e will know $\mathtt{d}[u]$ for every vertex u in the fundamental cycle of T and e. We defer the proof of Theorem 9 to the full version.

Theorem 9. *There is a distributed algorithm* Fc-Cast *using* $O(h(T))$ *time and* $O(\min\{E \cdot h(T), V^2\})$ *messages that, for each non-tree edge* e*, for each* $v \in C_e$*, sends* $\mathtt{d}[v]$ *to both endpoints of* e*.*

5.3 Distributed Cut Pair Algorithm

It is not obvious how to implement our sequential cut pair algorithm (Algorithm 4) distributively: although the cut classes are properly labeled with high probability by ϕ, in order for edge e to know whether it belongs to any cut pair, it needs to determine if any other f has $\phi(e) = \phi(f)$, and this cannot be done using local information (i.e., in $O(1)$ rounds). We use fc-cast to overcome this obstacle. The following claim, whose proof we omit, relates fundamental cycles to cut classes.

Claim 3. *Let K be a cut class. Then $K \subset C_e$ for some $e \in E \backslash E(T)$.*

To describe our cut pair algorithm we use a variant of a standard technique, the *convergecast* (e.g., see Peleg [1, §4.2]). Informally, it allows each node to independently query its descendants. Let $desc(v)$ denote the set of v's descendants, including v itself. For each $v \in V$, and each $u \in desc(v)$, let $\mathtt{w}[u, v]$ be a boolean variable stored at u. We state the protocol and then give its application to cut pairs.

Proposition 7. *There is a distributed algorithm* Converge-Cast *using $O(h(T))$ time and $O(V \cdot h(T))$ messages so that each $v \in V$ determines whether any $u \in desc(v)$ has $\mathtt{w}[u, v] =$* TRUE.

Theorem 10. *There is a distributed algorithm to compute all cut classes with probability $1 - 1/V$ in $O(\mathcal{D})$ time and using $O(\min\{E \cdot \mathcal{D}, V^2\})$ messages.*

Proof. We will use two claims below; their proofs are deferred to the full version. As in Algorithm 4, for $b = \lceil \log_2(|V||E|^2) \rceil$ we compute a random b-bit circulation ϕ on G, using Theorem 6. Denote the following assumption by ($\star$).

For all edges e, f that are not cut edges, $\phi(e) = \phi(f)$ if and only if $\{e, f\}$ is a cut pair. ($\star$)

By the analysis in the proof of Theorem 3, we may assume that ($\star$) holds without violating the required bound of $1/V$ on the probability of error.

It remains only for each edge to determine whether it is a member of any cut pair, since then ϕ labels the cut classes. For each vertex $v \neq r$ let $\mathtt{d}[v] := \phi(v, p(v))$. We run Fc-Cast, and as a result, the endpoints of each non-tree edge e can compute the multiset $\Phi_e := \{\phi(f) \mid f \in C_e\}$. The following claim lets each non-tree edge determine if it is a member of any cut pair.

Claim 4. *A non-tree edge e is in a cut pair if and only if $\phi(e)$ occurs multiple times in Φ_e.*

To deal with tree edges, for each $v \in V$ and each $u \in desc(v)$, define

$\mathtt{w}[u, v] := (\exists e \in \delta(u) \backslash E(T)$ such that $\{v, p(v)\} \in C_e$ and $\phi(v, p(v))$ occurs multiple times in $\Phi_e)$.

and note that $\mathtt{w}[u, v]$ can be determined by u after the fc-cast. We run Converge-Cast.

Claim 5. *Tree edge $\{v, p(v)\}$ is in a cut pair if and only if $\exists u \in desc(v)$ such that $\mathtt{w}[u, v] =$* TRUE.

By Proposition 7, after the convergecast, each tree edge can use Claim 5 to determine if it is a member of any cut pair. Adding up the complexity associated with constructing a BFS tree and a random circulation, the fc-cast, and the converge-cast, we obtain $O(\mathcal{D}+\mathcal{D}+\mathcal{D}+\mathcal{D})$ time and $O(E+E+\min\{E\mathcal{D}, V^2\}+V\mathcal{D}) = O(\min\{E\mathcal{D}, V^2\})$ messages, as claimed. □

References

1. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM (2000)
2. Lotker, Z., Patt-Shamir, B., Peleg, D.: Distributed MST for constant diameter graphs. Distributed Computing 18(6), 453–460 (2006); Preliminary version In: Proc. 20th PODC, pp. 63–71 (2001)
3. Tarjan, R.: Depth first search and linear graph algorithms. SIAM J.Comput. 1(2), 146–160 (1972)
4. Hopcroft, J., Tarjan, R.: Dividing a graph into triconnected components. SIAM J. Comp. 2(3), 135–158 (1973)
5. Chang, E.J.H.: Echo algorithms: Depth parallel operations on general graphs. IEEE Trans. Softw. Eng. SE-8, 391–401 (1982)
6. Ahuja, M., Zhu, Y.: An efficient distributed algorithm for finding articulation points, bridges, and biconnected components in asynchronous networks. In: Veni Madhavan, C.E. (ed.) FSTTCS 1989. LNCS, vol. 405, pp. 99–108. Springer, Heidelberg (1989)
7. Hohberg, W.: How to find biconnected components in distributed networks. J. Parallel Distrib. Comput. 9(4), 374–386 (1990)
8. Huang, S.T.: A new distributed algorithm for the biconnectivity problem. In: Proc. 1989 International Conf. Parallel Processing, pp. 106–113 (1989)
9. Thurimella, R.: Sub-linear distributed algorithms for sparse certificates and biconnected components. J. Algorithms 23(1), 160–179 (1997); Preliminary version In: Proc. 14th PODC, pp. 28–37 (1995)
10. Jennings, E., Motyckova, L.: Distributed computation and incremental maintainance of 3-edge-connected components. In: Proc. 3rd SIROCCO, pp. 224–240 (1996)
11. Tsin, Y.H.: An efficient distributed algorithm for 3-edge-connectivity. Int. J. Found. Comput. Sci. 17(3), 677–702 (2006)
12. Garay, J.A., Kutten, S., Peleg, D.: A sublinear time distributed algorithm for minimum-weight spanning trees. SIAM J.Comput. 27(1), 302–316 (1998)
13. Tarjan, R.E., Vishkin, U.: An efficient parallel biconnectivity algorithm. SIAM J.Comput. 14(4), 862–874 (1985); Preliminary version In: Proc. 25th FOCS, pp. 12–20 (1984)
14. Fussell, D.S., Ramachandran, V., Thurimella, R.: Finding triconnected components by local replacement. SIAM J.Comput. 22, 587–616 (1993)
15. Galil, Z., Italiano, G.: Reducing edge connectivity to vertex connectivity. SIGACT News 22, 57–61 (1991)
16. Halperin, S., Zwick, U.: Optimal randomized EREW PRAM algorithms for finding spanning forests. J. Algorithms 39(1), 1–46 (2001); Preliminary version In: Proc. 7th SODA, pp. 438–447 (1996)
17. Tsin, Y.H.: A simple 3-edge-connected component algorithm. Theory Comput. Systems 40(2), 125–142 (2005)
18. Hoffman, A.: Some recent applications of the theory of linear inequalities to extremal combinatorial analysis. In: Proc. 10th AMS Symp. on Appl. Math., pp. 113–127 (1960)
19. Zhang, C.Q.: Integer flows and cycle covers of graphs. Marcel Dekker, New York (1997)

20. Borradaile, G., Klein, P.: An O $(n \log n)$ algorithm for maximum st-flow in a directed planar graph. In: Proc. 17th SODA, pp. 524–533 (2006)
21. Benjamini, I., Lovász, L.: Harmonic and analytic functions on graphs. J. Geom. 76(1), 3–15 (2003); Preliminary version In: Proc. 43rd FOCS, pp. 701–710 (2002)
22. Bondy, A., Murty, U.: Graph Theory with Applications. North-Holland, Amsterdam (1976)
23. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
24. Awerbuch, B., Peleg, D.: Network synchronization with polylogarithmic overhead. In: Proc. 31st FOCS, pp. 514–522 (1990)
25. Peleg, D.: Time-optimal leader election in general networks. J. Parallel Distrib. Comput. 8(1), 96–99 (1990)

Finding a Maximum Matching in a Sparse Random Graph in $O(n)$ Expected Time

Prasad Chebolu, Alan Frieze*, and Páll Melsted

Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh PA15213
U.S.A.

Abstract. We present a linear expected time algorithm for finding maximum cardinality matchings in sparse random graphs. This is optimal and improves on previous results by a logarithmic factor.

1 Introduction

A matching M in a graph $G = (V, E)$ is a set of vertex disjoint edges. The problem of computing a matching of maximum size is central to the theory of algorithms and has been subject to intense study. Edmond's landmark paper [3] gave the first polynomial time algorithm for the problem. Micali and Vazirani [6] reduced the running time to $O(mn^{1/2})$ where $n = |V|$ and $m = |E|$. These are worst-case results. In terms of average case results we have Motwani [7] and Bast, Mehlhorn, Schäfer and Tamaki [2] who have algorithms that run in $O(m \log n)$ expected time on the random graph $G_{n,m}$, in which each graph with vertex set $[n]$ and m edges is equally likely.

One natural approach to finding a maximum matching is to use a simple algorithm to find an initial matching and then augment it. This will not work in the worst-case, but as we will show, it can be used to obtain an $O(n)$ expected time algorithm for graphs with constant average degree ($O(m)$ in general). For a simple algorithm we go to the seminal paper of Karp and Sipser [5]. They describe a simple greedy algorithm and show that **whp** it will in linear time produce a matching that is within $o(n)$ of the maximum. Aronson, Frieze and Pittel [1] proved that **whp** the Karp-Sipser algorithm is off from the maximum by at most $\tilde{O}(n^{1/5})$. In this paper we show that **whp** we can take the output of the Karp-Sipser algorithm and augment it in $o(n)$ time to find a truly maximum matching. Our failure probability will be $o(1/\log n)$ and so we get a linear expected time algorithm if we back it up with the algorithm from [2]. We will define an algorithm Match and prove

Theorem 1. *Let $2m = cn$ where c is a sufficiently large constant. Let $G = G_{n,m}$. Then the algorithm Match finds a maximum matching in G in $O(n)$ expected time.*

* Supported in part by NSF Grant CCF-0502793.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 161–172, 2008.

1.1 The Karp-Sipser Algorithm

This is a simple greedy algorithm. If the current graph G has a vertex of degree one, then it chooses one such vertex v at random and adds the unique edge (u, v) to the matching it has found so far and deletes the vertices u, v and continues. If the current graph has minimum degree at least two then it picks a random edge (u, v), adds this to the matching and deletes u, v and continues. The algorithm stops when G has no edges. Algorithm 1 below is a formal description.

Algorithm 1. Karp-Sipser Algorithm

```
procedure KSGreedy(G)
    M ← ∅
    while G ≠ ∅ do
        if G has vertices of degree 1 then
            Select a vertex v uniformly at random from the set of vertices of degree 1
            Let (v, u) be the edge incident to v
        else
            Select an edge (v, u) uniformly at random
        end if
        M ← M ∪ (v, u)
        G ← G \ {v, u}
    end while
    return M
end procedure
```

We identify two phases in the execution of the Karp-Sipser algorithm. Phase one starts at the beginning and ends when the current graph has minimum at least degree two. We note that if M_1 is the set of edges chosen in Phase 1 then there is some maximum cardinality matching that contains M_1, i.e. no mistakes have been made so far.

Let the current graph at the beginning of Phase 2 be denoted by G'. As shown in [5], almost all vertices of G' are matched by the Karp-Sipser algorithm when G is a random graph. This result was improved in [1] to show in fact that **whp** all but $\tilde{O}(n^{1/5})$ vertices of G' are matched. When $G \simeq G_{n,m}$, G' is distributed as $G' \simeq G^{\delta\geq 2}_{n',m'}$ where G' has n' vertices and m' edges and $G^{\delta\geq 2}_{n',m'}$ is uniformly chosen from simple graphs with n' vertices, m' edges and minimum degree ≥ 2. The values m', n' are random variables which will be concentrated around their known means. It was further shown in Frieze and Pittel [4] that **whp** G' consists of a single giant component plus a collection of vertex disjoint cycles. The expected number of vertices on isolated cycles is $O(1)$. It is shown that **whp**, a maximum cardinality matching of G' matches every vertex except one for each isolated odd cycle and one vertex if the giant component of G' is odd [4]. (This is an existence result, non-algorithmic). So, after running the Karp-Sipser algorithm and dealing with isolated odd cycles, our task will **whp** be to match together $\tilde{O}(n^{1/5})$ isolated vertices.

1.2 Outline Decription of Match

We will take the output of the Karp-Sipser algorithm, remove small cyclic components and deal with them separately. We then take the isolated vertices of the graph in pairs and try to match them together using alternating paths. We will show that this can be done in $o(n)$ time **whp**. Our augmenting path phase will use all of the edges of the graph. The reader will be aware that the Karp-Sipser algorithm has conditioned the edges of the graph. We will show however that we can find a large set of edges A and show that they have an *understandable* conditional distribution. This distribution will be simple enough that we can make use of A to show that we succeed **whp**. Intuitively, we can do this because the Karp-Sipser algorithm only "looks" at a small number of edges and discards most of the edges incident with the pair u, v chosen at each step. Dealing with conditioning is a central problem in Probabilistic Analysis. Oft times it is achieved by the use of concentration. Here the problem is more subtle. Note that one cannot simply run the Karp-Sipser algorithm on a random subgraph $G_{n,m_1} \subseteq G_{n,m}$ and then use the $m - m_1$ random edges. This is because Phase 1 on the sub-graph will leave extra isolated vertices.

Algorithm 2 below is a formal description of Match. Note that the Augment algorithm takes a graph and a matching as arguments and returns an improved matching.

Algorithm 2. Algorithm Match

```
1: procedure MAIN(G)
2:     (M, G') ← KS-Greedy(G)        ▷ G' is the graph G after Phase 1 of KS-Greedy
3:     G' ← Remove-small-components(G')
4:     M' ← Augment(G', M ∩ G')
5:     return M △ M'
6: end procedure
1: procedure REMOVE-SMALL-COMPONENTS(G)
2:     Pick an arbitrary vertex u ∈ G and run a breadth first search starting from u
3:     if the connected component containing u, C_u, has size less than log^2 n then
4:         G ← G \ C_u
5:     end if
6:     return G
7: end procedure
```

The rest of the paper is structured as follows: We describe our augmenting path algorithm in the next section. Then in Section 3 we discuss the conditioning imposed by the Karp-Sipser algorithm. Then in Section 4 we add the final touches to the proof.

2 Augmenting Path Algorithm

A vertex is said to be *unmatched* if it is not incident to a matching edge. Given an unmatched vertex u, an *augmenting tree* T_u will be a tree of even depth rooted

at u such that an edge between vertices at depth $2k$ to $2k+1$ is not a matching edge and edges going from vertices at depth $2k+1$ to $2k+2$ are matching edges, for $k \geq 0$. We refer to the nodes at levels $2k$ as even nodes of the tree and nodes at level $2k-1$ as odd nodes for $k \geq 1$. We refer to the leaves of T_u as the front of the tree. Our growth procedure ensures that the leaves are always even vertices.

A *blossom* rooted at v is a cycle of odd length where the edges on the path starting and ending at v alternate between matching and non-matching edges.

Given two augmenting trees T_u, T_v, rooted at u, v respectively, a *hit edge* is an edge (x, y) such that x is an even node in T_u and y is an odd node in T_v. Note that given a hit edge (x, y) the subtree of T_v rooted at y can be taken from T_v and added to T_u, by removing the edge from y to its parent node in T_v (see figure 2). An edge (x, y) s.t. x and y are even nodes of the same tree is called a *blossom edge*.

Throughout the algorithm we will keep track of which vertices we have seen before, labeling some vertices as *exposed*. This is mainly to keep track of which vertices have "no randomness" left because we have seen all the vertices they are adjacent to. The algorithm Augment(G, M) takes as input a graph G and a matching M. The algorithm runs in rounds, where in each round we try to find an augmenting path between two unmatched vertices. If such a path is found, the matching is augmented, if not the algorithm returns Failure and we resort to an alternate algorithm. This is repeated until there is at most one unmatched vertex left, and then the current matching is returned. Clearly, if the algorithm does not fail then it finds a maximum cardinality matching.

In each round of the algorithm two augmenting trees are maintained, T_u, T_v, which are rooted at two unmatched vertices u and v. The trees are grown one at a time until we either find an augmenting path or the trees cannot be grown further. For each of u, v we maintain a list of blossom edges and hit edges encountered so far.

The smaller of the two trees is grown, unless one tree has $\leq n^{.59}$ unexposed vertices at the front and the other has $> n^{.59}$ unexposed vertices at the front.

Suppose T_u is to be grown. Then for each vertex x on the front of T_u and each non-matching edge (x, y) we do one of the following:

1. If y belongs to neither of the trees and is matched, add it to T_u along with its matching edge (y, y')
2. If y is unmatched we have an augmenting path from u to y
3. If y is an even vertex of T_v then the path from u to x in T_u, with the edge (x, y) and the path from y to v in T_v forms an augmenting path
4. If y is an odd vertex of T_v then (x, y) is a hit edge, append it to the list of hit edges for u.
5. If y is an even vertex of T_u then (x, y) along with the paths from x, y to their most common ancestor in T_u form a blossom, append (x, y) to the list of blossom edges for u.
6. If y is an odd vertex of T_u we do nothing.

After examining all edges incident to x, label it as exposed.

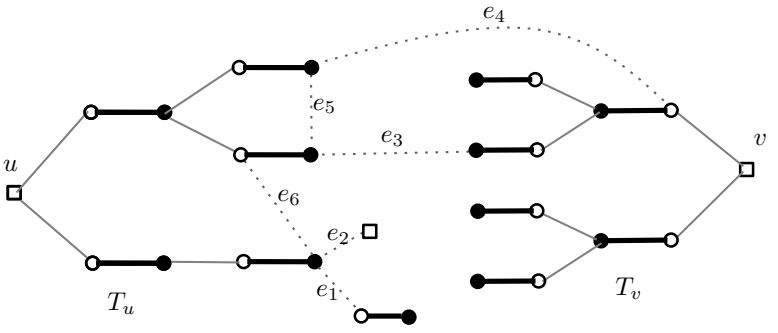


Fig. 1. The trees T_u and T_v are shown with bold edges. The edges e_i correspond to cases i in the algorithm for $i = 1, \ldots, 6$.

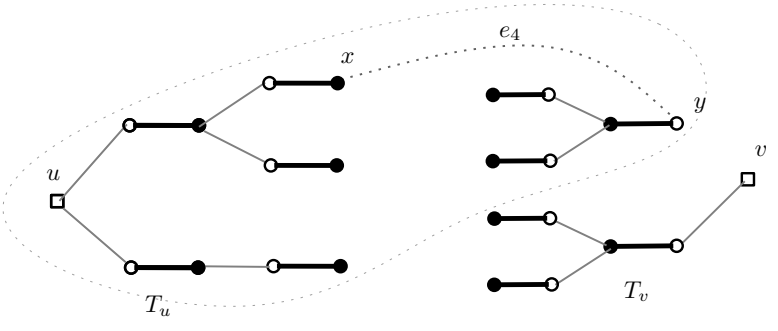


Fig. 2. The trees T_u and T_v after using the hit edge e_4

If an augmenting path is found then the round is finished. If the tree doesn't grow we inspect first the list of hit edges and see if we can grow the tree using them. If there are no hit edges we inspect the list of blossom edges. For each blossom found contract the blossom into a supernode and add this supernode to the front of the tree and try to grow from there. If the tree still doesn't grow exit the round and report failure. Examples of the 6 cases for the edge (x, y) are shown in Figure 1 and examples for hit edges and blossoms are shown in Figures 2 and 3.

2.1 Tree Expansion

We will show that there exist constants α_1, α_2 independent of c and a constant c_0 such that for $c \geq c_0$ and $2m = cn$ the following Lemmas hold. Proofs of these lemmas are left to the full paper. Note that we can choose α_2 to be arbitrarily small, but this would simply increase the bound on c_0.

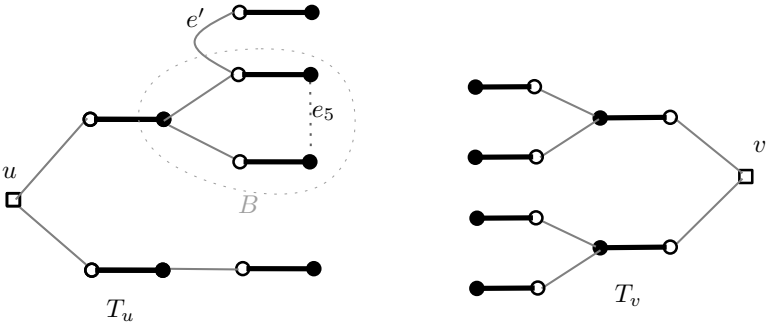


Fig. 3. The trees T_u and T_v after using the blossom edge e_5. Note that the blossom B is contracted and the edge e' becomes a part of the new tree.

Lemma 1. *The following will hold with probability* $1 - O(n^{-2})$. *For* $G \sim G^{\delta\geq 2}_{n,m}$ *and all matchings* M *of* G *and all augmenting trees* T *with* $c^{-1}\alpha_1 \log n \leq |T| \leq n^{.99}$. T *will expand to a new front of size* $s \in \left[\frac{9c}{10}|T|, \frac{11c}{10}|T|\right]$ *in one round.*

Lemma 2. *Let* $G \sim G^{\delta\geq 2}_{n,m}$, *then with probability* $1 - O(n^{-1+3\alpha_2})$ *there do not exist two cycles of length* a *and* b, *at distance* d *apart for any* a, b, d *such that* $a + b + d \leq \alpha_2 \log_c n$

Lemma 3. *Let* $G \sim G^{\delta\geq 2}_{n,m}$, *then with probability* $1 - O(n^{-2})$ *there does not exist a set* S *with* $\log n \leq |S| \leq n^{.99}$ *that has more than* $(1 + \epsilon)|S|$ *edges inside* S *for all* $\epsilon > 0$.

Lemma 4. *Let* G *be a graph such that Lemmas 1, 2 and 3 hold. For all matchings* M *of* G, *if Augment*(G, M) *returns Failure, then the trees grown must be of size* $\Omega(n^{.8})$.

3 Karp-Sipser Conditioning

We now view G as an ordered set of edges and look at an equivalent version of the Karp-Sipser algorithm. In the analysis of Karp-Sipser on random graphs we have two sources of randomness. One is the random graph itself and the other one is the random choices made by the algorithm. In order to simplify the analysis we change the choices into deterministic ones and simply randomize the order in which the edges are stored and take them in this (random) order. This is equivalent to original algorithm. We now state the modified Karp-Sipser algorithm. We assume the graph is given as $G = (e_1, \ldots, e_m)$ an ordered set of edges. We call say that edge $e \in G$ has index i if it is the i-th edge in the list, i.e. $e = e_i$. Note that every graph in the support of $G^{\delta\geq 2}_{n,m}$ will yield $m!$ ordered sets of edges, so from now on we will think of $G^{\delta\geq 2}_{n,m}$ as a family of ordered sets of edges. (We only need to concern ourselves with Phase 2 of the Karp-Sipser algorithm and we have replaced m', n' by m, n for notational convenience).

```
1: procedure KS*(G)
2:     M ← ∅
3:     while G ≠ ∅ do
4:         if G has vertices of degree 1 then
5:             Of all edges incident to vertices of degree 1, let e have the lowest
   index
6:             Let e = (v, u) where v has degree 1.
7:         else
8:             Let e = (v, u) be the edge of lowest index in G
9:         end if
10:        M ← M ∪ (v, u)
11:        G ← G \ {u, v}
12:    end while
13:    return M
14: end procedure
```

3.1 Witness Edges

In addition to the edges of the matching we define edges based on the run of the algorithm. We split the vertices of the graph into three classes, regular, pendant and unmatched. A vertex is regular if when it was removed from the graph, it had degree 2 or more. A vertex is said to be a pendant vertex if when it was removed it had degree exactly 1 and is the endpoint of a matching edge in M. Unmatched vertices are those vertices that are not incident to matching edges. We say that an edge e is regular if both of its endpoints are regular, i.e. it was removed from the graph in line 8. For each of these vertices we define witness edges.

- For a regular vertex v, it is removed from the graph when the edge e is picked as a matching edge. Since it has degree at least 2, there are other edges incident to it at the time it is removed. Pick the one with the lowest index and define it to be the *regular witness edge* for v.
- For a pendant vertex or an unmatched v. Find the last point of time when v has degree at least 2, an edge $e = (x, y)$ is removed from the graph and v is incident to at least one of them (perhaps both), say x. We then define (v, x) to be the *pendant witness edge* for v.
- For an unmatched vertex v, it has a pendant witness edge, and since it is never picked for a matching its last edge is incident to some matching edge $e = (x, y)$, say x, we then define (v, x) to be the *removal witness edge* for v.
- In case of any ambiguities, define pendant witness edges first and then removal witness edges. Use the lowest index of edges to break all ties. This can happen if a vertex goes from having degree 3 to pendant or from having degree 2 to degree 0 if it is incident to both endpoints of a matched edge.
- Note that an edge e can be a regular witness edge for one vertex and a pendant or removal witness edge for another vertex.

Let W be the set of witness edges. Regular and pendant vertices are incident to matching edges and their witness edges. Unmatched vertices are incident to two witness edges. Hence the graph defined by M and W has minimum degree 2 and size at most $2n$.

We think of the graph G as an ordered set of m boxes filled with edges. Suppose we know the output of KS*, M, W and also the order in which the matching and witness edges were added to M and W, but the underlying graph is unknown to us. This corresponds to m ordered boxes, of which the ones corresponding to M and W have been opened. We wish to figure out what the unopened boxes could possibly contain. The following lemma provides necessary and sufficient conditions for a graph G to yield M and W as the output of KS*.

Lemma 5. *Let G be a graph such that the algorithm KS* will produce the matching set M and witness set W. If e is an edge of G that belongs neither to M nor W then KS* will produce the same matching and witness set when run on $G'' = G \setminus \{e\}$. Let $e' = (u, v)$ be an edge not in G that satisfies conditions 1,2,3 below. Then KS* will produce the same matching and witness set M and W when run on $G' = G \cup \{e'\}$.*

1. *If both u and v are regular vertices and say u was removed from the graph before v then (u, v) can appear in any box that comes after the regular witness edge for u.*
2. *If u is a regular vertex and v is either a pendant or unmatched vertex . We need v to have degree at least 2 at the time when u is removed. Thus we need (u, v) to appear in a box that comes after the regular witness edge for u. Additionally if the pendant witness for v is incident to the matching edge of u we need (u, v) to appear in a box that comes after the pendant witness for v.*
3. *If neither u nor v are regular vertices, then the edge (u, v) cannot appear in the graph.*

Note that it is possible to add an edge to the graph that will produce the same set of matching edges, but a different witness set. Since we want to condition on both sets, and the exact order in which they were produced we are not interested in such cases.

3.2 Probability Space

We describe the probability space after we sample a random graph from $G_{n,m}^{\delta \geq 2}$ and run KS* on the graph and condition on the output matching edges M, as well as the witness edges W. Given the output M and W and Rules 1-3 we can find all graphs that would give M and W as the output of KS* and generate one uniformly at random.

First note that for each box i that is not in M or W we can create a list of edges E_i that could go into that box, from Rules 1-3 we see that this list depends only on M and W and is independent of the contents of other boxes. Also note that all the rules state that an edge can go into any box that comes after some

specified box, thus we have $E_i \subseteq E_j$ when $i < j$. This leads us to the following algorithm for generating a random graph from the distribution $G_{n,m}^{\delta\geq 2}|M, W$, i.e. conditioned on the output of KS*.

```
1: procedure GENERATE-RANDOM(M,W)
2:     for unfilled boxes i do
3:         E_i ← { all edges e that can go into box i}
4:     end for
5:     G ← M ∪ W
6:     for unfilled boxes i in increasing order do
7:         Select e uniformly at random from E_i
8:         G ← G ∪ {e}
9:         Remove e from E_j for all j > i
10:    end for
11:    return G
12: end procedure
```

Each G that outputs M and W can be generated with Generate-Random in exactly one way and that any graph G produced by Generate-Random will produce M and G when we run KS* on G. This shows that Generate-Random will give a uniformly random graph from $G_{n,m}^{\delta\geq 2}|M, W$.

4 Final Proof

In Section 3.2 we gave a complete description of the probability space. However this is not enough to finish the proof of Theorem 1, we must dig deeper into the analysis of KSGreedy. We begin by listing some definitions and lemmas from the paper that we will need.

In [1] it is shown that $G(t)$ is distributed uniformly at random from the set of all graphs with $v_0(t)$ vertices of degree 0, $v_1(t)$ vertices of degree 1, $v(t)$ vertices of degree at least 2 and $m(t)$ edges, we denote this sequence by $\boldsymbol{v}(t) = (v_0(t), v_1(t), v(t), m(t))$. Furthermore, the sequence $\boldsymbol{v}(t)$ is a Markov chain. Thus the analysis of the algorithm is done by tracking the sequence $\boldsymbol{v}(t)$. Additionally we define $z(t)$ by

$$\frac{2m(t) - v_1(t)}{v(t)} = \frac{z(t)(e^{z(t)} - 1)}{f(z(t))}$$

where $f(z) = e^z - z - 1$. Conditional on $\boldsymbol{v}(t)$, the degrees of vertices of degree at least 2 is distributed as independent copies of a truncated Poisson random variable Z, where

$$\mathsf{P}(Z = k) = \frac{z^k}{k! f(z)} \quad k = 2, 3, \ldots$$

conditional on $\sum_{v:\deg(v)\geq 2} Z_v = 2m(t) - v_1(t)$.

As our input is taken from $G_{n,m}^{\delta\geq 2}$ we start in the state $\boldsymbol{v}(0) = (0, 0, n, m)$, i.e. with $v_1(0) = 0$. For $t_1 < t_2$ such that $v_1(t_1) = v_1(t_2) = 0$ and $v_1(t) > 0$ for $t_1 < t < t_2$ we look at the edges and vertices removed from t_1 to t_2, i.e. the

graph $G(t_1) \setminus G(t_2)$ and call it a *batch*. Note that each batch contains the regular matching edge removed at time t_1 and hence a batch is a connected set.

4.1 Good Matching Edges

Let τ_0 be the last time such that the number of vertices removed from the graph is at most $n^{.99}$. We refer to vertices removed before τ_0 as *early vertices* and those removed after as *late*. We say that a matching edge e is a *good matching edge* if it is early, both of its endpoints are regular and the regular witness edges for both of its endpoints have index less than $m/2$.

Note that for $t = 0, \ldots, \tau_0$ we have removed at most $n^{.99}$ vertices and $O(n^{.99})$ edges, thus

$$\frac{2m(t) - v_1(t)}{v(t)} = (1 + o(1))\frac{2m(0) - v_1(0)}{v(0)} = (1 + o(1))c$$

and $z(t) = (1 + o(1))z(0)$ and $z(t)$ is bounded away from 0 by a constant. Corollary 3 of [1] then gives that $E[v_1(t + 1) - v(t)] \leq -\alpha$ for some positive constant α. Using this α in Lemmas 13 and 14 in [1] gives the following lemma

Lemma 6.

$$\mathsf{P}\left(\exists t \leq \tau_0 : v_1(t) > \frac{4\log^3 n}{\alpha}\right) = O(n^{-4})$$

and

$$\mathsf{P}\left(\exists t \leq \tau_0 - T : v_1(t) = 0, v_1(t') > 0 \text{ for } t < t' \leq t + T\right) = O(n^{-4})$$

for $T = \frac{16\log^3 n}{\alpha^3}$.

This shows that for $t \leq \tau_0$ each batch corresponds to an interval of time of length at most $O(\log^3 n)$ and that the total number of pendant vertices (at the time of removal) is $O(\log^6(n))$, which implies that the number of vertices in a batch is also $O(\log^6(n))$.

This also shows that during the first τ_0 time steps there will be at least $\Omega\left(\frac{n^{.99}}{\log^6(n)}\right)$ times when $v_1(t) = 0$ and thus at least that many regular edges are added to the matching set.

Lemma 7. *There are* $\Omega\left(\frac{n^{.99}}{\log^6 n}\right)$ *good matching edges in* G.

4.2 The Batch Graph

We split the edges removed up to time τ_0 into batches $B_1, \ldots, B_l$ and create a *Batch Graph* G_B, with vertices $B_1, \ldots, B_l$ and we put an edge between B_i and B_j if $\text{dist}(B_i, B_j) \leq 20\log_c(\log n)$.

Lemma 8. *The probability that there exists a connected component in* G_B *of size at least* 1000 *is* $O(n^{-4})$.

Lemma 9. *Let T be an augmenting tree of size $|T| = \Omega(n^{.02})$. Then,* **whp**, *at there are at least $\frac{|T|}{\log n}$ late vertices on the front of the tree.*

Lemma 10. *Let T be an augmenting tree with $o(n^{.8})$ exposed vertices and $s = \tilde{\Omega}(n^{.03})$ unexposed vertices at the front. Then,* **whp**, *there are $\tilde{\Omega}(\frac{s}{n^{.02}})$ unexposed vertices at the front, whose matching edge in the augmenting tree is a good matching edge.*

4.3 Putting It All Together

Proof of Theorem 1: We show that in each round the algorithm will always find an augmenting path and will find one by exposing at most $\tilde{O}(n^{.59})$ new vertices. This implies that the amount of work done in the i-th round is $\tilde{O}(i \cdot n^{.59})$, since we could in the worst case visit all previously exposed vertices. So the total work would be $\tilde{O}(l^2 n^{.59}) = \tilde{O}(n^{.99}) = o(n)$, where l is the total number of rounds and $l = \tilde{O}(n^{.2})$ since the number of unmatched vertices is $\tilde{O}(n^{.2})$.

Consider the i-th round and assume we've only exposed $\tilde{O}(i \cdot n^{.59}) = o(n^{.8})$ vertices. By Lemma 4 we know that **whp** the algorithm will be able to grow the trees and that we can assume that the trees T_u and T_v both have at least $n^{.59}$ unexposed vertices at the front. If the algorithm found an augmenting path before the first time this happened then we've exposed only $\tilde{O}(n^{.59})$ vertices, since there are at most $O(\log n)$ levels of the tree and each level has $\leq n^{.59}$ unexposed vertices. Assume therefore that we have two sets of unexposed vertices at the fronts S_u and S_v such that $|S_u|, |S_v| \geq n^{.59}$.

By Lemma 9 we know that there are at least $\frac{|S_u|}{\log n}$ late unexposed vertices in $|S_u|$, call this set S'_u. Since only $o(n^{.8})$ vertices have been exposed so far, Lemma 10 applies and there are at least $\tilde{\Omega}\left(\frac{|S_v|}{n^{.02}}\right)$ vertices in S_v that are endpoints of good matching edges, call this set S'_v. Thus we have that there are at least $(|S'_u| - 1)|S'_v| = \tilde{\Omega}(n^{1.16})$ potential edges that could go in any of the $m/2 - 2n - o(n^{.8})$ currently open boxes with index $\geq m/2$. The number of such edges dominates a binomially distributed random variable $X \simeq Bin(\left(\Omega(n), \tilde{\Omega}\left(\frac{n^{1.16}}{\binom{n}{2}}\right)\right)$, which has a mean of $\tilde{\Omega}(n^{.16})$ and thus is positive with probability $1 - \exp^{-\Omega(n^{.15})}$. This edge going between the fronts will guarantee that an augmenting path is found by inspecting either one of the augmenting trees. Thus the probability that we fail in the ith round is at most $\exp^{-\Omega(n^{.15})}$.

Since we repeat this for $\tilde{O}(n^{.2})$ rounds the probability of failure is $O(n^{-a})$ for any constant $a > 0$. ∎

5 Conclusion

We have shown that a maximum matching can be found in $O(n)$ expected time if the average degree is a sufficiently large constant. It is easy to extend this to

the case where the average degree grows with n. It is much more challenging to try to extend the result to any constant c. Karp and Sipser showed that if $c < e$ then **whp** Phase 1 leaves $o(n)$ vertices for Phase 2. In the paper [1], it was shown that for $c < e$, only a few vertex disjoint cycles are left, **whp**. So the problematical range is $e \leq c < c_0$.

References

1. Aronson, J., Frieze, A.M., Pittel, B.: Maximum matchings in sparse random graphs: Karp-Sipser revisited. Random Structures and Algorithms 12, 111–177 (1998)
2. Bast, H., Mehlhorn, K., Schäfer, G., Tamaki, H.: Matching Algorithms are Fast in Sparse Random Graphs. Theory of Computing Systems 39, 3–14 (2006)
3. Edmonds, J.: Paths, Trees and Flowers. Canadian Journal of Mathematics 17, 449–467 (1965)
4. Frieze, A.M., Pittel, B.: Perfect matchings in random graphs with prescribed minimal degree. In: Trends in Mathematics, pp. 95–132. Birkhauser Verlag, Basel (2004)
5. Karp, R.M., Sipser, M.: Maximum Matchings in Sparse Random Graphs. In: Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science, pp. 364–375 (1981)
6. Micali, S., Vazirani, V.V.: An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs. In: Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, pp. 17–27 (1980)
7. Motwani, R.: Average-case Analysis of Algorithms for Matchings and Related Problems. Journal of the ACM 41, 1329–1356 (1994)

Function Evaluation Via Linear Programming in the Priced Information Model

Ferdinando Cicalese*,[1] and Eduardo Sany Laber[2]

[1] AG Genominformatik, Technical Faculty, Bielefeld University, Germany
[2] Departamento de Informática, PUC – Rio de Janeiro, Brazil

Abstract. We determine the complexity of evaluating monotone Boolean functions in a variant of the decision tree model introduced in [Charikar *et al.* 2002]. In this model, reading different variables can incur different costs, and competitive analysis is employed to evaluate the performance of the algorithms. It is known that for a monotone Boolean function f, the size of the largest certificate, aka $PROOF(f)$, is a lower bound for $\gamma(f)$, the best possible competitiveness achievable by an algorithm on f. This bound has been proved to be achievable for some subclasses of the monotone Boolean functions, e.g., read once formulae, threshold trees. However, determining $\gamma(f)$ for an arbitrary monotone Boolean function has so far remained a major open question, with the best known upper bound being essentially a factor of 2 away from the above lower bound.

We close the gap and prove that *for any monotone Boolean function* f, $\gamma(f) = PROOF(f)$. In fact, we prove that $\gamma(f) \leq PROOF(f)$ holds for a class much larger than the set of monotone Boolean functions. This class also contains all Boolean functions.

1 Introduction

The decision tree is perhaps the simplest model of computation for studying the complexity of Boolean functions. In the classical variant of this model, a Boolean function is to be evaluated by querying about the initially unknown input. Each query asks the value of a single variable and it is charged a unitary cost. No other computational cost is taken into account. Then, the decision tree complexity $DC(f)$ of a function f is defined as the number of queries that an optimal algorithm for evaluating f needs to make on the most costly input. This is, in fact, the depth of the tree induced by the optimal algorithm. Clearly, $DC(f) \leq n$ for all functions on n variables. Conversely, a large part of the Boolean functions are known to be *evasive*, i.e., to satisfy $DC(f) = n$ [18]. In a certain way, this shows that the model is "too simple" to characterize the complexity of large classes of functions in an algorithmically meaningful way: in fact, for each evasive function, any algorithm has the same "optimal" behavior. There are basically two ways out of here: to restrict the classes of functions

* Supported by the Sofja Kovalevskaja Award 2004 of the Alexander von Humboldt Foundation and the Bundesministerium für Bildung und Forschung.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 173–185, 2008.

one focuses on or to slightly change the model in a way that allows more discriminative power. Both ways have been considered in the literature. Several subclasses of Boolean functions have been studied in the basic and in other variants of the decision tree model by, e.g., considering statistical knowledge on the input together with average case analysis, or by allowing randomization (see, e.g., [21,20,19] and references quoted therein). However, only very few non-trivial classes of functions have been completely characterized so far [19,12]. Conversely, for many classes studied, the gap between the best known bounds is still significant (see, e.g., [13,19], also, [2,10] include comprehensive surveys of the known results).

In this paper, we follow the more recent approach proposed by Charikar *et al.* in [3]. We assume that different variables can incur different reading costs and we employ competitive analysis to measure the performance of the evaluation algorithms. In particular, we study the following model.

Problem Statement. A function f over a set of variables $V = \{x_1, x_2, \ldots, x_n\}$ has to be evaluated for a fixed but unknown *assignment* σ, i.e., a choice of the values for the variables of V. Each variable x_i has an associated non-negative cost $c(x_i)$ which is the cost incurred to probe x_i, i.e., to read its value $x_i(\sigma)$. For each $i = 1, \ldots, n$, the cost $c(x_i)$ is fixed and known beforehand. The goal is to *adaptively* identify and probe a minimum cost set of variables $U \subseteq V$ whose values uniquely determine the value of f for the given assignment, regardless of the values of the variables not probed. The cost $c(U)$ of U is the sum of the costs of the variables it contains, i.e., $c(U) = \sum_{x \in U} c(x)$. We use $f(\sigma)$ to denote the value of f w.r.t. σ, i.e., $f(\sigma) = f(x_1(\sigma), \ldots, x_n(\sigma))$.

A set of variables $U \subseteq V$ is a *proof* for f with respect to a given assignment σ for the variables of V if the value $f(\sigma)$ is determined by the values that σ assigns to the variables of U regardless of the values assigned to the other variables.

An evaluation algorithm $\mathcal{A}$ for f is a rule to adaptively read the variables in V until the set of variables read so far is a proof for the value of f. The cost of algorithm $\mathcal{A}$ for an assignment σ is the total cost incurred by $\mathcal{A}$ to evaluate f under the assignment σ. Given a cost function $c(\cdot)$, we let $c^f_{\mathcal{A}}(\sigma)$ denote the cost of the algorithm $\mathcal{A}$ for an assignment σ and $c^f(\sigma)$ the cost of the cheapest proof for f under the assignment σ. We say that $\mathcal{A}$ is ρ-competitive if $c^f_{\mathcal{A}}(\sigma) \leq \rho c^f(\sigma)$, for every possible assignment σ. We use $\gamma^{\mathcal{A}}_c(f)$ to denote the competitive ratio of $\mathcal{A}$, that is, the minimum ρ for which $\mathcal{A}$ is ρ-competitive. The best possible competitive ratio for any deterministic algorithm, then, is $\gamma^f_c = \min_{\mathcal{A}} \gamma^{\mathcal{A}}_c(f)$, where the minimum is computed over all possible deterministic algorithms $\mathcal{A}$.

With the aim of evaluating the dependence of the competitive ratio on the structure of f, one defines the extremal competitive ratio $\gamma^{\mathcal{A}}(f)$ of an algorithm $\mathcal{A}$ as $\gamma^{\mathcal{A}}(f) = \max_c \gamma^{\mathcal{A}}_c(f)$. The best possible extremal competitive ratio for any deterministic algorithm, then, is $\gamma(f) = \min_{\mathcal{A}} \gamma^{\mathcal{A}}(f)$. This last measure is meant to capture the structural complexity of f independent of a particular cost assignment and algorithm. For instance, consider the Boolean function

$$f = (x_1 \text{ AND } x_2) \text{ OR } (x_2 \text{ AND } x_3) \text{ OR } (x_3 \text{ AND } x_4) \tag{1}$$

together with the costs $c(x_1) = 3, c(x_2) = 5, c(x_3) = 4$ and $c(x_4) = 1$. For the assignment $\sigma_R = (1, 0, 1, 1)$, we have $f(\sigma_R) = 1$ and $U = \{x_3, x_4\}$ as the only proof of minimum cost. Therefore, $c^f(\sigma_R) = 4 + 1$. On the other hand, for the assignment $\sigma_S = (1, 0, 0, 0)$, we have $f(\sigma_S) = 0$ and the cheapest proof is $\{x_2, x_4\}$. Thus, $c^f(\sigma_S) = 5 + 1$. Let now $\mathcal{A}$ be an algorithm that reads first x_1, then x_2, and so on, just skipping a variable x_i if, due to the values read so far, the value of x_i cannot affect the value of f. Thus, it is not hard to verify that $c^f_{\mathcal{A}}(\sigma_R) = 13$, since $\mathcal{A}$ reads the variables x_1, x_2, x_3, x_4. Furthermore, $c^f_{\mathcal{A}}(\sigma_S) = 12$, since in this case, $\mathcal{A}$ reads x_1, x_2 and x_3.

Beside its important theoretical aspects, function evaluation problems arise in several domains of computer science. In automatic diagnosis problems, a given system has to be checked by performing some tests whose costs can be different. This typically means evaluating some (Boolean) function and looking for the cheapest testing procedure (see, e.g., [1] and references therein). Applications are found in a variety of fields, e.g., telecommunications [16], manufacturing [7], computer networks [8], satisficing search problems [10], computer aided medical systems [17]. Also, the function evaluation problem arises in query optimization, a major issue in databases [14].

Related Work. The extremal competitive ratio was proposed by Charikar *et al.* in [3]. In the same paper, they present bounds on the extremal competitive ratio for monotone Boolean functions and some non-Boolean functions as the function "minimum of a list" and the function "searching a sorted list". Most of these bounds were improved in subsequent papers [11,4,5]. For some results regarding the measure γ^f_c see also [3,6].

For monotone Boolean functions, Charikar *et al.* proved that for every function f representable by an AND/OR tree it holds $\gamma(f) = PROOF(f)$, where $PROOF(f)$ is the size of the largest minimal proof for f. In addition, they mention that the inequality $\gamma(f) \leq 2 \times PROOF(f)$ holds for every monotone Boolean function f. This follows from the existence of an exponential time algorithm that achieves this bound. In [4], the authors observed that $\gamma(f) \geq PROOF(f)$ holds for any monotone Boolean function and a polytime algorithm[1] was presented showing that $\gamma(f)$ is slightly smaller than $2 \times PROOF(f)$. In [5], $\gamma(f) = PROOF(f)$ is proved for the classes of threshold tree functions (a class that includes AND/OR trees) and monotone Boolean functions for which no variable appears in more than 3 minterms. Determining whether $\gamma(f) = PROOF(f)$ holds for every monotone Boolean function f has so far remained a major open question.

Our Results. We prove that $\gamma(f) = PROOF(f)$ for all monotone Boolean functions. Actually, our main result is much more general: we prove $\gamma(f) \leq PROOF(f)$ for any non constant function $f : S_1 \times \cdots \times S_n \to \mathbb{R}$ whose domain is given by the Cartesian product of the domains of the individual variables. The class of such functions, which we denote by $\mathcal{F}^{\times}$, contains the whole class

[1] The time bound assumes that, given a σ, the algorithm can obtain $f(\sigma)$ in polytime.

of (total) Boolean functions. This seems a rather interesting result, considering how few assumptions are made on the structure of the functions in question.

We achieve the above bound by employing an LP-based approach ($\mathcal{LPA}$) to the design of algorithms for the function evaluation problem. The $\mathcal{LPA}$ is based on the solution of a linear program defined on the set of the minimal proofs of the function under consideration. The same linear program had been used for a variant of the problem in which the costs are unknown beforehand [6]. The optimal solution of this LP is used to capture the impacts of the variables of f in the process of evaluating f. The competitiveness of an algorithm obtainable by this approach is directly provided by the cost of the optimal solution for the linear program. Through an elegant geometric argument we are able to prove that this cost is not larger than $PROOF(f)$ for any $f \in \mathcal{F}^{\times}$.

Finding the optimal solution for our linear program can be a hard problem due to its potentially huge number of constraints. Therefore, when, in practical applications, also the cost of the computation, other than just the cost of probing the variables, is to be taken into account, suboptimal feasible solutions that are easy to compute might be preferable. We point out that for AND/OR trees and threshold trees, subclasses of monotone Boolean functions which have been studied in the literature, polynomial time algorithms with competitive ratio $\gamma(f)$ can be obtained through our linear programming based approach. In addition, when monotone Boolean functions are represented by their list of minterms (minimal proofs that guarantee that the function evaluates to 1) we can use our approach to isolate, in polynomial time, up to n strategies such that at least one of them has competitive ratio $\gamma(f)$. These results are presented in Section 3.1

Though we focus mainly on monotone Boolean functions, our linear programming approach has much broader applicability since it does not rely on any structure of the (class of) function(s) under consideration. We discuss how to employ it for other basic problems studied under the priced information framework, such as finding the minimum, sorting and searching [3,11,15]. We show that $\mathcal{LPA}$ allows us to devise algorithms with the best known competitive ratio for finding the minimum and for sorting.

We remark that the $\mathcal{LPA}$ extends the *General Approach* introduced in [4]. In fact, this can be considered a restriction of the $\mathcal{LPA}$, in which the impact of a variable takes values in $\{0, 1/p\}$ for some (implementation dependent) integer $0 < p \leq n$. Due to this constraint, the *General Approach* suffers from some intrinsic limitations [4] that are not inherited by the new methodology we present here. As an example, our main result, $\gamma(f) = PROOF(f)$ for monotone Boolean function f, cannot be proved with the *General Approach*.

2 Preliminaries

Let f be a function over a set of variables $V = \{x_1, x_2, \ldots, x_n\}$. In this paper we shall always assume that V does not contain redundant variables, i.e., the value of f depends upon the value of all the variables in V.

Let $Y \subseteq V$ and let σ be an assignment for the variables of Y. We use $f_{Y,\sigma}$ to denote the function over $V \setminus Y$ obtained from f by fixing the values of the variables in Y as given by σ. Consider, e.g., the function f in (1). Let $Y = \{x_2, x_4\}$ and $\sigma = (x_2 = 1, x_4 = 1)$. Then, we have $f_{Y,\sigma} = x_1$ OR x_3. In general, throughout this paper, Y will denote the set of variables read so far by the algorithm that evaluates f, and σ will be the assignment given by the values obtained when the variables of Y are read. We will usually write f_Y instead of $f_{Y,\sigma}$ whenever the assignment σ is clear from the context.

Given an assignment σ for the variables of V and a set of variables $Y \subseteq V$, we use σ_Y to denote the assignment σ restricted to the variables of Y, i.e., σ_Y is the assignment for the variables of Y satisfying $x(\sigma_Y) = x(\sigma)$ for every $x \in Y$.

Let v be a value in the range of f. A *v-witness* for f is a set of variables $C \subseteq V$ such that there is an assignment σ, with $f(\sigma) = v$, for which C is a proof for f with respect to σ.

We say that $C \subseteq V$ is a *v-certificate* for f if C is a v-witness for f and it is minimal, i.e., for any $x \in C$, the set $C \setminus \{x\}$ is not a v-witness for f. More generally, we say that a set $C \in V$ is a *certificate* for f if there exists a v such that C is a v-certificate for f. Note that for all assignments σ, every proof for f with respect to σ contains a certificate. Moreover, we have that $PROOF(f)$ is the size of the largest certificate for f.

Recall that $\mathcal{F}^{\times}$ denotes the class of functions whose domain is given by the Cartesian product of the domains of the single variables. In other words, the class $\mathcal{F}^{\times}$ is the set of functions whose variables' values can be chosen independently of each other. For instance, a function $g : \{(0,1),(1,0),(1,1)\} \to \{0,1\}$, is not in $\mathcal{F}^{\times}$. In fact, for g, both variables have individually domain $\{0,1\}$ but the function is not defined in $(0,0)$. In the last section, we will cope with some special functions not in $\mathcal{F}^{\times}$.

Proposition 1. *Let f be a non-constant function in $\mathcal{F}^{\times}$. Let u and v be distinct values in the range of f and $C, D \subseteq V$ be a u-witness and v-witness for f respectively. Then, $C \cap D \neq \emptyset$.*

The above proposition holds because otherwise we could construct an assignment σ for f such that $f(\sigma) = v$ and $f(\sigma) = u$.

Proposition 2. *Let V be the set of variables of a function f. Then, for every $Y \subset V$ and for every assignment σ for the variables of Y, we have $PROOF(f_{Y,\sigma}) \leq PROOF(f)$.*

Monotone Boolean functions. In this paper, by a Boolean function, we shall classically understand a total Boolean function, i.e., such that, the function is defined over the complete domain $\{0,1\}^n$. A Boolean function f over the set of variables $V = \{x_1, \ldots, x_n\}$ is *monotone* (increasing) iff $f(\sigma) \leq f(\sigma')$, for each pair of assignments σ and σ' such that $x_i(\sigma) \leq x_i(\sigma')$, for $i = 1, \ldots, n$.

For monotone Boolean functions, 0-certificates are called *maxterms* and 1-certificates are called *minterm*. As an example, in the function presented in (1), $\{x_1, x_2\}$ is a minterm and $\{x_2, x_4\}$ is a maxterm. We use $k(f)$ and $l(f)$ to denote

the size of the largest minterm and the largest maxterm of a monotone Boolean function f respectively. Then, $PROOF(f) = \max\{k(f), l(f)\}$.

The following result was first proved in [3] for the class of AND/OR trees and generalized to arbitrary monotone Boolean functions in [4].

Theorem 1. *If f is a monotone Boolean function then $\gamma(f) \geq \max\{k(f), l(f)\}$.*

3 The Linear Programming Approach

We shall now describe our general schema for the design of algorithms for evaluating functions. We call this methodology the *Linear Programming Approach* ($\mathcal{LPA}$). As suggested by the name itself, this schema is based on the solution of a linear program defined on the variables of the function under consideration and constrained on its certificates.

The linear program $\mathbf{LP_f}$ (see below) is used in the $\mathcal{LPA}$ to estimate how important a variable is in the process of evaluating f, that is, its *impact*. It tries to capture the intuitive idea that the relevance of a variable is proportional to the number of certificates it appears in and inversely proportional to the size of these certificates (small certificates tend to include variables with higher impact).

The linear programming approach consists of reading a variable that minimizes the ratio between its evaluation cost and its impact as estimated by the solution available for the linear program $\mathbf{LP_f}$. The cost function is then updated (scaled) in order to charge to every potential proof a fraction of the cost spent by the method. The procedure is then recursively applied on the new instance obtained by fixing the value of the variable just read and using the scaled cost.

The Linear Programming Approach. Let f be the function to evaluate and V its set of variables. Let $\mathcal{P} = \{P \subseteq V \mid P \text{ is a certificate for } f\}$. We define the following linear program $\mathbf{LP_f}$ where we have one non-negative real variable $s(x)$ for each variable $x \in V$ and one constraint for each certificate $P \in \mathcal{P}$.

$$\mathbf{LP_f} : \left\{ \text{Minimize } \sum_{x \in V} s(x) : \sum_{x \in P} s(x) \geq 1, \forall P \in \mathcal{P} \text{ and } s(x) \geq 0, \forall x \in V \right\}$$

The procedure below formalizes the linear programming approach. An *implementation* of this meta-algorithm is then obtained by fixing the rule used to choose at each iteration the feasible solution of $\mathbf{LP_{f_Y}}$, where Y is the set of variables already probed.

$\mathcal{LPA}(f, V, c)$
$Y \leftarrow \emptyset$;
While the value of f is unknown
  Let s_Y be a feasible solution for $\mathbf{LP_{f_Y}}$.
  Let u be the unread variable x that minimizes $\frac{c(x)}{s_Y(x)}$
  Read(u)
  For each $v \in V \setminus Y$ **do** $c(v) \leftarrow c(v) - s_Y(v) \times \frac{c(u)}{s_Y(u)}$
  $Y = Y \cup \{u\}$
End While
Return the value of f

We shall now present a lemma that is a key tool for the analysis of the implementations of $\mathcal{LPA}$. More precisely, this lemma allows to straightforwardly give an upper bound on the competitiveness of an implementation of the linear programming approach in terms of the feasible solution selected for the linear program. Therefore, in the different implementations presented, we shall simply verify the feasibility of the solution used for the linear program $\mathbf{LP_f}$ and provide a bound on the corresponding objective function. Then, we shall employ this lemma to state the competitiveness of the resulting algorithm.

Lemma 1. *Let $\mathbb{LP}$ be an implementation of $\mathcal{LPA}$ and, for each $Y \subset V$ let $s_Y(\cdot)$ be the feasible solution used by $\mathbb{LP}$ when the set of variables already read is Y. Then, $\gamma^{\mathbb{LP}}(f) \leq \max_{Y \subset V} \left\{ \sum_{x \in V \setminus Y} s_Y(x) \right\}$*

Proof. If f has only one variable the result holds. We assume as induction hypothesis that the result holds for every function that depends on less than n variables. Let f be a function that depends on n variables and let $c(\cdot)$ be a cost function such that $\gamma_c^{\mathbb{LP}}(f) = \gamma^{\mathbb{LP}}(f)$. Furthermore, let σ be an assignment for f which maximizes $c^f_{\mathbb{LP}}(\sigma)/c^f(\sigma)$. Let u be the first variable selected by $\mathbb{LP}$. Let us denote $s_\emptyset(\cdot)$ with $s(\cdot)$. Then,

$$c^f_{\mathbb{LP}}(\sigma) \leq \sum_{v \in V} s(v) \left(\frac{c(u)}{s(u)} \right) + \tilde{c}^{f_{\{u\}}}_{\mathbb{LP}}(\sigma_{V \setminus \{u\}}) = \frac{c(u)}{s(u)} \sum_{v \in V} s(v) + \tilde{c}^{f_{\{u\}}}_{\mathbb{LP}}(\sigma_{V \setminus \{u\}}), \quad (2)$$

where $\tilde{c}$ denotes the new cost function after that the costs of the variables in V have been decreased in the **For** loop of the $\mathcal{LPA}$ pseudo-code.

Let X be the cheapest proof for f w.r.t. cost function $c(\cdot)$ and assignment σ. Moreover, let X' be the cheapest proof for $f_{\{u\}}$ w.r.t. cost function $\tilde{c}$ and assignment $\sigma_{V \setminus \{u\}}$. Note that $X \setminus \{u\}$ is also a proof for $f_{\{u\}}$ w.r.t. assignment $\sigma_{V \setminus \{u\}}$. By the definition of the linear program $\mathbf{LP_f}$ we have $\sum_{v \in X} s(v) \geq 1$. Then,

$$c(X) = \sum_{v \in X} s(v) \left(\frac{c(u)}{s(u)} \right) + \tilde{c}(X \setminus \{u\}) \geq \frac{c(u)}{s(u)} \sum_{v \in X} s(v) + \tilde{c}(X') \geq \frac{c(u)}{s(u)} + \tilde{c}(X'). \quad (3)$$

Putting together (2) and (3) and noting that $\tilde{c}^{f_{\{u\}}}_{\mathbb{LP}}(\sigma_{V \setminus \{u\}})/\tilde{c}(X') \leq \gamma^{\mathbb{LP}}(f_{\{u\}})$, we have that $\gamma^{\mathbb{LP}}(f) = \gamma_c^{\mathbb{LP}}(f) = \dfrac{c^f_{\mathbb{LP}}(\sigma)}{c(X)} \leq \dfrac{\frac{c(u)}{s(u)} \sum_{v \in V} s(v) + \tilde{c}^{f_{\{u\}}}_{\mathbb{LP}}(\sigma_{V \setminus \{u\}})}{\frac{c(u)}{s(u)} + \tilde{c}(X')} \leq$

$\leq \max \left\{ \sum_{v \in V} s(v), \gamma^{\mathbb{LP}}(f_{\{u\}}) \right\}$. Since $f_{\{u\}}$ depends on less than n variables, the induction hypothesis yields

$$\gamma^{\mathbb{LP}}(f) \leq \max \left\{ \sum_{v \in V} s(v), \max_{Y \subset V \setminus \{u\}} \sum_{x \in V \setminus Y} s_Y(x) \right\} \leq \max_{Y \subset V} \sum_{v \in V \setminus Y} s_Y(v).$$

Note that the previous lemma does not rely on any assumption on the structure of f. It also motivates the following definition.

Definition 1. *The* fractional cover number *of a function f is defined by* $\Delta(f) = \max_{Y \subset V} \left\{ \sum_{x \in V \setminus Y} s^*_Y(x) \right\}$, *where $s^*_Y(\cdot)$ denotes the optimal solution of* $\mathbf{LP_{f_Y}}$.

Using this definition, Lemma 1 states that for every function f, we have $\gamma(f) \leq \Delta(f)$. We shall now prove for any function $f \in \mathcal{F}^{\times}$ an upper bound on the fractional cover number of f in terms of the size of its largest proof.

For a point $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$, we use $\|\mathbf{v}\|_p$ to denote the ℓ_p-norm of $\mathbf{v}$, i.e., $\|\mathbf{v}\|_p = \sqrt[p]{\sum_{i=1}^n v_i^p}$. Given two points $\mathbf{p}, \mathbf{q} \in \mathbb{R}^n$ we shall denote with $\mathbf{p} \cdot \mathbf{q}$ their dot product, i.e., $\mathbf{p} \cdot \mathbf{q} = \sum_{j=1}^n p_i q_i$.

We shall need the following simple technical result.

Proposition 3. *Let $S \subset \mathbb{R}^n$ be a convex set and let $\mathbf{u}$ be a point in $\mathbb{R}^n$. In addition, let $\mathbf{v}$ be a point in S closest to $\mathbf{u}$ in ℓ_2-norm. Then, $\|\mathbf{u}-\mathbf{w}\|_2 \geq \|\mathbf{v}-\mathbf{w}\|_2$ for all $\mathbf{w} \in S$.*

Lemma 2. *Let $f \in \mathcal{F}^{\times}$ be a non-constant function. Then, $\Delta(f) \leq PROOF(f)$.*

Proof. We shall show that for any function $f \in \mathcal{F}^{\times}$ there exists a feasible solution $\mathbf{s}$ for the linear program $\mathbf{LP_f}$ that has ℓ_1-norm not larger than $PROOF(f)$. We shall use the following geometric construction. For each value v in the range of f we consider the convex hull of the characteristic vectors of the v-certificates of f. We take the point $\mathbf{p}$ with the smallest ℓ_2-norm in the union of these convex hulls. Let $\mathbf{p}$ be in the convex hull of the v-certificates and take $w \neq v$ in the range of f. We prove that the desired $\mathbf{s}$ is given by the closest point (in ℓ_2-norm) to $\mathbf{p}$ among the points in the convex hull of the w-certificates of f.

Let Q be the range of f. Note that $|Q| \geq 2$, because f is not constant. For each $v \in Q$, let $\mathcal{P}_v$ denote the set of v-certificates for f. Thus, $\mathcal{P} = \bigcup_{v \in Q} \mathcal{P}_v$.

For each $P \in \mathcal{P}$, let $\mathbf{p}^P = (p_1^P, \ldots, p_n^P) \in [0,1]^n$ be defined by $p_i^P = 1$ if $x_i \in P$, and $p_i^P = 0$, otherwise.[2] Abusing notation, let us denote with $conv(v)$ the convex hull of the set $\{\mathbf{p}^P \mid P \in \mathcal{P}_v\}$.

Claim. Let $v, v' \in Q$ with $v \neq v'$. Then, $\mathbf{y} \cdot \mathbf{p}^{P'} \geq 1$ holds for each $\mathbf{y} \in conv(v)$ and for each proof $P' \in \mathcal{P}_{v'}$.

Proof of the Claim For each pair of proofs $P \in \mathcal{P}_v, P' \in \mathcal{P}_{v'}$, Proposition 1 assures that $P \cap P' \neq \emptyset$. Thus, $\mathbf{p}^P \cdot \mathbf{p}^{P'} \geq 1$.

Since $\mathbf{y} \in conv(v)$ we have that $\mathbf{y} = \sum_{P \in \mathcal{P}_v} \lambda_P \mathbf{p}^P$, where $\sum_{P \in \mathcal{P}_v} \lambda_P = 1$ and $\lambda_P \geq 0$, for each $P \in \mathcal{P}_v$. Thus, we have $\mathbf{y} \cdot \mathbf{p}^{P'} = \sum_{P \in \mathcal{P}_v} \lambda_P \mathbf{p}^P \cdot \mathbf{p}^{P'} \geq 1$. The proof of the claim is complete.

Now, let us rewrite $\mathbf{LP_f}$ in the following equivalent way.

$$\mathbf{LP_f} : \left\{ \text{Minimize } \|\mathbf{s}\|_1 : \mathbf{s} \cdot \mathbf{p}^P \geq 1, \text{ for every } P \in \bigcup_v \mathcal{P}_v \text{ and } \mathbf{s} \geq \mathbf{0} \right\}$$

[2] We look at the certificates of $\mathcal{P}_v$ as vectors in $\mathbb{R}^n$ with 0 and 1 coordinates.

Let $\mathbf{z}$ be a point with minimum ℓ_2 norm among the points in $\bigcup_{v\in Q} conv(v)$, i.e., $\|\mathbf{z}\|_2 \leq \|\mathbf{y}\|_2$, for each $\mathbf{y} \in \bigcup_{v\in Q} conv(v)$. Let v be such that $\mathbf{z} \in conv(v)$. In addition, let v^* be an arbitrarily chosen element in $Q - \{v\}$. Let $\mathbf{z}^*$ be the point in $conv(v^*)$ closest to $\mathbf{z}$ in the ℓ_2-norm, i.e., $\|\mathbf{z} - \mathbf{z}^*\|_2 \leq \|\mathbf{z} - \mathbf{y}^*\|_2$, for any $\mathbf{y}^* \in conv(v^*)$. We shall prove that $\mathbf{z}^*$ is a feasible solution for $\mathbf{LP_f}$ and $\|\mathbf{z}^*\|_1 \leq PROOF(f)$.

For the latter, it is enough to observe that $\mathbf{z}^* \in conv(v^*)$ implies that $\mathbf{z}^* = \sum_{P\in\mathcal{P}_{v^*}} \lambda_P \mathbf{p}^P$, for some non-negative scalars λ_P such that $\sum_{P\in\mathcal{P}_{v^*}} \lambda_P = 1$. Thus, $\|\mathbf{z}^*\|_1 = \|\sum_{P\in\mathcal{P}_{v^*}} \lambda_P \mathbf{p}^P\|_1 = \sum_{P\in\mathcal{P}_{v^*}} \lambda_P |P| \leq PROOF(f)$.

We now prove the feasibility of $\mathbf{z}^*$. By the above claim, we immediately have that $\mathbf{z}^* \cdot \mathbf{p}^P \geq 1$ for any $P \in \mathcal{P}_u$, with $u \in Q$, $u \neq v^*$. It remains to prove that $\mathbf{z}^* \cdot \mathbf{p}^P \geq 1$, for each $P \in \mathcal{P}_{v^*}$.

By Proposition 3, we have $||\mathbf{z}^* - \mathbf{y}^*||_2 \leq ||\mathbf{z} - \mathbf{y}^*||_2$, for each $\mathbf{y}^* \in conv(v^*)$. In particular, for each $P \in \mathcal{P}_{v^*}$, we have

$$(||\mathbf{z}^* - \mathbf{p}^P||_2)^2 \leq (||\mathbf{z} - \mathbf{p}^P||_2)^2. \tag{4}$$

Recall that for any $\mathbf{p}$ and $\mathbf{q}$ it holds that $(\|\mathbf{p} - \mathbf{q}\|_2)^2 = (\|\mathbf{p}\|_2)^2 + (\|\mathbf{q}\|_2)^2 - 2\mathbf{p} \cdot \mathbf{q}$. Thus, (4) becomes

$$(||\mathbf{z}^*||_2)^2 - 2\mathbf{z}^* \cdot \mathbf{p}^P \leq (||\mathbf{z}||_2)^2 - 2\mathbf{z} \cdot \mathbf{p}^P. \tag{5}$$

By the choice of $\mathbf{z}$ we have that $||\mathbf{z}||_2 \leq ||\mathbf{z}^*||_2$, which together with (5) yields $\mathbf{z}^* \cdot \mathbf{p}^P \geq \mathbf{z} \cdot \mathbf{p}^P \geq 1$, where the last inequality follows by the above claim.

Summarizing, we have shown that for any function f satisfying the hypothesis, the optimal solution s^* of $\mathbf{LP_f}$, has cost not greater than $PROOF(f)$. In particular, for every restriction f_Y of f, denoting with s^*_Y the optimal solution of $\mathbf{LP_{f_Y}}$, we have $\|s^*_Y\|_1 \leq PROOF(f_Y) \leq PROOF(f)$. This concludes the proof.

Directly from Lemmas 1 and 2 and Theorem 1 we have the following results.

Theorem 2. *Let $f = f(x_1, x_2, \ldots, x_n)$ be a non-constant function in $\mathcal{F}^\times$. Then, $\gamma(f) \leq PROOF(f)$.*

Corollary 1. *For every monotone Boolean function $\gamma(f) = PROOF(f)$.*

Some remarks on the issue of efficiency. We have shown that an optimal implementation of the $\mathcal{LPA}$ can be used to obtain an algorithm for evaluating monotone Boolean functions with the optimal extremal competitive ratio. By an optimal implementation we mean one that always uses the optimal solution to the $\mathbf{LP_f}$. We shall now touch upon the issue of whether or when this can be done efficiently. Due to the space limitation, we shall defer a more formal treatment of the following material to the full version of the paper.

Let f be a monotone Boolean function over n variables. Clearly the existence of a polynomial time implementation of the $\mathcal{LPA}$ for f critically depends on the representation of f which is given. We shall here assume that f is given as the list of its minterms. Let A be the binary matrix whose rows are given by the characteristic vectors of the minterms of f, indexed over the set of the n variables

of f. Let B be the binary matrix whose rows are the characteristic vectors of the maxterms of f, i.e., the minimal hitting sets of the family of the minterms. An optimal implementation of the $\mathcal{LPA}$ has to find the vector of minimum ℓ_1-norm in the polyhedron $\mathcal{A} = \{\mathbf{p} \mid A\mathbf{p} \geq 1, B\mathbf{p} \geq 1, \mathbf{p} \geq 0\}$. We observe that the the number of maxterms can be exponential in the size of the representation of f. Furthermore, the separation problem of $\mathbf{LP_f}$ consists of solving the hitting set problem, a well known NP-complete problem. Therefore, for all practical purposes, rather than trying to directly solve $\mathbf{LP_f}$, it seems reasonable to look for "good" feasible solutions for $\mathbf{LP_f}$ that can be constructed in polynomial time. To this aim, we can try to construct another polyhedron $\mathcal{B}$ with the following properties: (i) it is contained in the polyhedron $\mathcal{A}$ and (ii) linear functions can be optimized over it in polynomial time. Clearly, the point of minimum ℓ_1-norm in $\mathcal{B}$ gives us a feasible solution for $\mathbf{LP_f}$ in polynomial time. We aim at having such minimum in $\mathcal{B}$ not much larger than the minimum in $\mathcal{A}$.

Let $\mathcal{M} = \{\mathbf{z} \in [0,1]^n \mid A\mathbf{z} \geq \mathbf{1}\}$ and define $\mathcal{B} = \{\mathbf{p} \mid \mathbf{p} \geq \mathbf{0}, A\mathbf{p} \geq \mathbf{1},\ \mathbf{p} \cdot \mathbf{z} \geq 1\, \forall \mathbf{z} \in \mathcal{M}\}$. Since every row $\mathbf{y}$ in the maxterm matrix B satisfies $A\mathbf{y} \geq 1$, we have that $\mathbf{y} \in \mathcal{M}$ and then $\mathcal{B} \subseteq \mathcal{A}$, as desired. Moreover, although $\mathcal{B}$ is defined by an infinite number of constraints (one for any $\mathbf{z} \in \mathcal{M}$) the corresponding separation problem can be solved in polynomial time. In fact, given a point $\mathbf{p} \in \mathbb{R}^n$, we can use the linear program $\mathbf{LP^B} : \{$ Minimize $\mathbf{p} \cdot \mathbf{z} : A\mathbf{z} \geq \mathbf{1},\ \mathbf{z} \geq 0\}$ to verify whether $\mathbf{p}$ is in $\mathcal{B}$ or not. It is easy to see that $\mathbf{p} \in \mathcal{B}$ if and only if the cost of the optimal solution to $\mathbf{LP^B}$ is not smaller than 1 and $A\mathbf{p} \geq 1$. In addition, if $\mathbf{u}$ ($\mathbf{v}$) is the characteristic vector of an arbitrary minterm (maxterm) of f then $\mathbf{z} = \mathbf{u} + \mathbf{v}$ belongs to $\mathcal{B}$ and $||\mathbf{z}||_1 \leq 2PROOF(f)$. By a more refined construction we can prove that there exists a point $\mathbf{p} \in \mathcal{B}$ such that $\|\mathbf{p}\|_1 \leq 2PROOF(f) - \sqrt{k(f), \ell(f)}$. This implies that an implementation of the $\mathcal{LPA}$ that as feasible solution of $\mathbf{LP_f}$ uses the point with minimum ℓ_1 norm of $\mathcal{B}$ has competitive ratio at most $2PROOF(f) - \sqrt{k(f), \ell(f)}$, and is polytime. This matches the competitiveness of the best known polytime algorithm for evaluating monotone Boolean functions[4]. We do not know if this bound is tight or not, that is, whether a precise estimation of the minimum ℓ_1-norm over all points in $\mathcal{B}$ could improve the factor of 2 in the competitive ratio.

Optimizing a linear function over the polyhedron $\mathcal{B}$ can be slow for practical applications since one needs to use the ellipsoid method. However, by using the theory of blocking polyhedra (see, e.g., [9, Ch. 30]), it is possible to prove that $\mathcal{B}$ is the projection into $\mathbb{R}^n$ of the polyhedron $\mathcal{C} = \{(\mathbf{y}, \mathbf{w}) \mid \mathbf{y} \in \mathbb{R}^n,\ \mathbf{w} \in \mathbb{R}^m_+,\ \|\mathbf{w}\|_1 = 1,\ A\mathbf{y} \geq \mathbf{1},\ \mathbf{y} \geq \mathbf{w}^T A\}$, defined in $\mathbb{R}^{m+n}$, where m is the number of minterms. Since $\mathcal{C}$ is defined by a polynomial (in fact linear) number of constraints, the point of minimum ℓ_1-norm in $\mathcal{B}$ can be quickly obtained by projecting in $\mathbb{R}^n$ the point of $\mathcal{C}$ which minimizes the sum of the first n coordinates.

We remark that for functions that have a compact circuit representation we can obtain a polynomial time algorithm if the separation problem for $\mathbf{LP_f}$ is solvable in polynomial time. This is the case, e.g., of previously studied classes of functions like threshold trees and, AND/OR trees (see, e.g., [3,4,5,6]).

The existence of a polytime algorithm with competitive ratio $PROOF(f)$ for evaluating monotone Boolean functions remains an open problem.

4 Final Remarks: Beyond Monotone Boolean Functions

In this section we shall give more evidence of the power of the $\mathcal{LPA}$ and its broad applicability. We shall first discuss the case of arbitrary Boolean functions and then outline how the $\mathcal{LPA}$ can be used to obtain efficient and highly competitive algorithms for the functions: finding the minimum and sorting.

Arbitrary Boolean functions. Since the class of boolean functions belong to $\mathcal{F}^{\times}$ it follows from Theorem 2 that $\gamma(f) \leq PROOF(f)$ for every boolean function f. Then, it it is natural to ask whether Corollary 1 holds without the monotonicity assumption, that is, whether $\gamma(f) = PROOF(f)$ for every boolean function f. We now give an example showing that $\gamma(f)$ can be smaller than $PROOF(f)$. Let, e.g., $f = (z \text{ AND } x_1) \text{ OR } (z \text{ AND } x_2) \text{ OR } (\bar{z} \text{ AND } x_3) \text{ OR } (\bar{z} \text{ AND } x_4)$. We have $PROOF(f) \geq 4$ since $\{x_1, x_2, x_3, x_4\}$ is a 0-certificate for the assignment $(x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, z = 0)$. A careful inspection shows that $\Delta(f) = 3$ which implies, by Lemma 1, that $\gamma(f) < PROOF(f)$.

We shall note however that $\gamma(f) = \Delta(f)$ for the above function. In fact, we do not know any example for which $\gamma(f) < \Delta(f)$. The above example can be generalized to show that that $\gamma(f)$ can be much smalller than $PROOF(f)$.

Sorting and Finding the minimum. The $\mathcal{LPA}$ can be implemented to provide very competitive solutions to the problems of "sorting" and "finding the minimum" in the context of function evaluation with costs. We want to compute the functions $f^{sort} \equiv sort(v_1, \ldots, v_n)$ and $f^{min} \equiv \min\{v_1, \ldots, v_n\}$ when $v_1, \ldots, v_n$ are variables taking values in some totally ordered set. As is customary, the performance of algorithms for such problems is in terms of the comparisons that it performs We shall assume that each pair of variables, v_i, v_j has an associated cost $c(v_i, v_j)$ to be paid to compare them. Different comparisons may incur different costs.

As model we use a complete weighted graph in which the set of vertices is given by the set of variables $S = \{v_1, \ldots, v_n\}$ and each edge represents the comparison between the incident variables. The weight of an edge is the cost incurred to execute the comparison it represents. We can view the outcome of the comparison as the operation of disclosing the orientation of the edge, assuming that the orientation goes from the smaller to the larger variable. A feasible assignment σ is a transitive orientation of the edges. In the case of f^{min}, a proof for the identification of the minimum w.r.t. a feasible assignment σ is a set of edges P such that, when oriented according to σ, there is a vertex v which reaches every other vertex in the graph $G_P = (S, P)$. For f^{sort} a minimal proof is a Hamiltonian path such that at least one of its two possible orientations is compatible with the assignment σ. The following summarizes our results.

Proposition 4. *Let* $f \in \{f^{sort}, f^{min}\}$. *Then,* $\Delta(f) = n - 1$.

This implies the bound $\gamma(f^{min}) \leq n-1$ previously given in [3,11]. The exact value $\gamma(f^{min}) = n-2$ was given in [4] by an *ad hoc* implementation of the General Approach. Moreover, by Proposition 4, it also follows that $\gamma^{\mathcal{LPA}}(f^{sort}) \leq n-1$ which improves (the constant of) the best known result for sorting [11]. The problem of determining the exact value of $\gamma(f^{sort})$ remains open.

Final open questions. We have shown that in the class of monotone Boolean functions and also in some special proper superclass of it we have $\gamma(f) = \Delta(f)$. On the other hand, we know that for some functions outside $\mathcal{F}^{\times}$ like f^{min} we can prove a strict inequality, in fact, $n-2 = \gamma(f^{min}) < \Delta(f^{min}) = n-1$. Another negative example is given by the function searching for an element in a sorted list, also considered in [3]. In this case, we have $O(\log n) \leq \gamma(f) << \Delta(f) = n$.

An interesting question is the following. "Does $\gamma(f) = \Delta(f)$ hold for each $f \in \mathcal{F}^{\times}$?" This class already contains all the (total) Boolean functions.

References

1. Boros, E., Ünlüyurt, T.: Diagnosing double regular systems. Annals of Mathematics and Artificial Intelligence 26(1-4), 171–191 (1999)
2. Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: a survey. Theoretical Computer Science 288(1), 21–43 (2002)
3. Charikar, M., Fagin, R., Guruswami, V., Kleinberg, J.M., Raghavan, P., Sahai, A.: Query strategies for priced information. Journal of Computer and System Sciences 64(4), 785–819 (2002)
4. Cicalese, F., Laber, E.S.: A new strategy for querying priced information. In: Proc. of STOC 2005, pp. 674–683. ACM Press, New York (2005)
5. Cicalese, F., Laber, E.S.: An Optimal Algorithm for Querying Priced Information: Monotone Boolean Functions and Game Trees. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 664–676. Springer, Heidelberg (2005)
6. Cicalese, F., Laber, E.S.: On the competitive ratio of evaluating priced functions. In: Proc. of SODA 2006, pp. 944–953 (2006)
7. Duffuaa, S.O., Raouf, A.: An optimal sequence in multicharacteristics inspection. J. Optim. Theory Appl. 67(1), 79–86 (1990)
8. Gillies, D.W.: Algorithms to schedule tasks with and/or precedence constraints. PhD thesis, Champaign, IL, USA (1993)
9. Graham, R.L., Grötschel, M., Lovász, L. (eds.): Handbook of Combinatorics. The Mit Press - North-Holland (1996)
10. Greiner, R., Hayward, R., Jankowska, M., Molloy, M.: Finding optimal satisficing strategies for and-or trees. Artificial Intelligence 170(1), 19–58 (2006)
11. Gupta, A., Kumar, A.: Sorting and selection with structured costs. In: Proc. of FOCS 2001, pp. 416–425. IEEE, Los Alamitos (2001)
12. Heiman, R., Newman, I., Wigderson, A.: On read-once threshold formulae and their randomized decision tree complexity. TCS 107(1), 63–76 (1993)
13. Heiman, R., Wigderson, A.: Randomized vs. deterministic decision tree complexity for read-once boolean functions. Comput. Complexity 1, 311–329 (1991)
14. Hellerstein, J.M.: Optimization techniques for queries with expensive methods. ACM Transactions on Database Systems 23(2), 113–157 (1998)
15. Kannan, S., Khanna, S.: Selection with monotone comparison cost. In: Proc. of SODA 2003, pp. 10–17. ACM/SIAM (2003)

16. Louis AnthonyCox, J., Qiu, Y., Kuehner, W.: Heuristic least-cost computation of discrete classification functions with uncertain argument values. Ann. Oper. Res. 21(1-4), 1–30 (1989)
17. Qiu, Y., Anthony Cox Jr., L., Davis, L.: Guess-and-verify heuristics for reducing uncertainties in expert classification systems. In: Dubois, D., Wellman, M.P. (eds.) UAI, pp. 252–258. Morgan Kaufmann, San Francisco (1992)
18. Rivest, R.L., Vuillemin, J.: On recognizing graph properties from adjacency matrices. TCS 3(3), 371–384 (1976)
19. Saks, M., Wigderson, A.: Probabilistic Boolean decision trees and the complexity of evaluating game trees. In: Proc. of FOCS 1986, pp. 29–38. IEEE, Los Alamitos (1986)
20. Snir, M.: Lower bounds on probabilistic linear decision trees. TCS 38, 69–82 (1985)
21. Tarsi, M.: Optimal search on some game trees. JACM 30(3), 389–396 (1983)

Improved Approximation Algorithms for Budgeted Allocations*

Yossi Azar[1], Benjamin Birnbaum[2], Anna R. Karlin[2], Claire Mathieu[3], and C. Thach Nguyen[2]

[1] Microsoft Research and Tel-Aviv University
azar@tau.ac.il
[2] University of Washington
{birnbaum,karlin,ncthach}@cs.washington.edu
[3] Brown University
claire@cs.brown.edu

Abstract. We provide a 3/2-approximation algorithm for an offline budgeted allocations problem with applications to sponsored search auctions. This an improvement over the $e/(e-1)$ approximation of Andelman and Mansour [1] and the $e/(e-1)-\epsilon$ approximation (for $\epsilon \approx 0.0001$) of Feige and Vondrak [2] for the more general Maximum Submodular Welfare (SMW) problem. For a special case of our problem, we improve this ratio to $\sqrt{2}$. We also show that the problem is APX-hard.

1 Introduction

The rising economic importance of online sponsored search advertising has led to a great deal of research focused on developing its theoretical underpinnings. (See e.g., [3] for a survey.) Since search engines such as Google, Yahoo! and MSN depend on sponsored search for a significant fraction of their revenue, a key problem is how to optimally allocate ads to keywords (user searches) so as to maximize search engine revenue [1,4,5,6,7]. In this direction, Mehta et al. [7] studied a stylized version of the problem, which we call the *Online Budgeted Allocation* problem. In their model, there is a set of bidders U and a set of keywords V. Each bidder $i \in U$ has a known daily budget B_i and a non-negative bid b_{ij} for every keyword $j \in V$. The keywords arrive one-by-one in an online fashion, with the bids for keyword j revealed only when j arrives. At each keyword arrival, the algorithm (i.e., the search engine) allocates the keyword to one of the bidders (i.e., displays that bidder's ad as one of the sponsored search results the user sees). The total profit extracted by the algorithm from each bidder is the minimum of the budget B_i of that bidder and the sum of the b_{ij}'s for keywords j allocated to it. The goal is to find an allocation of keywords to bidders that maximizes the total profit extracted by the algorithm. Mehta et al. [7] presented an algorithm that achieves an optimal competitive ratio of

* This research was supported by the Israeli Science Foundation, NSF Grant CCF-0635147, and by an NSF Graduate Research Fellowship.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 186–197, 2008.

$e/(e-1)$ for the case when the bids are much smaller than the budgets, a result also proved by Buchbinder et al. [4]. When there is no restriction on the values of the bids relative to the budgets, the best known competitive ratio is 2 [8].

Surprisingly, the approximability of the *offline* version of Budgeted Allocation, in which the algorithm can see all of the bids before allocating keywords, is still not well understood. Lehmann et al. [8] showed that the problem is NP-hard, and Andelman and Mansour [1] provided the first non-trivial approximation ratio of $e/(e-1)$. Feige and Vondrak [2] improved this ratio to $e/(e-1)-\epsilon$ (for $\epsilon \approx 0.0001$) for the more general SMW problem.

In this paper, we give improved approximation algorithms for two versions of the offline Budgeted Allocation problem. In the *uniform* version, each keyword j has a single price b_j. If a bidder i is interested in j, its bid b_{ij} is equal to b_j. Otherwise, its bid b_{ij} is 0. The *non-uniform* version removes this restriction, so that the b_{ij} values can be arbitrary for each i and j.

1.1 Our Results

We provide a deterministic 3/2-approximation algorithm for the non-uniform Budgeted Allocation problem (Section 2). This improves the previous best-known approximation ratio of $e/(e-1)-\epsilon$ (for $\epsilon \approx 0.0001$) [2]. For the uniform version of the problem, we improve the approximation ratio to $\sqrt{2}$ (Section 3). In both these algorithms, we assume that the maximum bid is no larger than the smallest budget, i.e. $\max_{i,j} b_{ij} \leq \min_i B_i$.

We also show that the problem is APX-hard (Section 4).

1.2 Related Work

As discussed above, before this work, the first non-trivial approximation ratio for Budgeted Allocation was $e/(e-1)$, due to Andelman and Mansour [1]. For the special case in which the bidders all have the same budget, these authors were able to lower this ratio to approximately 1.39. Our algorithms apply to the more general case in which the budgets may be different for different bidders.

Two recent unpublished works also study the Budgeted Allocation problem and provide better approximation ratios than those obtained in this paper: Independently of our work, Chakrabarty and Goel [9] have provided two elegant algorithms, an iterative rounding algorithm and a primal-dual algorithm, both of which achieve an approximation ratio of 4/3, matching the integrality gap of the linear program used in this and other papers. Their paper also shows that it is NP-hard to approximate Budgeted Allocation to a factor better than 16/15, which subsumes the result in this paper on APX-hardness. In addition, building on our approach, Srinivasan [10] has recently provided another LP-rounding algorithm that achieves an approximation ratio of 4/3.

The Budgeted Allocation problem is an important special case of SMW, the problem of maximizing utility in a combinatorial auction in which the utility functions are submodular. In the combinatorial auction setting the keywords are items to be sold. SMW has been widely studied [2,8,11,12,13,14]. For submodular

auctions using the value query model, the best approximation algorithm gives a factor $e/(e-1)$ [14]. This ratio has been shown to be the best possible for this model [12,13]. For the stronger demand query model it is possible to do at least slightly better, that is $e/(e-1)-\epsilon$ (for $\epsilon \approx 0.0001$) [2]. For solving the Budgeted Allocation problem using the SMW demand query model one needs to provide a polynomial-time demand query oracle. As noted by [15], one can use a knapsack-type FPTAS algorithm to provide an approximate oracle that is good enough for solving the problem.

The Budgeted Allocation problem is also similar to the *generalized assignment problem* (GAP) [2,15,16,17]. The main difference between Budgeted Allocation and GAP is that in GAP every keyword (or *item*, in GAP parlance) has a weight and each bidder (or *bin*) has a fixed capacity that cannot be exceeded, whereas in Budgeted Allocation, budgets can be exceeded, but no extra profit is obtained from doing so. The best approximation algorithm for GAP does slightly better than $e/(e-1)$ [2].

2 The 3/2-Approximation Algorithm

In this section, we describe the 3/2-approximation for the non-uniform version of the problem.

2.1 High-Level Idea

Our algorithms use linear program rounding. We represent the Budgeted Allocation problem with the same natural integer program used in [1,9,10], in which the 0-1 variables x_{ij} represent whether keyword j is allocated to bidder i:

$$\max \sum_{i \in U} \min(B_i, \sum_{j \in V} b_{ij} x_{ij}) \quad \text{s.t.} \quad \begin{cases} \sum_{i \in U} x_{ij} \leq 1 & \forall j \in V \\ x_{ij} \in \{0,1\} & \forall i \in U, j \in V \end{cases} .$$

Let L be the linear relaxation of this integer program in which the second constraint is replaced by $x \geq 0$. (The upper-bound of 1 is guaranteed by the other constraint.) Rounding the optimal solution carefully is what allows us to beat the factor of $e/(e-1)$ of Andelman and Mansour [1].

For any fractional allocation $\mathbf{x}$, define the graph G *induced by* $\mathbf{x}$ to be the bipartite graph over $U \cup V$ with an edge $\{i,j\}$ for every $x_{ij} > 0$, with weight $w_{ij} = b_{ij}x_{ij}$.[1] Rounding $\mathbf{x}$ can be viewed as a transformation of G into another graph in which the degree of every keyword is 1. Our algorithms do this iteratively, at each step modifying local structures so that the degree of at least one new keyword is reduced to 1. Some weight in the objective function will be lost at each step, but we use a charging argument to bound this loss by 1/3 of the original value of $\mathbf{x}$.

[1] Notice that in a fractional solution, we can assume without loss of generality that no bidder's budget is exceeded. Therefore, the value of $\mathbf{x}$ is equal to the sum of the edge weights in G.

The first observation for the proofs is that one can assume that the graph G induced by a feasible fractional solution to L has a special structure. This observation was made for optimal fractional solutions by [1], proved in [18], and used by [9,10]. We use a slightly stronger version that holds for any feasible solution. This will allow us to assume that this special structure holds after each rounding step, not just at the beginning. Say that bidder i is *saturated* if $\min(B_i, \sum_{j \in V} b_{ij} x_{ij}) = B_i$ and is *unsaturated* otherwise.

Lemma 1. *Any feasible solution* $\mathbf{x}$ *of* L *with induced graph* G *can be transformed, in polynomial time, to another feasible solution* $\tilde{\mathbf{x}}$ *that has an induced graph that is a subgraph of* G *and that is a forest with at most one unsaturated bidder per tree.*

Proof. The proof follows by standard arguments and is very similar to the proof in [18]. It can be found in the full version of this paper.[2]

For a graph G induced by a fractional solution, say that a bidder is *active* if it has at least one neighbor of degree 2 or more in G, and is *inactive* otherwise. In our charging argument for the 3/2-algorithm, we charge to the weight allocated to bidders when they move from being active to inactive. (Since this happens at most once for each bidder, each unit of profit in the optimal fractional solution is charged at most once.)

We call the process of transforming a subgraph to reduce the degrees of the keywords *rounding* the subgraph. In general, this process will remove some of the edges and transfer some of the weight removed to the edges that remain, while respecting the constraints of the LP. For example, suppose that two keywords j_1 and j_2 are allocated fractionally to bidder i and fractionally to some other bidders i_1 and i_2, as shown in Fig. 1. One way to round this part of the graph would be to remove the edges $\{i, j_1\}$ and $\{i_2, j_2\}$. Once this is done, i has some unused budget that can be used to transfer an additional fraction of j_2 from i_2 to i. In general, transferring weight in this manner will be essential to obtaining our approximation ratio.

The main idea of our proof is that in every tree with active bidders, there is a small local structure involving only a constant number of nodes, such that if we round that structure so as to minimize the resulting loss in the objective function, the loss is at most 1/3 of the budget spent by the bidders that become inactive.

2.2 The Algorithm

Our 3/2-approximation algorithm is given below by Algorithm 1. At each iteration of the while loop, our algorithm looks for one of the *interesting nodes* (Definition 4) and rounds keywords in the associated structure. For each of the four types of interesting nodes, we describe a rounding subroutine used by the

[2] The full version of this paper is available at http://www.cs.washington.edu/homes/birnbaum/budgetedallocation.pdf.

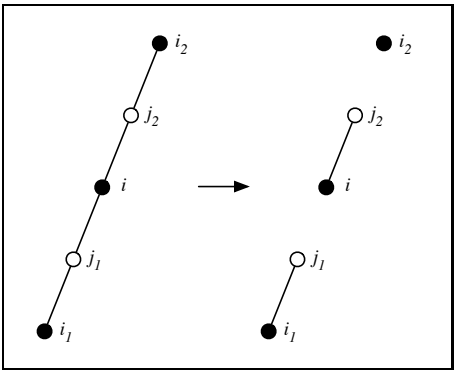


Fig. 1. One way to round a path that has three bidders. When edge $\{i, j_1\}$ is removed, this frees up some budget in i to accommodate some of the weight of j_2 that was originally allocated to i_2.

algorithm. We will show that the loss of these subroutines can be charged to the nodes that become inactive, which we will use to prove that Algorithm 1 is a 3/2-approximation.

Theorem 2. *Algorithm 1 is a polynomial-time* 3/2*-approximation for the Budgeted Allocation problem.*

Algorithm 1. 3/2-approximation algorithm for Budgeted Allocation

Input: Set of bidders U, set of keywords V; for each i, j, bid b_{ij} of bidder i for keyword j; and for each bidder i, budget B_i.
Output: Allocation of keywords to bidders.
solve the following LP to get an optimal solution $\mathbf{x}$ with induced graph G:

$$\max \sum_{i \in U} \min(B_i, \sum_{j \in V} b_{ij} x_{ij}) \quad \text{s.t.} \quad \begin{cases} \sum_{i \in U} x_{ij} \leq 1 & \forall j \in V \\ 0 \leq x_{ij} & \forall i \in U, j \in V \end{cases} .$$

transform G into a forest with ≤ 1 unsaturated bidder per tree (Lemma 1).
while G *contains active bidders* **do**
  Round the subgraph associated to an interesting node according to its type;
  Transform G into a forest with ≤ 1 unsaturated bidder per tree (Lemma 1);
end
allocate each keyword to its unique adjacent bidder in G.

Root each tree of G at the unsaturated bidder if there is one, or at an arbitrary bidder if there is not.

Definition 3. *Consider a path* $i_1, j_1, i_2, j_2, \ldots, i_{k-1}, j_{k-1}, i_k$ *consisting of* $2k-1$ *nodes, starting and ending with a bidder, such that*

– *the* $k-1$ *keywords* $j_1, j_2, \ldots, j_{k-1}$ *all have degree exactly 2,*

– *bidder i_1 is the highest node on the path, called "root" of the path,*
– *for all other bidders $i_2, i_3, \ldots, i_k$, any keyword not on the path that is adjacent to the bidder has degree exactly 1.*

We call the graph formed by this path and all the degree-1 keywords that are neighbors of $i_1, i_2, \ldots, i_k$ a k-chain.

Definition 4. *In a tree of G, we say that a node is* interesting *if it has one of the following four types.*

1. *The root of the tree, if the tree consists of a 2-chain (Fig. 2(a)).*
2. *A bidder v whose subtree contains at least one 3-chain rooted at v (Fig. 2(b)).*
3. *A bidder v who is the root of more than one 2-chain and who is not the root of a k-chain, for $k > 2$ (Fig. 2(c)).*
4. *A keyword with at least 2 children, such that each child is a root of a 1-chain or a 2-chain (Fig. 2(d)).*

Before we describe the rounding subroutines, we establish the correctness of Algorithm 1.

Lemma 5. *A tree that has a keyword of degree more than 1 must have at least one interesting node.*

Proof. A straightforward proof can be found in the full version of this paper.

Hence, in the forest produced by Algorithm 1, every keyword has degree 1, and the output is an integer allocation.

This lemma, along with the analysis of the rounding subroutines described below, will give us all of the ingredients we need to prove that Algorithm 1 is a 3/2-approximation.

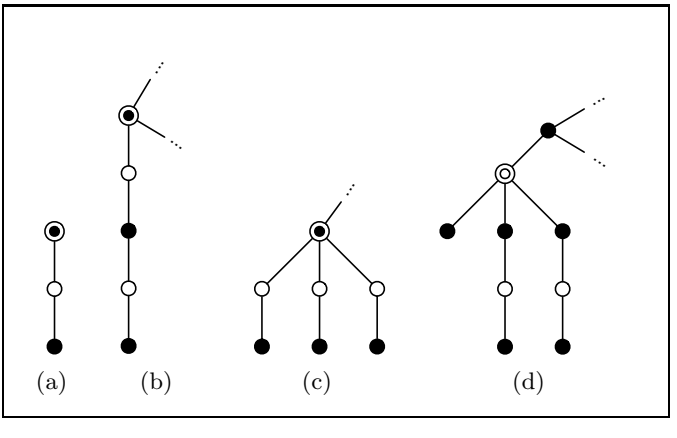


Fig. 2. Examples of the four types of interesting nodes with their associated subgraphs, as defined in Definition 4. For each type, the interesting node is shown with an extra circle.

Proof of Theorem 2. Lemma 5 proves that when Algorithm 1 terminates, it returns a graph in which each keyword has degree 1. Each rounding step described below clearly takes polynomial time and makes at least one new bidder become inactive. Hence, the running time of the algorithm is polynomial.

For each rounding, we will associate each unit lost to 1/3 of the weight spent by a bidder that becomes inactive. Since each bidder becomes inactive only once, the total weight lost must be no larger than 1/3 of the weight of the optimal fractional solution. Therefore, the algorithm returns a solution with weight at least 2/3 of the optimal fractional solution and hence with weight at least two thirds of the optimal integral solution. □

2.3 The Rounding Subroutines

To simplify the exposition, we assume, without loss of generality, that all of the bids and budgets have been scaled so that the maximum bid is 1 (and hence the minimum budget is at least 1).

Type 1 Rounding

Rounding. Let i be the interesting node, j be its child of degree 2, and k be its grandchild. By Lemma 1, bidder k is saturated, while bidder i may have some unused budget $s \geq 0$.

Consider two ways to round the 2-chain rooted at i. In the first way, we remove the edge $\{i, j\}$. In the second way, we remove the edge $\{k, j\}$ and transfer as much as possible of the removed weight to the edge $\{i, j\}$ while maintaining feasibility. Of those two ways, we choose the one that incurs the smaller loss in the objective function.

Analysis. Since this rounding makes both i and k inactive, we can charge their total value, which is at least $\max(1, 2 - s)$. The following lemma states the performance of this rounding.

Lemma 6. *Let L be the total weight lost by the rounding. Then $L \leq \frac{1}{4}\max(1, 2 - s)$.*

Proof. The proof is technical and can be found in the full version of this paper.

Type 2 and Type 3 Roundings

Rounding. In type 2 rounding, we have a path with two keywords of degree 2; we consider all four possible ways of allocating each of those two keywords integrally to one of its two neighbors, transferring as much weight as possible in each allocation while respecting the LP constraints.

In type 3 rounding, we take a partial subtree rooted at the interesting node consisting of two of the paths below it. Together, these two paths define a path with two keywords of degree 2. We then proceed as in type 2 to define an integer allocation of those two keywords.

Analysis. In all cases, two saturated bidders become inactive. We show that the loss in the objective function is no more than 2/3 (Lemma 7), and charge it to the total value of the two bidders that become inactive, which is at least 2.

More precisely, consider a path i_1, j_1, i_2, j_2, i_3 where j_1 and j_2 are keywords. We show four ways to round this path so that one of the edges $\{i_1, j_1\}$, $\{i_2, j_1\}$ and one of the edges $\{i_2, j_2\}$, $\{i_3, j_2\}$ is removed in a way that loses no more than 2/3 (Lemma 7). If this path is a 3-chain rooted at i_1, then this procedure makes i_2 and i_3 inactive. Hence, we can charge the loss to the total value of i_2 and i_3, which is at least 2. On the other hand, if i_2 is the highest node on this path, j_1 and j_2 are degree-2 keywords and i_1 and i_3 do not have any degree-2 children, then this procedure makes i_1 and i_3 inactive. Hence, we can charge the loss to the total value of i_1 and i_3, which is at least 2.

Let $\gamma = b_{i_2 j_1}/b_{i_1 j_1}$, $\beta = b_{i_2 j_2}/b_{i_3 j_2}$, $a = x_{i_1 j_1} b_{i_1 j_1}$, $b = x_{i_2 j_1} b_{i_1 j_1}$, $c = x_{i_2 j_2} b_{i_3 j_2}$ and $d = x_{i_3 j_2} b_{i_3 j_2}$. Then $w_{i_1 j_1} = a$, $w_{i_2 j_1} = b\gamma$, $w_{i_2 j_2} = c\beta$ and $w_{i_3 j_2} = d$. This situation is illustrated in Fig. 3(a).

We consider four ways to round the path, illustrated in Figs. 3(b)-3(e). In the first way (Fig. 3(b)), we remove the edges $\{i_1, j_1\}$ and $\{i_3, j_2\}$, losing $a + d$. In the second way (Fig. 3(c)), we remove the edges $\{i_2, j_1\}$ and $\{i_2, j_2\}$, losing $b\gamma + c\beta$. In the third way (Fig. 3(d)), we remove the edges $\{i_1, j_1\}$ and $\{i_2, j_2\}$ and move part of the removed weight to $\{i_2, j_1\}$. If the entire amount of j_1 that was previously allocated to i_1 were allocated to i_2, then $w_{i_2 j_1}$ would increase by $x_{i_1 j_1} b_{i_2 j_1} = (a/b_{i_1 j_1}) b_{i_2 j_1} = a\gamma$. The budget freed up at i_2 from the removal of edge $\{i_2, j_2\}$ is $c\beta$. Thus, $w_{i_2 j_1}$ can be increased to at least $b\gamma + \min(a\gamma, c\beta)$, causing a loss of $a + c\beta - \min(a\gamma, c\beta) = a + \max(0, c\beta - a\gamma)$. In the fourth way (Fig. 3(e)), we remove the edges $\{i_2, j_1\}$ and $\{i_3, j_2\}$ and transfer as much as possible of the removed weight to $\{i_2, j_2\}$, causing a loss of $d + \max(0, b\gamma - d\beta)$.

Again, we choose the way that incurs the smallest loss. The following lemma states that this loss is never greater than 2/3.

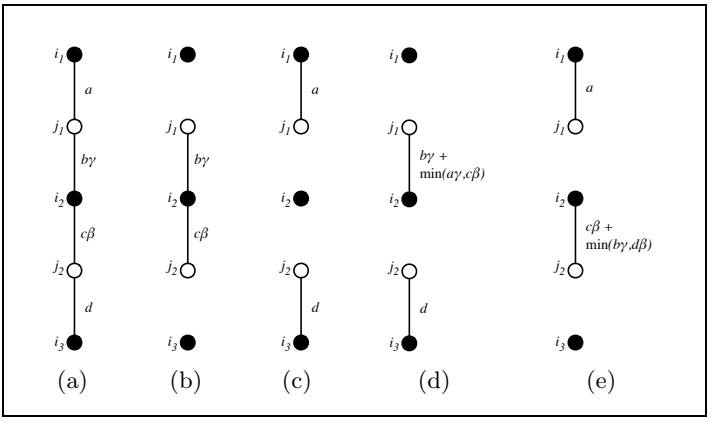


Fig. 3. A path with three bidders (a) and four ways to round that path (b)-(e)

Lemma 7. *Let*

$$L = \min\left(a + d, b\gamma + c\beta, a + \max(0, c\beta - a\gamma), d + \max(0, b\gamma - d\beta)\right) \quad .$$

Then $L \leq \frac{2}{3}$.

Proof. The proof is technical and can be found in the full version of this paper.

Type 4 Rounding

Let v be the interesting node of type 4, and let u be its parent. Let h and k be the number of 1-chains and 2-chains rooted at children of v, respectively. We consider three cases based on the value of h and k:

1. There are no 2-chains attached to v ($k = 0$). Then $h > 1$.
   Rounding. Among the edges adjacent to v, retain the edge of largest weight and delete all others.
   Analysis. We lose at most $h/(h+1)$ and make at least h saturated bidders inactive. Therefore, we can charge the loss to these nodes.
2. There are no 1-chains attached to v ($h = 0$). Then $k > 1$. Let $p_1, p_2, \ldots p_k$ be v's children.
   Rounding. We first round the path consisting of the edges $\{u, v\}$, $\{v, p_1\}$ and the 2-chain rooted at p_1, losing at most 2/3 by Lemma 7. After this step, either p_1 or u is disconnected from v. We repeat the above step with the path containing the edge joining v and the other node (either u or p_1), $\{v, p_2\}$ and the 2-chain rooted at p_2. We repeat this k times.
   Analysis. We lose at most $2k/3$ and make $2k$ saturated bidders inactive: $p_1, p_2, \ldots p_k$ and their grandchildren. Hence, we can charge the loss to these nodes.
3. Both $h > 0$ and $k > 0$.
   Rounding. We choose one 1-chain rooted at, say, q, and one 2-chain rooted at, say, p and round the path containing q, v, p and the 2-chain rooted at p.
   Analysis. We lose at most 2/3 by Lemma 7 and make two saturated bidders inactive: p's grandchild and either q or p. Hence, we can charge the loss to these nodes.

3 A $\sqrt{2}$-Approximation Algorithm for the Uniform Problem

In this section, we provide an algorithm that improves the approximation ratio to $\sqrt{2}$ for the uniform case of the problem. The main observation that leads to this improvement is that in the proof of Theorem 2, there was some weight that we could have charged to but that we did not use. For example, consider the type 2 rounding shown in Figure 3(b). We charged the loss of the rounding to the weight allocated to bidders i_2 and i_3, which must be at least 2 since these bidders are saturated. We can do better than this, however. In the rounding, a weight of

a is deallocated from i_1. Because we rebalance according to Lemma 1 between every rounding, this is weight that will never be charged to again. Therefore, instead of charging to 2, we can actually charge to $2 + a$.

To make this more precise, we define an *active edge* to be an edge that is adjacent to an active bidder. During each rounding, the sum of the weights on active edges will decrease, both from active edges becoming inactive and from active edges being deleted or losing weight. Define the *accountable amount* of a rounding to be the amount by which this quantity decreases. We will show that the loss of each rounding can be charged to $(1 - 1/\sqrt{2})$ of the accountable amount of that rounding. Since each unit of accountable amount is charged at most once, this suffices to prove the approximation ratio.

The structure of Algorithm 2, our $\sqrt{2}$-approximation, is the same as that of Algorithm 1. The only difference is in the rounding subroutines and their analysis. Instead of choosing the rounding that minimizes the loss at each step, we choose the one that minimizes the ratio between the loss and the accountable amount. In the remainder of this section we prove the following.

Theorem 8. *Algorithm 2 is a polynomial-time $\sqrt{2}$-approximation for the uniform version of the Budgeted Allocation problem.*

Proof. As in Theorem 2, the algorithm terminates in polynomial time and outputs an integral solution. For each rounding subroutine, we show that we can charge the loss to $1 - 1/\sqrt{2}$ of the accountable amount. This implies that Algorithm 2 returns a solution of value at least $1/\sqrt{2}$ of optimal. □

We believe that Theorem 8 applies to the non-uniform version of the problem, but we have not been able to prove this, since it seems to involve calculations that are significantly more complicated than those in the proof of Lemma 7.

3.1 The Rounding Subroutines

For convenience, we define x to be $2 - \sqrt{2}$. For each rounding subroutine, we show that the ratio of the loss to the accountable amount is no greater than $x/2 = 1 - 1/\sqrt{2}$.

Type 1 Rounding

The rounding and analysis for type 1 is the same as for Algorithm 1. By Lemma 6, the ratio of the loss to the accountable amount is no greater than $1/4 < x/2$.

Type 2 Rounding

Rounding. Let i_1 be the interesting node and i_1, j_1, i_2, j_2, i_3 be the 3-chain associated with i_1, and let $a = w_{i_1 j_1}$, $b = w_{i_2 j_1}$, $c = w_{i_2 j_2}$ and $d = w_{i_3 j_2}$. Of the four ways to round this chain described in Fig. 3, we choose the rounding that minimizes the ratio of the loss over the accountable amount.

Analysis. The four ways to round the chain incur losses of $a+d$, $b+c$, $\max(a,c)$, and $\max(b,d)$, respectively. (Recall that in the uniform version $\beta = \gamma = 1$.) To derive the accountable amounts of the roundings, note that the first rounding makes i_2 and i_3 inactive, and thus makes all edges adjacent to these nodes inactive; the sum of these edges is at least 2, since these bidders are saturated. Furthermore, it also removes the active edge $\{i_1, j_1\}$. Thus, the accountable amount of the first rounding is $2 + a$. Similarly, the accountable amount of the second, third and fourth roundings are $2, 2 + a$ and 2, respectively. Hence, the following lemma shows that the we can always choose a rounding such that the ratio of the loss to the accountable amount is no greater than $x/2$.

Lemma 9. *Let*

$$R = \min\left(\frac{a+d}{2+a}, \frac{b+c}{2}, \frac{\max(a,c)}{2+a}, \frac{\max(b,d)}{2}\right) .$$

Then $R \leq x/2$.

Proof. The proof is technical and can be found in the full version of this paper.

Type 3 and Type 4 Roundings

The rounding subroutine and analysis for type 3 interesting nodes is similar to the rounding and analysis for type 2 interesting nodes. The rounding and analysis for type 4 interesting nodes is quite involved, though the main ideas are similar. The details of type 3 and type 4 rounding can be found in the full version of this paper.

4 APX-Hardness

In this section, we show the following result.

Theorem 10. *Budgeted Allocation is APX-hard, even in the uniform version.*

Proof. The proof is a simple reduction from 3D-Matching, and can be found in the full version of this paper.

References

1. Andelman, N., Mansour, Y.: Auctions with Budget Constraints. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111. Springer, Heidelberg (2004)
2. Feige, U., Vondrak, J.: Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In: FOCS 2006, pp. 667–676 (2006)
3. Lahaie, S., Pennock, D., Saberi, A., Vohra, R.: Sponsored search auctions. In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V. (eds.) Algorithmic Game Theory, pp. 699–716. Cambridge University Press, Cambridge (2007)
4. Buchbinder, N., Jain, K., Naor, J.: Online primal-dual algorithms for maximizing ad-auctions revenue. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 253–264. Springer, Heidelberg (2007)

5. Goel, G., Mehta, A.: Online budgeted matching in random input models with applications to adwords. In: SODA 2008, pp. 982–991 (2008)
6. Mahdian, M., Nazerzadeh, H., Saberi, A.: Allocating online advertisement space with unreliable estimates. In: EC 2007, pp. 288–294 (2007)
7. Mehta, A., Saberi, A., Vazirani, U., Vazirani, V.: Adwords and generalized online matching. J. ACM 54(5), 22 (2007)
8. Lehmann, B., Lehmann, D., Nisan, N.: Combinatorial auctions with decreasing marginal utilities. Games and Economic Behavior 55(2), 270–296 (2006)
9. Chakrabarty, D., Goel, G.: On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and GAP (manuscript, 2008)
10. Srinivasan, A.: Budgeted allocations in the full-information setting (manuscript, 2008)
11. Dobzinski, S., Schapira, M.: An improved approximation algorithm for combinatorial auctions with submodular bidders. In: SODA 2006, pp. 1064–1073 (2006)
12. Khot, S., Lipton, R., Markakis, E., Mehta, A.: Inapproximability Results for Combinatorial Auctions with Submodular Utility Functions. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 92–101. Springer, Heidelberg (2005)
13. Mirrokni, V., Schapira, M., Vondrak, J.: Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions (manuscript, 2007)
14. Vondrak, J.: Optimal approximation for the submodular welfare problem in the value oracle model. In: STOC 2008 (to appear, 2008)
15. Fleischer, L., Goemans, M., Mirrokni, V., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: SODA 2006, pp. 611–620 (2006)
16. Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. In: SODA 2000, pp. 213–222 (2000)
17. Shmoys, D., Tardos, E.: An approximation algorithm for the generalized assignment problem. Mathematical Programming 62, 461–474 (1993)
18. Andelman, N.: Online and strategic aspects of network resource management algorithms. PhD thesis, Tel Aviv University (2006)

The Travelling Salesman Problem in Bounded Degree Graphs*

Andreas Björklund[1], Thore Husfeldt[1], Petteri Kaski[2], and Mikko Koivisto[2]

[1] Lund University, Department of Computer Science,
P.O.Box 118, SE-22100 Lund, Sweden
andreas.bjorklund@logipard.com, thore.husfeldt@cs.lu.se
[2] Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, P.O.Box 68, FI-00014 University of Helsinki, Finland
petteri.kaski@cs.helsinki.fi, mikko.koivisto@cs.helsinki.fi

Abstract. We show that the travelling salesman problem in bounded-degree graphs can be solved in time $O\big((2-\epsilon)^n\big)$, where $\epsilon > 0$ depends only on the degree bound but not on the number of cities, n. The algorithm is a variant of the classical dynamic programming solution due to Bellman, and, independently, Held and Karp. In the case of bounded integer weights on the edges, we also present a polynomial-space algorithm with running time $O\big((2-\epsilon)^n\big)$ on bounded-degree graphs.

1 Introduction

There is no faster algorithm known for the travelling salesman problem than the classical dynamic programming solution from the early 1960s, discovered by Bellman [2,3], and, independently, Held and Karp [9]. It runs in time within a polynomial factor of 2^n, where n is the number of cities. Despite the half a century of algorithmic development that has followed, it remains an open problem whether the travelling salesman problem can be solved in time $O(1.999^n)$ [15].

In this paper we provide such an upper bound for graphs with bounded maximum vertex degree. For this restricted graph class, previous attemps have succeeded to prove such bounds when the degree bound, Δ, is three or four. Indeed, Eppstein [6] presents a sophisticated branching algorithm that solves the problem in time $2^{n/3}n^{O(1)} = O(1.260^n)$ on cubic graphs ($\Delta = 3$) and in time $O(1.890^n)$ for $\Delta = 4$. Recently, Iwama and Nakashima [10] improved the former bound to $O(1.251^n)$. These algorithms run in space polynomial in n. Very recently, Gebauer [7] gave an exponential-space algorithm that runs in time $(\Delta-1)^{n/2}n^{O(1)}$ and can also list the Hamiltonian cycles, improving the time bound for $\Delta = 4$ to $O(1.733^n)$. However, for $\Delta > 4$ none of these techniques seems to improve upon $O(2^n)$.

* This research was supported in part by the Swedish Research Council, project "Exact Algorithms" (A.B., T.H.), and the Academy of Finland, Grants 117499 (P.K.) and 109101 (M.K.).

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 198–209, 2008.

We show that, perhaps somewhat surprisingly, with minor modifications the classical Bellman–Held–Karp algorithm can be made to run in time $O\big((2-\epsilon)^n\big)$, where $\epsilon > 0$ depends only on the degree bound:

Theorem 1. *The travelling salesman problem for an n-vertex graph with maximum degree $\Delta = O(1)$ can be solved in time $\xi_\Delta^n n^{O(1)}$ with*

$$\xi_\Delta = (2^{(\Delta+1)} - 2\Delta - 2)^{1/(\Delta+1)} \ .$$

Our main contribution is indeed more analytical than algorithmic, and largely relies on exploiting variants of a beautiful lemma due to Shearer [5] ("Shearer's Entropy Lemma") that in a combinatorial context enables one to derive upper bounds for the size of a set family based on the sizes of its projections.

We used this lemma recently in connection with analysing expedited versions of the FFT-like algorithm of Yates to solve covering problems for bounded-degree graphs via Moebius inversion [4], realising only later that classical algorithms for the travelling salesman problem yield to the same analytical tools. In general, this approach seems to be new and quite versatile for bounding the running time of dynamic programming algorithms on restricted graph classes; to illustrate this, we prove a stronger bound for regular triangle-free graphs:

Theorem 2. *The travelling salesman problem for a triangle-free n-vertex graph where every vertex has degree $\Delta = O(1)$ can be solved in time $\eta_\Delta^n n^{O(1)}$ with*

$$\eta_\Delta = \big(2^{2\Delta} - (\Delta+1)2^{\Delta+1} + 2(\Delta^2+1)\big)^{1/(2\Delta)} \ .$$

To motivate a yet further illustration, we observe that the algorithms in Theorems 1 and 2 both require exponential space, which immediately prompts the question whether there exists a polynomial-space algorithm with running time $O\big((2-\epsilon)^n\big)$ on bounded-degree graphs. This turns out to be the case if the edge weights are bounded integers.

Indeed, a classical polynomial-space algorithm due to Karp [11] and, independently, Kohn, Gottlieb, and Kohn [12], can be made to run in time $O\big((2-\epsilon)^n\big)$ on bounded-degree graphs, again with only minor tailoring.

Somewhat perplexingly, we characterise the running time of the polynomial-space algorithm in terms of the *connected dominating sets* of the input graph. To properly state the result, we recall the definitions here. For a graph G and a set $W \subseteq V$ of vertices, the set W is a *connected set* if the induced subgraph $G[W]$ is connected; and, a *dominating set* if every vertex $v \in V$ is in W or adjacent to a vertex in W. Denote by $\mathcal{C}$ the family of connected sets of G, and by $\mathcal{D}$ the family of dominating sets of G.

Theorem 3. *The travelling salesman problem for an n-vertex graph with bounded integer weights can be solved in time $|\mathcal{C} \cap \mathcal{D}| n^{O(1)}$ and in space $n^{O(1)}$. In particular, for maximum degree Δ it holds that $|\mathcal{C} \cap \mathcal{D}| \leq \gamma_\Delta^n + n$, where*

$$\gamma_\Delta = (2^{\Delta+1} - 2)^{1/(\Delta+1)} \ .$$

Table 1. The constants in Theorems 1, 2, and 3 for small values of Δ

Δ	3	4	5	6	7	8	$\cdots$
β_Δ	1.9680	1.9874	1.9948	1.9978	1.9991	1.9999	$\cdots$
γ_Δ	1.9343	1.9744	1.9894	1.9955	1.9980	1.9991	$\cdots$
ξ_Δ	1.6818	1.8557	1.9320	1.9672	1.9840	1.9921	$\cdots$
η_Δ	1.6475	1.8376	1.9231	1.9630	1.9820	1.9912	$\cdots$

Remark. Table 1 displays the constants in Theorems 1, 2, and 3 for small values of Δ. We expect there to be room for improvement in each of the derived bounds. In particular, in this regard we would like to highlight the question of asymptotically tight upper bounds for $|\mathcal{C}|$, $|\mathcal{D}|$, and $|\mathcal{C} \cap D|$ on bounded-degree graphs (cf. Lemma 3). Such bounds should be of independent combinatorial interest, and we fully expect better bounds to occur in the literature, even if we were unable to find these.

Organisation. The combinatorial analysis tools are established in Sect. 2. We establish a precursor to Theorem 1 in Sect. 3 using a simple argument that illustrates the main ideas of our approach, but leads to a weaker running time bound $\beta_\Delta^n n^{O(1)}$ with $\beta_\Delta = (2^{\Delta+1} - 1)^{1/(\Delta+1)}$. Theorems 1, 2, and 3 are proved in Sects. 4, 5, and 6, respectively.

1.1 Conventions

We consider the directed, asymmetric variant of the travelling salesman problem. A problem instance consists of an n-element ground set V and a *weight* $d(u, v) \in \{0, 1, \ldots\} \cup \{\infty\}$ for all distinct $u, v \in V$. A *tour* is a permutation $(v_1, v_2, \ldots, v_n)$ of V. The *weight* of a tour is $d(v_1, v_2) + d(v_2, v_3) + \cdots + d(v_{n-1}, v_n) + d(v_n, v_1)$. Given a problem instance, the task is to find the minimum weight of a tour. For further background on the travelling salesman problem, we refer to [1,8,13].

We associate with each problem instance an **undirected** graph G with vertex set V and edge set E such that any two distinct $u, v \in V$ are joined by an edge $\{u, v\}$ if and only if $d(u, v) < \infty$ or $d(v, u) < \infty$. Unless explicitly indicated otherwise, all graph-theoretic terminology refers to the graph G. For standard graph-theoretic terminology we refer to [14].

2 Combinatorial Preliminaries

We are interested in upper bounds for the sizes of certain set families associated with a graph with maximum degree Δ. Our starting point is the following lemma due to Shearer (see [5]).

Lemma 1 (Chung, Frankl, Graham, and Shearer [5]). *Let V be a finite set with subsets $A_1, A_2, \ldots, A_r$ such that every $v \in V$ is contained in at least*

δ subsets. Let $\mathcal{F}$ be a family of subsets of V. For each $1 \le i \le r$ define the projections $\mathcal{F}_i = \{F \cap A_i : F \in \mathcal{F}\}$. Then,

$$|\mathcal{F}|^\delta \le \prod_{i=1}^{r} |\mathcal{F}_i| \ .$$

First, we bring the lemma into a format that is more useful for our present purposes. For instance, we will find it handy to leave out a constant number s of special subsets. The following lemma abstracts and to a certain extent generalises an analysis we have presented earlier [4, Theorem 3.2].

Lemma 2. *Let V be a finite set with r elements and with subsets $A_1, A_2, \ldots, A_r$ such that every $v \in V$ is contained in exactly δ subsets. Let $\mathcal{F}$ be a family of subsets of V and assume that there is a log-concave function $f \ge 1$ and an $0 \le s \le r$ such that the projections $\mathcal{F}_i = \{F \cap A_i : F \in \mathcal{F}\}$ satisfy $|\mathcal{F}_i| \le f(|A_i|)$ for each $s+1 \le i \le r$. Then,*

$$|\mathcal{F}| \le f(\delta)^{r/\delta} \prod_{i=1}^{s} 2^{|A_i|/\delta} \ .$$

Proof. Let $a_i = |A_i|$ and note that $a_1 + a_2 + \cdots + a_r = \delta r$. By Lemma 1, we have

$$|\mathcal{F}|^\delta \le \prod_{i=1}^{s} 2^{a_i} \prod_{i=s+1}^{r} f(a_i) \le \prod_{i=1}^{s} 2^{a_i} \prod_{i=1}^{r} f(a_i) \ . \tag{1}$$

Since f is log-concave, Jensen's inequality gives

$$\frac{1}{r} \sum_{i=1}^{r} \log f(a_i) \le \log f\big((a_1 + a_2 + \cdots + a_r)/r\big) = \log f(\delta) \ .$$

Taking exponentials and combining with (1) gives

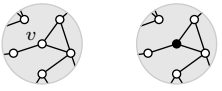
$$|\mathcal{F}|^\delta \le f(\delta)^r \prod_{i=1}^{s} 2^{a_i} \ ,$$

which yields the claimed bound. □

For Theorem 1 it suffices to consider the special case where the A_i are defined in terms of neighbourhoods of the vertices of G. For each $v \in V$, define the closed neighbourhood $N(v)$ by

$$N(v) = \{v\} \cup \{u \in V : u \text{ and } v \text{ are adjacent in } G\} \ .$$

Begin by defining the subsets A_v for $v \in V$ as $A_v = N(v)$. Then, for each $u \in V$ with degree $d(u) < \Delta$, add u to $\Delta - d(u)$ of the sets A_v not already containing it (it does not matter which). This ensures that every vertex $u \in V$ is contained in exactly $\Delta + 1$ sets A_v. Fig. 1(a) shows an example. For each $v \in V$, call the set A_v so obtained the *region* of v.



(a) (b)

Fig. 1. (a) The region A_v of a vertex v in a graph with $\Delta = 5$. (b) Impossible projection for a connected set $C \in \mathcal{C}$, $|C| \geq 2$; if only the black vertex belongs to C then C cannot be connected, because all of v's neighbours belong to A_v.

Lemma 3. *An n-vertex graph with maximum vertex degree Δ has at most $\beta_\Delta^n + n$ connected sets and at most $\gamma_\Delta^n + n$ connected dominating sets, where*

$$\beta_\Delta = (2^{\Delta+1} - 1)^{1/(\Delta+1)}, \quad \gamma_\Delta = (2^{\Delta+1} - 2)^{1/(\Delta+1)} \ .$$

Proof. Recall that by $\mathcal{C}$ we denote the family of connected sets and by $\mathcal{D}$ the family of dominating sets. Let $\mathcal{C}' = \mathcal{C} \setminus \{\{v\} : v \in V\}$. Then for every $C' \in \mathcal{C}'$ and every region A_v, $C' \cap A_v \neq \{v\}$; see Fig. 1(b). Thus the number of sets in the projection $\mathcal{C}'_v = \{F \cap A_v : F \in \mathcal{C}\}$ is at most $2^{|A_v|} - 1$. To obtain the bound on connected sets, apply Lemma 2 with the log-concave function $f(a) = 2^a - 1$ and $s = 0$. To obtain the upper bound for $|\mathcal{C} \cap \mathcal{D}|$, observe that, in addition to the singleton projection excluded for a connected set, also the empty projection is excluded for each region in the case of a connected dominating set. □

3 Connected Sets

This section establishes Theorem 1, but with a weaker bound; the purpose is to show a very straightforward argument for an $O\big((2-\epsilon)^n\big)$ upper bound.

Our starting point is the dynamic programming solution, which we proceed to recall. Select an arbitrary reference vertex $s \in V$. For $T \subseteq V$ and $v \in T$, denote by $D(T, v)$ the minimum weight of a directed path (in the complete directed graph with vertex set V and edge weights given by d) from s to v that consists of the vertices in T. The minimum weight of a tour is then solved by computing

$$\min_{v \in V} D(V, v) + d(v, s) \ .$$

To construct $D(T, v)$ for all $s \in T \subseteq V$ and all $v \in T$, the algorithm starts with $D(\{s\}, s) = 0$, and evaluates the recurrence

$$D(T, v) = \min_{u \in T \setminus \{v\}} D(T \setminus \{v\}, u) + d(u, v) \ . \tag{2}$$

The values $D(T, v)$ are stored a table when they are computed to avoid redundant recomputation,an idea sometimes called *memoisation*. The space and time requirements are within a polynomial factor of 2^n, the number of subsets $T \subseteq V$.

Our idea to expedite this will restrict the family of subsets for which (2) is ever evaluated. To this end, consider any prefix $(v_1, v_2, \ldots, v_k)$ of a finite-weight tour with $v_1 = s$. The set of vertices $T = \{v_1, v_2, \ldots, v_k\}$ satisfies certain

connectivity properties that we want to exploit. In the present section, we use merely the trivial observation that T must be a connected set. Put otherwise, $D(T, v) = \infty$ unless T is a connected set. Thus, it suffices to evaluate (2) not over all subsets of V, but only over the family of connected sets $\mathcal{C}$. A bottom-up evaluation of (2) with memoisation gives an algorithm for solving the travelling salesman problem within time $|\mathcal{C}|$ up to polynomial factors. (Indeed, whether $T \in \mathcal{C}$ can be tested in polynomial time by, e.g., depth-first search; furthermore, for every $T \in \mathcal{C}$ with $|T| > 1$ there exists at least one $v \in T$ with $T \setminus \{v\} \in \mathcal{C}$ – consider the leaves of a spanning tree of $G[T]$ – which enables T to be discovered from $T \setminus \{v\}$.) With Lemma 3 this gives $O\big((2-\epsilon)^n\big)$ running time when G has maximum degree $O(1)$.

4 Transient Sets

This section establishes Theorem 1, which amounts to a more careful analysis of sets of vertices T occurring in prefixes of a tour with finite weight. For example, such a set T cannot contain all vertices adjacent to a vertex $v \notin T \cup N(s)$, because then the tour necessarily either avoids v or gets stuck at v without returning to s.

In precise terms, a vertex set $T \subseteq V$ is *transient with endpoint* $u \in T$ if it is connected, $s \in T$, and the following holds for every vertex $v \notin N(s) \cup N(u)$:

1. if v belongs to T, then so do at least two of its adjacent vertices;
2. if v does not belong to T, then neither do at least two of its adjacent vertices.

Note that testing if a vertex set is transient is a polynomial time task, we merely need to run a depth-first-search and checking each vertex neighbourhood for the two properties above.

Let $\mathcal{T}_u$ denote the family of vertex sets that are transient with endpoint u.

Observe that any prefix $(v_1, v_2, \ldots, v_k)$ of a finite-weight tour with $v_1 = s$ and $v_k = u$ has the property that $\{v_1, v_2, \ldots, v_k\} \in \mathcal{T}_u$. It thus suffices to consider the recurrence

$$D(T, v) = \min_{\substack{u \in T \setminus \{v\} \\ T \setminus \{v\} \in \mathcal{T}_u}} D(T \setminus \{v\}, u) + d(u, v) \ , \tag{3}$$

where we tacitly assume that the minimum of an empty set is ∞.

A top-down evaluation of (3) with memoisation leads to running time bounded by, up to polynomial factors,

$$\sum_{u \in V} |\mathcal{T}_u| \leq n \max_{u \in V} |\mathcal{T}_u| \ . \tag{4}$$

To derive an upper bound for the size of $\mathcal{T}_u$, consider an arbitrary $u \in V$ and set $\delta = \Delta + 1$. Call a vertex $v \in V$ *special* if $N(v) \cap \big(N(s) \cup N(u)\big) \neq \emptyset$, and observe that there are at most $2(1 + \Delta^2) < 2\delta^2$ special vertices.

Now consider a non-special $v \in V$ and an arbitrary $T \in \mathcal{T}_u$. Let $a_v = |A_v|$. We can rule out the following projections $A_v \cap T$; see Fig. 2 for an example.

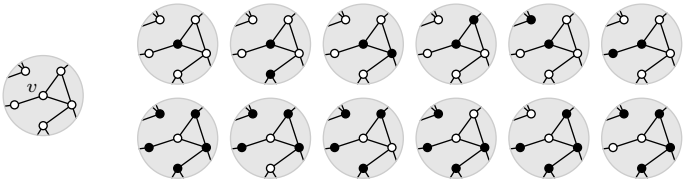


Fig. 2. A non-special region A_v (left) and the impossible intersections of A_v with a (black) transient set

1. $v \in T$ and $|A_v \cap T| = 1$, so v has no neighbours in T. The tour never enters or leaves v. This can happen only if v is special.
2. $v \in T$ but $|A_v \cap T| = 2$, so v has at most one neighbour in T. The tour never leaves v. This can happen only if v is special. There are at least $a_v - 1$ such cases (more if A_v contains vertices not connected to v).
3. $v \notin T$ but $A_v \setminus \{v\} \subseteq T$, so all of v's neighbours are in T. When the tour arrives in v it cannot leave. This can happen only if v is special.
4. $v \notin T$ but $|A_v \cap T| = a_v - 2$, so v has at most one neighbour also not in T. When the tour arrives in v it cannot leave. This can happen only if v is special. There are $a_v - 1$ such cases (more if A_v contains vertices not connected to v).

In total, we can rule out $2a_v$ of the 2^{a_v} potential projections. We now want to apply Lemma 2. To this end, we have to be slightly more careful as regards the arbitrary construction of the regions A_v (recall Sect. 2). In particular, whenever v is special, we want $|A_v| \leq \delta$. For all large enough n and $\delta = O(1)$ this is easily arranged by not inserting additional vertices into a special A_v when $|A_v| = \delta$. Thus, we can apply Lemma 2 with $f(a) = 2^a - 2a$ and at most $2\delta^2$ special projectors A_v, each of size at most δ. We conclude that

$$|\mathcal{T}_u| \leq \left(2^\delta - 2\delta\right)^{n/\delta} 2^{2\delta^2} \quad . \tag{5}$$

Theorem 1 follows, with the asymptotic notation absorbing a factor n from (4) and a constant factor from (5).

5 Triangle-Free Graphs

To prove Theorem 2 we analyse the vertex sets of tour prefixes using a family of subsets B_e centered around every edge. The argument is somewhat more involved, but the bound becomes slightly better. We assume that G is regular with degree $\Delta = O(1)$ and contains no triangles.

Consider again the vertices $T = \{v_1, v_2, \ldots, v_k\}$ on a prefix of a finite-weight tour, $v_1 = s$, $v_k = u$. Suppose that e is an edge joining two vertices, x and y. Then, provided that e is again non-special, that is, sufficiently far from both s and u, we can again rule out certain projections of T to B_e:

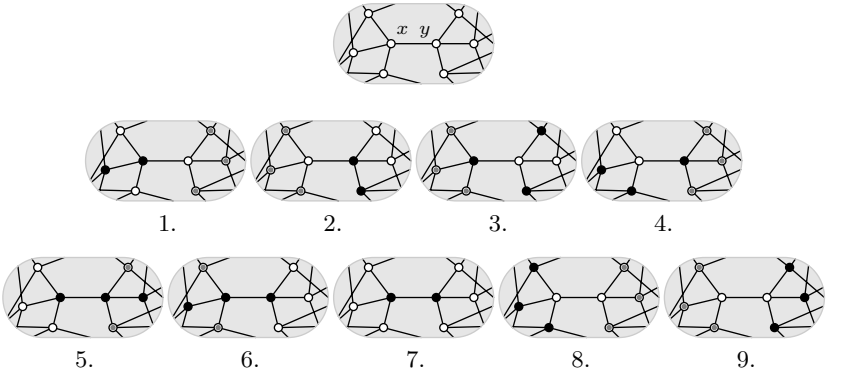


Fig. 3. Some impossible projections for regular triangle-free graphs. B_e is the vertex subset at the top. The black vertices are in T, the grey vertices can be in T or not.

1. if both x and y belong to T then either the tour travels along e, in which case x and y each must have another neighbour in T, or the edge e is not on the tour, in which case x and y each must have two other neighbours in T.
2. if only one of the vertices, say x, belongs to T then it must have two other neighbours in T. Moreover, the other vertex y cannot be completely surrounded by neighbours in T.

There are a number of symmetrical cases to these, all of which are checked in constant time around every edge. See Fig. 3 for an example; a detailed enumeration of the cases appears as part of the analysis below.

For each edge e in G, define B_e as the union of the closed neighbourhoods of its endpoints,

$$B_e = N(x) \cup N(y), \quad e \text{ joins } x \text{ and } y \ .$$

Because G is triangle-free and Δ-regular, each vertex $v \in V$ belongs to exactly $\delta = \Delta^2$ sets B_e.

We now turn to a detailed analysis of the projections $B_e \cap T$. To this end, partition B_e into $B_e = \{x\} \cup \{y\} \cup M(x) \cup M(y)$, where $M(x) = N(x) \setminus \{x, y\}$ and $M(y) = N(y) \setminus \{x, y\}$. We have $|M(x)| = |M(y)| = \Delta - 1$ because G is triangle-free. Call an edge e *special* if $B_e \cap \big(N(s) \cup N(u)\big) \neq \emptyset$. Because $\Delta = O(1)$, there are $O(1)$ special edges.

For a non-special e, we can rule out the following (non-disjoint) types of intersections $B_e \cap T$, exemplified in Fig. 3.

1. $x \in T$, $y \notin T$, $|M(x) \cap T| \leq 1$. The tour would never leave x. There are $\Delta 2^{\Delta-1}$ such cases.
2. Symmetrically, $y \in T$, $x \notin T$, $|M(y) \cap T| \leq 1$. There are $\Delta 2^{\Delta-1}$ such cases.
3. $x \in T$, $y \notin T$, $|M(y) \cap T| \geq \Delta - 2$. The tour would never reach and leave y. There are $\Delta 2^{\Delta-1}$ such cases.
4. Symmetrically, $y \in T$, $x \notin T$, $|M(x) \cap T| \geq \Delta - 2$. There are $\Delta 2^{\Delta-1}$ such cases.

5. $x \in T, y \in T$, $M(x) \cap T = \emptyset$, and $M(y) \cap T \neq \emptyset$. The tour never leaves x. There are $2^{\Delta-1} - 1$ such cases.
6. Symmetrically, $x \in T, y \in T$, $M(y) \cap T = \emptyset$, and $M(x) \cap T \neq \emptyset$. There are $2^{\Delta-1} - 1$ such cases.
7. $x \in T$, $y \in T$, $M(x) \cap T = M(y) \cap T = \emptyset$. The tour cannot leave $\{x, y\}$. There is 1 such case.
8. $x \notin T$, $y \notin T$, $M(x) \subseteq T$. The tour cannot leave x. There are $2^{\Delta-1}$ such cases.
9. Symmetrically, $x \notin T$, $y \notin T$, $M(y) \subseteq T$. There are $2^{\Delta-1}$ such cases.

In calculating the total number of forbidden intersections, observe that Types 1 and 3 are not disjoint (symmetrically, Types 2 and 4 are not disjoint). Both pairs of types have Δ^2 cases in common. Also, Types 8 and 9 are not disjoint; there is 1 case in common. Thus, in total we can rule out

$$4\Delta 2^{\Delta-1} + 2(2^{\Delta-1} - 1) + 1 + 2 \cdot 2^{\Delta-1} - 2\Delta^2 - 1 = (\Delta + 1)2^{\Delta+1} - 2(\Delta^2 + 1)$$

projections, so the number of projections is bounded by

$$2^{2\Delta} - (\Delta + 1)2^{\Delta+1} + 2(\Delta^2 + 1) \ .$$

We can apply Lemma 2 with $\delta = \Delta^2$, $r = |E| = \Delta n/2$, the resulting bound is

$$\left(2^{2\Delta} - (\Delta + 1)2^{\Delta+1} + 2(\Delta^2 + 1)\right)^{r/\delta} \cdot O(1) \ ,$$

which establishes Theorem 2 with (4) and (5).

6 Polynomial Space

For Theorem 3 our starting point is an algorithm of Karp [11], and, independently, Kohn, Gottlieb, and Kohn [12]. We assume that the weights $d(u, v)$ are bounded, that is, $d(u, v) \in \{0, 1, \ldots, B\} \cup \{\infty\}$, $B = O(1)$.

The algorithm is most conveniently described in terms of generating polynomials. Select an arbitrary reference vertex, $s \in V$, and let $U = V \setminus \{s\}$. For each $X \subseteq U$, denote by $q(X)$ the polynomial over the indeterminate z for which the coefficient of each monomial z^w counts the directed closed walks (in the complete directed graph with vertex set V and edge weights given by d) through s that (i) avoid the vertices in X; (ii) have length n; and (iii) have finite weight w.

We can compute $q(X)$ for a given $X \subseteq U$ in time polynomial in n by solving the following recurrence and setting $q(X) = p(n, s)$. Initialise the recurrence for each vertex $u \in V \setminus X$ with

$$p(0, u) = \begin{cases} 1 & \text{if } u = s; \\ 0 & \text{otherwise.} \end{cases}$$

For convenience, define $z^\infty = 0$. For each length $\ell = 1, 2, \ldots, n$ and each vertex $u \in V \setminus X$, let

$$p(\ell, u) = \sum_{v \in V \setminus X} p(\ell - 1, v) z^{d(v,u)} \ .$$

Note that due to our assumption on bounded weights, each $p(\ell, u)$ has at most a polynomial number of monomials with nonzero coefficients.

By the principle of inclusion–exclusion, the monomials of the polynomial

$$Q = \sum_{X \subseteq U} (-1)^{|X|} q(X) \tag{6}$$

count, by weight, the number of directed closed walks through s that (i) visit each vertex in U at least once; and (ii) have length n. Put otherwise, what is counted by weight are the directed Hamilton cycles. It follows immediately that the travelling salesman problem can be solved in space polynomial in n and in time $2^n n^{O(1)}$. This completes the description of the algorithm.

Let us now analyse (6) in more detail, with the objective of obtaining an algorithm with better running time on bounded-degree graphs. It will be convenient to work with a complemented form of (6), that is, for each $S \subseteq U$, let

$$r(S) = q(U \setminus S) \ ,$$

and rewrite (6) in the form

$$Q = (-1)^n \sum_{S \subseteq U} (-1)^{|S|} r(S) \ . \tag{7}$$

We want to reduce the number of $S \subseteq U$ that need to be considered in (7). To this end, observe that the induced subgraph $G[\{s\} \cup S]$ need not be connected. Associate with each $S \subseteq U$ the unique $f(S) \subseteq U$ such that $G[\{s\} \cup f(S)]$ is the connected component of $G[\{s\} \cup S]$ that contains s. Observe that $r(S) = r\big(f(S)\big)$ for all $S \subseteq U$. This observation enables the following partition of the subsets of U into f-preimages of constant r-value. For each $T \subseteq U$, let

$$f^{-1}(T) = \{S \subseteq U : f(S) = T\} \ ,$$

and rewrite (7) in the partitioned form

$$Q = (-1)^n \sum_{T \subseteq U} r(T) \sum_{S \in f^{-1}(T)} (-1)^{|S|} \ . \tag{8}$$

The inner sum in (8) turns out to be determined by the connected dominating sets of G.

Lemma 4. *For every $T \subseteq U$ it holds that*

$$\sum_{S \in f^{-1}(T)} (-1)^{|S|} = \begin{cases} (-1)^{|T|} & \text{if } \{s\} \cup T \text{ is a connected dominating set of } G; \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Consider an arbitrary $T \subseteq U$. The preimage $f^{-1}(T)$ is clearly empty if $G[\{s\} \cup T]$ is not connected. Thus in what follows we can assume that $G[\{s\} \cup T]$ is connected. For a set $W \subseteq V$, denote by $\bar{N}(W)$ the set of vertices in W or

adjacent to at least one vertex in W. Observe that $f(S) = T$ holds for an $S \subseteq U$ if and only if $S \supseteq T$ and $S \cap \bar{N}(\{s\} \cup T) = T$. In particular, if $V \setminus \bar{N}(\{s\} \cup T)$ is nonempty, then $f^{-1}(T)$ contains equally many even- and odd-sized subsets. Conversely, if $V \setminus \bar{N}(\{s\} \cup T)$ is empty (that is, $\{s\} \cup T$ is a dominating set of G), then $f^{-1}(T) = \{T\}$. □

Using Lemma 4 to simplify (8), we have

$$Q = (-1)^n \sum_{\substack{T \subseteq U \\ \{s\} \cup T \in \mathcal{C} \cap \mathcal{D}}} (-1)^{|T|} r(T) \ . \tag{9}$$

To arrive at an algorithm with running time $|\mathcal{C} \cap \mathcal{D}| n^{O(1)}$ and space usage $n^{O(1)}$, it now suffices to list the elements of $\mathcal{C} \cap \mathcal{D}$ in space $n^{O(1)}$ and with delay bounded by $n^{O(1)}$.

The following listing strategy can be considered folklore and is here sketched for interests of self-containment only. Observe that $\mathcal{C} \cap \mathcal{D}$ is an up-closed family of subsets of V, that is, if a set is in the family, then so are all of its supersets. Furthermore, whether a given $W \subseteq V$ is in $\mathcal{C} \cap \mathcal{D}$ can be decided in time $n^{O(1)}$. These observations enable the following top-down, depth-first listing algorithm for the sets in $\mathcal{C} \cap \mathcal{D}$. Initially, we visit the set V if and only if G is connected; otherwise $\mathcal{C} \cap \mathcal{D}$ is empty. Whenever we visit a set $Y \subseteq V$, we first list it, and then consider each of its maximal proper subsets $Y \setminus \{y\}$, $y \in Y$, in turn. We recursively visit $Y \setminus \{y\}$ if both (i) $Y \setminus \{y\} \in \mathcal{C} \cap \mathcal{D}$; and (ii) Y is the maximum (say, w.r.t. lexicographic order of subsets of V) minimal proper superset of $Y \setminus \{y\}$ in $\mathcal{C} \cap \mathcal{D}$.

Theorem 3 now follows from Lemma 3.

References

1. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton (2006)
2. Bellman, R.: Combinatorial Processes and Dynamic Programming. In: Bellman, R., Hall Jr., M. (eds.) Proceedings of Symposia in Applied Mathematics. Combinatorial Analysis, vol. 10, pp. 217–249. American Mathematical Society (1960)
3. Bellman, R.: Dynamic Programming Treatment of the Travelling Salesman Problem. J.Assoc.Comput.Mach. 9, 61–63 (1962)
4. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Trimmed Moebius Inversion and Graphs of Bounded Degree. In: 25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008). Dagstuhl Seminar Proceedings 08001, pp. 85–96. IBFI Schloss Dagstuhl (2008)
5. Chung, F.R.K., Frankl, P., Graham, R.L., Shearer, J.B.: Some Intersection Theorems for Ordered Sets and Graphs. J.Combinatorial Theory Ser.A 43, 23–37 (1986)
6. Eppstein, D.: The Traveling Salesman Problem for Cubic Graphs. J.Graph Algorithms Appl. 11, 61–81 (2007)
7. Gebauer, H.: On the Number of Hamilton Cycles in Bounded Degree Graphs. In: Fourth Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2008). SIAM, Philadelphia (2008)

8. Gutin, G., Punnen, A.P. (eds.): The Traveling Salesman Problem and its Variations. Kluwer, Dordrecht (2002)
9. Held, M., Karp, R.M.: A Dynamic Programming Approach to Sequencing Problems. J.Soc.Indust.Appl.Math. 10, 196–210 (1962)
10. Iwama, K., Nakashima, T.: An Improved Exact Algorithm for Cubic Graph TSP. In: Lin, G. (ed.) COCOON. LNCS, vol. 4598, pp. 108–117. Springer, Heidelberg (2007)
11. Karp, R.M.: Dynamic Programming Meets the Principle of Inclusion and Exclusion. Oper.Res.Lett. 1, 49–51 (1982)
12. Kohn, S., Gottlieb, A., Kohn, M.: A Generating Function Approach to the Traveling Salesman Problem. In: ACM Annual Conference (ACM 1977), pp. 294–300. ACM Press, New York (1977)
13. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.): The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester (1985)
14. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice–Hall (2001)
15. 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)

Treewidth Computation and Extremal Combinatorics*

Fedor V. Fomin and Yngve Villanger

Department of Informatics, University of Bergen,
N-5020 Bergen, Norway
{fedor.fomin,yngve.villanger}@ii.uib.no

Abstract. For a given graph G and integers $b, f \geq 0$, let S be a subset of vertices of G of size $b+1$ such that the subgraph of G induced by S is connected and S can be separated from other vertices of G by removing f vertices. We prove that every graph on n vertices contains at most $n\binom{b+f}{b}$ such vertex subsets. This result from extremal combinatorics appears to be very useful in the design of several enumeration and exact algorithms. In particular, we use it to provide algorithms that for a given n-vertex graph G

- compute the treewidth of G in time $\mathcal{O}(1.7549^n)$ by making use of exponential space and in time $\mathcal{O}(2.6151^n)$ and polynomial space;
- decide in time $\mathcal{O}((\frac{2n+k+1}{3})^{k+1} \cdot kn^6)$ if the treewidth of G is at most k;
- list all minimal separators of G in time $\mathcal{O}(1.6181^n)$ and all potential maximal cliques of G in time $\mathcal{O}(1.7549^n)$.

This significantly improves previous algorithms for these problems.

1 Introduction

The aim of *exact algorithms* is to optimally solve hard problems exponentially faster than brute-force search. The first papers in the area date back to the sixties and seventies [19,27]. For the last two decades the amount of literature devoted to this topic has been tremendous and it is impossible to give here a list of representative references without missing significant results. Recent surveys [14,21,26,29] provide a comprehensive information on exact algorithms. It is very natural to assume the existence of strong links between the area of exact algorithms and some areas of extremal combinatorics, especially the part of extremal combinatorics which studies the maximum (minimum) cardinalities of a system of subsets of some set satisfying certain properties. Strangely enough, there are not so many examples of such links in the literature, and the majority of exact algorithms are based on the so-called branching (backtracking) technique which traces back to the works of Davis, Putnam, Logemann, and Loveland [11,12].

In this paper, we prove a combinatorial lemma which appears to be very useful in the analysis of certain enumeration and exact algorithms. For a vertex

* This research was partially supported by the Research Council of Norway.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 210–221, 2008.

v of a graph G and integers $b, f \geq 0$, let $t(b, f)$ be the maximum number of connected induced subgraphs of G of size $b+1$ such that the intersection of all these subgraphs is nonempty and each such a subgraph has exactly f neighbors (a neighbor of a subgraph H is a vertex of $G \setminus H$ which is adjacent to a vertex of H). Then the combinatorial lemma states that $t(b, f) \leq \binom{b+f}{b}$ (and it is easy to check that this bound is tight). This can be seen as a variation of Bollobáss Theorem [7], which is one of the corner-stones in extremal set theory. (See Section 9.2.2 of [22] for detailed discussions on Bollobáss Theorem and its variants.)

We use this combinatorial result to obtain faster algorithm for a number of problems related to the treewidth of a graph. The treewidth is a fundamental graph parameter from Graph Minors Theory by Robertson and Seymour [25] and it has numerous algorithmic applications, see the surveys [4,6]. The problems to compute the treewidth is known to be NP-hard [1] and the best known approximation algorithm for treewidth has a factor $\sqrt{\log OPT}$ [13]. It is an old open question whether the treewidth can be approximated within a constant factor. Treewidth is known to be fixed parameter tractable. Moreover, for any fixed k, there is a linear time algorithm due to Bodlaender [3] computing the treewidth of graphs of treewidth at most k. Unfortunately, huge hidden constants in the running time of Bodlaender's algorithm is a serious obstacle to its implementation. For small values of k, the classical algorithm of Arnborg, Corneil and Proskurowski [1] from 1987 which runs in time $\mathcal{O}(n^{k+2})$ can be used to decide if the treewidth of a graph is at most k. The first exact algorithm computing the treewidth of an n-vertex graph is due to Fomin et al. [15] and has running time $\mathcal{O}(1.9601^n)$. Later these results were improved in [16,28] to $\mathcal{O}(1.8899^n)$. Both algorithms use exponential space. The fastest polynomial space algorithm for treewidth prior to this work is due to Bodlaender et al. [5] and runs in time $\mathcal{O}(2.9512^n)$.

Our results. We introduce a new (exponential space) algorithm computing the treewidth of a graph G on n vertices in time $\mathcal{O}(1.7549^n)$ and a polynomial space algorithm computing the treewidth in time $\mathcal{O}(2.6151^n)$. We also show that if the treewidth of G is at most k, then it can be computed in time $\mathcal{O}((\frac{2n+k+1}{3})^{k+1} \cdot kn^6)$. This is a refinement of the classical result of Arnborg et al. Running times of all these algorithms strongly depend on possibilities of fast enumeration of specific structures in a graph, namely, potential maximal cliques, and minimal separators [5,8,9,15,28]. The new combinatorial lemma is crucial in obtaining new combinatorial bounds and enumeration algorithms for minimal separators and potential maximal cliques, which, in turn, provides faster algorithms for treewidth.

Similar improvements in running times from $\mathcal{O}(1.8899^n)$ to $\mathcal{O}(1.7549^n)$ can be obtained for a number of results in the literature on problems related to treewidth (we skip definitions here). For example, by combining the ideas from [15] it is possible to compute the fill-in of a graph in time $\mathcal{O}(1.7549^n)$. Another example are the treelength and the Chordal Sandwich problem [24] which also can be solved in time $\mathcal{O}(1.7549^n)$ by making use of our technique.

The remaining part of the paper is organized as follows. In the next section we provide definitions and preliminary results. In Section 3, we prove our main combinatorial tool. By making use of this tool, in Section 4, we prove combinatorial bounds on the number of minimal separators and potential maximal cliques and obtain algorithm enumerating these structures. These results form the basis for all our algorithms computing the treewidth of a graph presented in Sections 5, 6, and 7. Some of the proofs have been removed due to space restrictions, for a full version see [17].

2 Preliminaries

We denote by $G = (V, E)$ a finite, undirected and simple graph with $|V| = n$ vertices and $|E| = m$ edges. For any non-empty subset $W \subseteq V$, the subgraph of G induced by W is denoted by $G[W]$. We say that a vertex set $S \subseteq V$ is *connected* if $G[S]$ is connected.

The *neighborhood* of a vertex v is $N(v) = \{u \in V : \ \{u, v\} \in E\}$ and for a vertex set $S \subseteq V$ we set $N(S) = \bigcup_{v \in S} N(v) \setminus S$. A *clique* C of a graph G is a subset of V such that all the vertices of C are pairwise adjacent.

Minimal separators. Let u and v be two non adjacent vertices of a graph $G = (V, E)$. A set of vertices $S \subseteq V$ is an *u, v-separator* if u and v are in different connected components of the graph $G[V \setminus S]$. A connected component C of $G[V \setminus S]$ is a *full* component associated to S if $N(C) = S$. S is a *minimal u, v-separator* of G if no proper subset of S is an u, v-separator. We say that S is a *minimal separator* of G if there are two vertices u and v such that S is a minimal u, v-separator. Notice that a minimal separator can be strictly included in another one. We denote by Δ_G the set of all minimal separators of G.

We need the following result due to Berry et al. [2] (see also Kloks et al. [23])

Proposition 1 ([2]). *There is an algorithm listing all minimal separators of an input graph G in $\mathcal{O}(n^3|\Delta_G|)$ time.*

The following proposition is an exercise in [18].

Proposition 2 (Folklore). *A set S of vertices of G is a minimal a, b-separator if and only if a and b are in different full components associated to S. In particular, S is a minimal separator if and only if there are at least two distinct full components associated to S.*

Potential maximal cliques. A graph H is *chordal* (or *triangulated*) if every cycle of length at least four has a chord, i.e. an edge between two non-consecutive vertices of the cycle. A *triangulation* of a graph $G = (V, E)$ is a chordal graph $H = (V, E')$ such that $E \subseteq E'$. H is a *minimal triangulation* if for any set E'' with $E \subseteq E'' \subset E'$, the graph $F = (V, E'')$ is not chordal.

A set of vertices $\Omega \subseteq V$ of a graph G is called a *potential maximal clique* if there is a minimal triangulation H of G such that Ω is a maximal clique of H. We denote by Π_G the set of all potential maximal cliques of G.

The following result on the structure of potential maximal cliques is due to Bouchitté and Todinca.

Proposition 3 ([8]). *Let $K \subseteq V$ be a set of vertices of the graph $G = (V, E)$. Let $\mathcal{C}(K) = \{C_1(K), \ldots, C_p(K)\}$ be the set of the connected components of $G[V \setminus K]$ and let $\mathcal{S}(K) = \{S_1(K), S_2(K), \ldots, S_p(K)\}$ where $S_i(K)$, $i \in \{1, 2, \ldots, p\}$, is the set of those vertices of K which are adjacent to at least one vertex of the component $C_i(K)$. Then K is a potential maximal clique of G if and only if:*

1. *$G[V \setminus K]$ has no full component associated to K, and*
2. *the graph on the vertex set K obtained from $G[K]$ by completing each $S_i \in \mathcal{S}(K)$ into a clique, is a complete graph.*

The following result is also due to Bouchitté and Todinca.

Proposition 4 ([8]). *There is an algorithm that, given a graph $G = (V, E)$ and a set of vertices $K \subseteq V$, verifies if K is a potential maximal clique of G. The time complexity of the algorithm is $\mathcal{O}(nm)$.*

Treewidth. A *tree decomposition* of a graph $G = (V, E)$ is a pair (χ, T) in which $T = (V_T, E_T)$ is a tree and $\chi = \{\chi_i | i \in V_T\}$ is a family of subsets of V such that: (1) $\bigcup_{i \in V_T} \chi_i = V$; (2) for each edge $e = \{u, v\} \in E$ there exists an $i \in V_T$ such that both u and v belong to χ_i; and (3) for all $v \in V$, the set of nodes $\{i \in V_T | v \in \chi_i\}$ forms a connected subtree of T. To distinguish between vertices of the original graph G and vertices of T, we call vertices of T *nodes* and their corresponding χ_i's *bags*. The maximum size of a bag minus one is called the *width* of the tree decomposition. The *treewidth* of a graph G, $tw(G)$, is the minimum width over all possible tree decompositions of G.

An alternative definition of treewidth is via minimal triangulations. The *treewidth* of a graph G is the minimum of $\omega(H) - 1$ taken over all triangulations H of G. (By $\omega(H)$ we denote the maximum clique-size of a graph H.)

Our algorithm for treewidth is based on the following result.

Proposition 5 ([15]). *There is an algorithm that, given a graph G together with the list of its minimal separators Δ_G and the list of its potential maximal cliques Π_G, computes the treewidth of G in $\mathcal{O}(n^3 (|\Pi_G| + |\Delta_G|)$ time. Moreover, the algorithm constructs an optimal triangulation for the treewidth.*

3 Combinatorial Lemma

The following lemma is our main combinatorial tool.

Lemma 1 (Main Lemma). *Let $G = (V, E)$ be a graph. For every $v \in V$, and $b, f \geq 0$, the number of connected vertex subsets $B \subseteq V$ such that*

(i) $v \in B$,
(ii) $|B| = b + 1$, *and*
(iii) $|N(B)| = f$

is at most $\binom{b+f}{b}$.

Proof. Let v be a vertex of a graph $G = (V, E)$. For $b + f = 0$ Lemma trivially holds. We proceed by induction assuming that for some $k > 0$ and every b and f such that $b + f \leq k - 1$, Lemma holds. For b and f such that $b + f = k$ we define $\mathcal{B}$ as the set of sets B satisfying $(i), (ii), (iii)$. We claim that

$$|\mathcal{B}| \leq \binom{b+f}{b}.$$

Since the claim always holds for $b = 0$, let us assume that $b > 0$.

Let $N(v) = \{v_1, v_2, \ldots, v_p\}$. For $1 \leq i \leq p$, we define $\mathcal{B}_i$ as the set of all connected subsets B such that

- Vertices $v, v_i \in B$,
- For every $j < i$, $v_j \notin B$,
- $|B| = b + 1$,
- $|N(B)| = f$.

Let us note, that every set B satisfying the conditions of the lemma is in some set $\mathcal{B}_i$ for some i, and that for $i \neq j$, $\mathcal{B}_i \cap \mathcal{B}_j = \emptyset$. Therefore,

$$|\mathcal{B}| = \sum_{i=1}^{p} |\mathcal{B}_i|. \tag{1}$$

For every $i > f + 1$, $|\mathcal{B}_i| = 0$ (this is because for every $B \in B_i$, the set $N(B)$ contains vertices $v_1, \ldots, v_{i-1}$ and thus is of size at least $f + 1$.) Thus (1) can be rewritten as follows

$$|\mathcal{B}| = \sum_{i=1}^{f+1} |\mathcal{B}_i|. \tag{2}$$

Let G_i be the graph obtained from G by contracting edge $\{v, v_i\}$ (removing the loop, reduce double edges to single edges, and calling the new vertex by v) and removing vertices $v_1, \ldots, v_{i-1}$. Then the cardinality of $\mathcal{B}_i$ is equal to the number of the connected vertex subsets B of G_i such that

- $v \in B$,
- $|B| = b$,
- $|N(B)| = f - i + 1$.

By the induction assumption, this number is at most $\binom{f+b-i}{b-1}$ and (2) yields that

$$|\mathcal{B}| = \sum_{i=1}^{f+1} |\mathcal{B}_i| \leq \sum_{i=1}^{f+1} \binom{f+b-i}{b-1} = \binom{b+f}{b}.$$

□

The inductive proof of the Main Lemma can be easily turned into a recursive polynomial space enumeration algorithm (we skip the proof here).

Lemma 2. *All vertex sets of size $b + 1$ with f neighbors in a graph G can be enumerated in time $\mathcal{O}(n\binom{b+f}{b})$ by making use of polynomial space.*

4 Combinatorial Bounds

In this section we provide combinatorial bounds on the number of minimal separators and potential maximal cliques in a graph. Both bounds are obtained by applying the Main Lemma on the respectice problems.

4.1 Minimal Separators

Theorem 1. *Let Δ_G be the set of all minimal separators in a graph G on n vertices. Then $|\Delta_G| = \mathcal{O}(1.6181^n)$.*

Proof. For $1 \leq i \leq n$, let $f(i)$ be the number of all minimal separators in G of size i. Then

$$|\Delta_G| = \sum_{1}^{n} f(i). \tag{3}$$

Let S be a minimal separator of size αn, where $0 < \alpha < 1$. By Proposition 2, there exists two full components C_1 and C_2 associated to S. Let us assume that $|C_1| \leq |C_2|$. Then $|C_1| \leq (1-\alpha)n/2$. From the definition of a full component C_1 associated to S, we have that $N(C_1) = S$. Thus, $f(\alpha n)$ is at most the number of connected vertex sets C of size at most $(1-\alpha)n/2$ with neighborhoods of size $|N(C)| = \alpha n$. Hence, to bound $f(\alpha n)$ we can use the Main Lemma for every vertex of G.

By Lemma 1, we have that for every vertex v, the number of full components of size $b+1 = (1-\alpha)n/2$ containing v and with neighborhoods of size αn is at most

$$\binom{b+\alpha n}{b} \leq \binom{(1+\alpha)n/2}{b}.$$

Therefore

$$f(\alpha n) \leq n \cdot \sum_{i=1}^{(1-\alpha)n/2} \binom{i+\alpha n}{i} < n \cdot \sum_{i=1}^{(1-\alpha)n/2} \binom{(1+\alpha)n/2}{i}. \tag{4}$$

For $\alpha \leq 1/3$, we have

$$\sum_{i=1}^{(1-\alpha)n/2} \binom{(1+\alpha)n/2}{i} < 2^{(1+\alpha)n/2} < 2^{2n/3} < 1.59^n,$$

and thus

$$\sum_{i=1}^{n/3} f(i) \ = \mathcal{O}(1.59^n). \tag{5}$$

For $\alpha \geq 1/3$, one can use the well known fact that the sum $\sum_{k=1}^{\lfloor j/2 \rfloor} \binom{j-k}{k}$ is equal to the $(j+1)$-st Fibonacci number to show that

$$\sum_{i=1}^{(1-\alpha)n/2} \binom{(1+\alpha)n/2}{i} < n \cdot \varphi^n,$$

where $\varphi = (1+\sqrt{5})/2 < 1.6181^n$ is the golden ratio.

Therefore,

$$\sum_{i=n/3}^{n} f(i) \ = \mathcal{O}(1.6181^n). \tag{6}$$

Finally, the theorem follows from the formulas (3),(5) and (6). □

4.2 Potential Maximal Cliques

Definition 1 ([8]). *Let Ω be a potential maximal clique of a graph G and let $S \subset \Omega$ be a minimal separator of G. We say that S is an* active separator for Ω, *if Ω is not a clique in the graph obtained from G by completing all the minimal separators contained in Ω, except S. A potential maximal clique Ω containing an active separator (for Ω) is called a* nice potential maximal clique.

We need the following result by Bouchitté and Todinca.

Proposition 6 ([9]). *Let Ω be a potential maximal clique of $G = (V, E)$, let u be a vertex of G, and let $G' = G[V \setminus \{u\}]$. Then one of the following holds:*

1. *Either Ω, or $\Omega \setminus \{u\}$ is a potential maximal clique of G';*
2. *$\Omega = S \cup \{u\}$, where S is a minimal separator of G;*
3. *Ω is a nice potential maximal clique.*

Let Π_n be the maximum number of nice potential maximal cliques that can be contained in a graph on n vertices. Proposition 6 is useful to bound the number of potential maximal cliques in a graph by the number of minimal separators Δ_G and Π_n.

Lemma 3. *For any graph $G = (V, E)$, $|\Pi_G| \leq n(n|\Delta_G| + \Pi_n)$.*

Proof. Let $v_1, v_2, ..., v_n$ be an ordering of V and let $V_i = \bigcup_{j=1}^{i} v_j$. The proof of the lemma follows from the following claim $\Pi_{G[V_{i+1}]} \leq \Pi_{G[V_i]} + n|\Delta_G| + \Pi_n$ which can be proved by making inductive use of Proposition 6. □

Definition 2. *Let $\Omega \in \Pi_G$, $v \in \Omega$, and C_v be the connected component of $G[V \setminus (\Omega \setminus \{v\})]$ containing v. We call the pair (C_v, v) by* vertex representation *of Ω.*

Lemma 4. *Let (C_v, v) be a vertex representation of Ω. Then $\Omega = N(C_v) \cup \{v\}$.*

Proof. By Proposition 3, every vertex $u \in \Omega \setminus \{v\}$, is either adjacent to v, or there exists a connected component C of $G[V \setminus \Omega]$ such that $u, v \in N(C)$. Since $C \subset C_v$, we have that $\Omega \setminus \{v\} \subseteq N(C_v)$. Every connected component C of $G[V \setminus \Omega]$ that contains $v \in N(C)$ is contained in C_v and $N(C) \subset \Omega$ for every C, therefore $\Omega \setminus \{v\} = N(C_v)$. □

We need also the following result from [28].

Proposition 7 ([28]). *Let Ω be a nice potential maximal clique of size αn in a graph G. There exists a vertex representation (C_v, v) of Ω such that $|C_v| \leq \lceil \frac{2(1-\alpha)n}{3} \rceil$.*

Now everything is settled to apply Main Lemma.

Lemma 5. *The number of nice potential maximal cliques in a graph $G = (V, E)$ is $\mathcal{O}(1.7549^n)$.*

Proof. By Proposition 7, for every nice potential maximal clique Ω of cardinality αn, there exists a vertex representation (C_v, v) of Ω such that $|C_v| \leq \lceil 2n(1-\alpha)/3 \rceil$. Let $b+1$ be the number of vertices in C_v. By Lemma 1, for every vertex v, the number of such pairs (C_v, v) is at most

$$\sum_{i=1}^{2(1-\alpha)n/3} \binom{(2+\alpha)n/3}{i}.$$

As in the proof of Theorem 1, for $\alpha \leq 2/5$ the above sum is $\mathcal{O}(1.7549^n)$. For $\alpha \geq 2/5$, by making use of the fact that $\sum_{k=1}^{\lfloor j/2 \rfloor} \binom{j-k}{2k}$ is equal to the $(j+1)$-st number of the sequence $\{a_i\}_{i=0}^{\infty}$ such that $a_i = 2a_{i-1} - a_{i-2} + a_{i-3}$, with $a_0 = 0$, $a_1 = 1$, and $a_2 = 2$, it is possible to show that the value of the above sum, and thus the number of nice potential maximal cliques, is $\mathcal{O}(1.7549^n)$. □

By combining Lemma 3, 5 and Theorem 1 we arrive at the main result of this subsection.

Theorem 2. *For any graph G, $|\Pi_G| = \mathcal{O}(1.7549^n)$.*

5 Exponential Space Exact Algorithm for Treewidth

Our algorithm computing the treewidth of a graph is based on Proposition 5. By making use of Proposition 5 we need to know how to list minimal separators and potential maximal cliques. By Proposition 1 and Theorem 1, all minimal separators can be listed in time $\mathcal{O}(1.6181^n)$. The proof of the following lemma is postponed till the full version of this paper.

Lemma 6. *For any graph G on n vertices, the set of potential maximal cliques can be listed in $\mathcal{O}(1.7549^n)$ time.*

As an immediate corollary of Proposition 1 and Lemma 6, we have the following result.

Theorem 3. *The treewidth of a graph on n vertices can be computed in time $\mathcal{O}(1.7549^n)$.*

6 Computing Treewidth at Most k

In this section we show how the lemma bounding the number of connected components can be used to refine the classical result of Arnborg et al. [1].

By Proposition 5, the treewidth of a graph can be computed in $\mathcal{O}(n^3\,(|\Pi_G| + |\Delta_G|))$ time if the list of all minimal separators Δ_G and the list of all potential maximal cliques Π_G of G are given. Actually, the results of Proposition 5 can be strengthened (with almost the same proof as in [16]) as follows. Let $\Delta_G[k]$ be the set of minimal separators and let $\Pi_G[k]$ be the set of potential maximal cliques of size at most k.

Lemma 7. *Given a graph G with sets $\Delta_G[k]$ and $\Pi_G[k+1]$, it can be decided in time $\mathcal{O}(n^3\,(|\Pi_G[k+1]|+|\Delta_G[k]|))$ if the treewidth of G is at most k. Moreover, if the treewidth of G is at most k, an optimal tree decomposition can be constructed within the same time.*

By Lemma 2 and Equation (4),

$$|\Delta_G[k]| \leq kn \cdot \sum_{i=1}^{(n-k)/2} \binom{(n+k)/2}{i} \leq kn^2 \cdot \binom{(n+k)/2}{k}, \tag{7}$$

and it is possible to list all vertex subsets containing all separators from $\Delta_G[k]$ in time $\mathcal{O}(kn^2 \cdot \binom{(n+k)/2}{k}))$. For each such a subset one can check in time $\mathcal{O}(n^2)$ if it is a minimal separator or not, and thus all minimal separators of size at most k can be listed in time $\mathcal{O}(kn^4 \cdot \binom{(n+k)/2}{k})$.

Let $\Pi_n[k]$ be the maximum number of nice potential maximal cliques of size at most k that can be in a graph on n vertices. By Proposition 7,

$$|\Pi_n[k]| \leq kn \cdot \sum_{i=1}^{(n-k)2/3} \binom{(2n+k)/3}{i} \leq kn^2 \cdot \binom{(2n+k)/3}{k},$$

and by making use of Proposition 4, all nice potential maximal cliques of size at most k can be listed in time $\mathcal{O}(kn^5 \cdot \binom{(2n+k)/3}{k})$.

Finally, we use nice potential maximal cliques and minimal separators of size k to generate all potential maximal cliques of size at most k.

Lemma 8. *For every graph G on n vertices, $|\Pi_G[k]| \leq n(|\Delta_G[k]| + \Pi_n[k])$ and all potential maximal cliques of G of size at most k can be listed in time $\mathcal{O}(kn^6 \cdot \binom{(2n+k)/3}{k})$.*

Proof. Let $v_1, v_2, ..., v_n$ be an ordering of V and let $V_i = \bigcup_{j=1}^{i} v_j$. By Proposition 6 and Lemma 3, every potential maximal clique of $G[V_i]$ either is a nice potential maximal clique of $G[V_i]$, or is a potential maximal clique of $G[V_{i-1}]$, or

is obtained by adding v_i to a minimal separator or a potential maximal clique of $G[V_{i-1}]$. This yields that $|\Pi_G[k]| \leq n(|\Delta_G[k]| + \Pi_n[k])$. To list all potential maximal cliques, for each i, $1 \leq i \leq n$, we list all minimal separators and nice potential maximal cliques in $G[V_i]$. This can be done in time $\mathcal{O}(kn^6 \cdot \binom{(2n+k)/3}{k})$. The total number of all such structures is at most $kn^3 \cdot \binom{(2n+k)/3}{k}$. By making use of dynamic programing, one can check if adding v_i to a minimal separator or potential maximal clique of $G[V_{i-1}]$ creates a potential maximal clique in $G[V_i]$, which by Proposition 4 can be done in time $\mathcal{O}(n^3)$. Thus, dynamic programming can be done in $\mathcal{O}(kn^6 \cdot \binom{(2n+k)/3}{k})$ steps. □

Now putting Lemma 7, Lemma 8 and Equation (7) together, we obtain the main result of this section.

Theorem 4. *There exists an algorithm that for a given graph G and integer $k \geq 0$, either computes a tree decomposition of G of the minimum width, or correctly concludes that the treewidth of G is at least $k+1$. The running time of this algorithm is $\mathcal{O}(kn^6 \cdot \binom{(2n+k+1)/3}{k+1}) = \mathcal{O}(kn^6 \cdot (\frac{2n+k+1}{3})^{k+1})$.*

7 Polynomial Space Exact Algorithm for Treewidth

The algorithm used in Proposition 1 requires exponential space because it is based on dynamic programming which keeps a table with all potential maximal cliques. As a consequence our $\mathcal{O}(1.7549^n)$ time algorithm for computing the treewidth also uses $\mathcal{O}(1.7549^n)$ space.

When restricting to polynomial space, we cannot store all the minimal separators and all the potential maximal cliques. The idea used to avoid this is to search for a "central" potential maximal clique or a minimal separator in the graph which can safely be completed into a clique. A similar idea is used in [5], however the improvement in the running time of our algorithm, is due to the following lemma and the technique used for listing minimal separators. Both results are, again, based on the Main Lemma.

Lemma 9. *For a given graph $G = (V, E)$ and $0 < \alpha < 1$, one can list in time $\mathcal{O}(mn^2 \cdot 2^{n(1-\alpha)})$ and polynomial space all potential maximal cliques of G such that for every potential maximal clique Ω from this list, there is a connected component of $G[V \setminus \Omega]$ of size at least αn.*

Proof. Let Ω be a potential maximal clique satisfying the conditions of the lemma, and let C be the connected component of size at least αn. By Proposition 3, $N(C)$ is a minimal separator contained in Ω and $\Omega \setminus N(C) \neq \emptyset$. Let (C_u, u) be a vertex representation of Ω, where $u \in \Omega \setminus N(C)$. Since u is not adjacent to any vertex in C, we have that $C_u \cap C = \emptyset$. To find Ω, we try to find its vertex representation by a connected vertex set such that the closed neighborhood of this set is of size at most $n(1-\alpha)$. By the Main Lemma, the number of such sets is at most

$$n \cdot \sum_{i=1}^{n(1-\alpha)} \binom{n(1-\alpha)}{i} = n \cdot 2^{n(1-\alpha)},$$

and by Lemma 2, all these sets can be listed in $\mathcal{O}(n \cdot 2^{n(1-\alpha)})$ steps and within polynomial space. Finally, for each set we use Lemma 4 and Proposition 4 to check in time $\mathcal{O}(mn)$ if the set is a potential maximal clique. □

We also use the following result which is a slight modification of the result from [5], where it is stated in terms of elimination orderings.

Proposition 8 ([5]). *For a given graph $G = (V, E)$ and a clique $K \subset V$, there exists a polynomial space algorithm, that computes the optimum tree decomposition (χ, T) of G, subject to the condition that the vertices of K form a bag which is a leaf of T. This algorithm runs in time $\mathcal{O}^*(4^{n-|K|})$.*

Theorem 5. *The treewidth of a graph $G = (V, E)$ can be computed in $\mathcal{O}(2.6151^n)$ time and polynomial space.*

Acknowledgement. We are grateful to Saket Saurabh for many useful comments, and to the anonymous referee pointing out that one of the bounds matched the golden ratio.

References

1. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods 8, 277–284 (1987)
2. Berry, A., Bordat, J.P., Cogis, O.: Generating all the minimal separators of a graph. Int. J. Found. Comput. Sci. 11, 397–403 (2000)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
4. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science 209, 1–45 (1998)
5. Bodlaender, H.L., Fomin, F.V., Koster, A.M.C.A., Kratsch, D., Thilikos, D.M.: On Exact Algorithms for Treewidth. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 672–683. Springer, Heidelberg (2006)
6. Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial Optimization on Graphs of Bounded Treewidth. The Computer Journal (to appear)
7. Bollobás, B.: On generalized graphs. Acta Math. Acad. Sci. Hungar, 447–452 (1965)
8. Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: Grouping the minimal separators. SIAM J. Comput. 31, 212–232 (2001)
9. Bouchitté, V., Todinca, I.: Listing all potential maximal cliques of a graph. Theor. Comput. Sci. 276, 17–32 (2002)
10. Buneman, P.: A characterization of rigid circuit graphs. Discrete Math. 9, 205–212 (1974)
11. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Comm. ACM 5, 394–397 (1962)
12. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. Assoc. Comput. Mach. 7, 201–215 (1960)
13. Feige, U., Hajiaghayi, M.T., Lee, J.R.: Improved approximation algorithms for minimum-weight vertex separators. In: STOC, pp. 563–572. ACM press, New York (2005)

14. Fomin, F., Grandoni, F., Kratsch, D.: Some new techniques in design and analysis of exact (exponential) algorithms. Bulletin of the European Association for Theoretical Computer Science 87, 47–77 (2005)
15. Fomin, F.V., Kratsch, D., Todinca, I.: Exact (exponential) algorithms for treewidth and minimum fill-in. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 568–580. Springer, Heidelberg (2004)
16. Fomin, F.V., Kratsch, D., Todinca, I., Villanger, Y.: Exact algorithms for treewidth and minimum fill-in. SIAM J. Comput. (accepted)
17. Fomin, F.V., Villanger, Y.: Treewidth computation and extremal combinatorics, arXiv:0803.1321v1
18. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
19. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of SIAM 10, 196–210 (1962)
20. Ho, C.-W., Lee, R.C.T.: Counting clique trees and computing perfect elimination schemes in parallel. Inform. Process. Lett. 31, 61–68 (1989)
21. Iwama, K.: Worst-case upper bounds for k-SAT. Bulletin of the European Association for Theoretical Computer Science 82, 61–71 (2004)
22. Jukna, S.: Extremal combinatorics with applications in computer science. Springer, Berlin (2001)
23. Kloks, T., Kratsch, D.: Listing all minimal separators of a graph. SIAM J. Comput. 27, 605–613 (1998)
24. Lokshtanov, D.: On the Complexity of Computing Treelength. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 276–287. Springer, Heidelberg (2007)
25. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. Journal of Algorithms 7, 309–322 (1986)
26. Schöning, U.: Algorithmics in Exponential Time. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 36–43. Springer, Heidelberg (2005)
27. Tarjan, R.E., Trojanowski, A.E.: Finding a maximum independent set. SIAM Journal on Computing 6, 537–546 (1977)
28. Villanger, Y.: Improved exponential-time algorithms for treewidth and minimum fill-in. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 800–811. Springer, Heidelberg (2006)
29. Woeginger, G.: 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)

Fast Scheduling of Weighted Unit Jobs with Release Times and Deadlines

C. Greg Plaxton*

University of Texas at Austin
1 University Station C0500
Austin, Texas 78712–0233
plaxton@cs.utexas.edu

Abstract. We present a fast algorithm for the following classic scheduling problem: Determine a maximum-weight schedule for a collection of unit jobs, each of which has an associated release time, deadline, and weight. All previous algorithms for this problem have at least quadratic worst-case complexity. This job scheduling problem can also be viewed as a special case of weighted bipartite matching: each job represents a vertex on the left side of the bipartite graph; each time slot represents a vertex on the right side; each job is connected by an edge to all time slots between its release time and deadline; all of the edges adjacent to a given job have weight equal to the weight of the job. Letting U denote the set of jobs and V denote the set of time slots, our algorithm runs in $O(|U| + k \log^2 k)$ time, where $k \leq \min\{|U|, |V|\}$ denotes the cardinality of a maximum-cardinality matching. Thus our algorithm runs in nearly linear time, a dramatic improvement over the previous quadratic bounds.

1 Introduction

We address the classic scheduling problem in which the input is a collection of jobs, each with an associated release time, deadline, and weight, and our objective is to schedule a maximum-weight subset of the jobs. For the reader who is familiar with Graham's notation for scheduling problems, this problem is denoted $(1 \mid r_j;\, p_j = 1 \mid \sum w_j U_j)$, where U_j is a 0-1 variable indicating whether job j is successfully scheduled. A number of special cases and variants of this scheduling problem have been studied for many years; see the scheduling survey of Graham et al. [8] for additional pointers to the early literature in this area. Various well-known algorithms textbooks include a detailed treatment of the special case in which all jobs are released at time zero, for which it is relatively straightforward to design a nearly linear time algorithm [1, pp. 207–214], [3, pp. 399–401], [9, pp. 161–168].

The unweighted version of the above scheduling problem is equivalent to a special case of bipartite matching called "convex" bipartite matching. Since we find it convenient to present our results in the matching framework, we now develop the necessary definitions. A subset A of a totally ordered set $(S, \leq)$ is said to be *convex* if the following condition holds for all x, y, and z in S: If x and y belong to A and $x \leq z \leq y$, then

* Supported by NSF Grants ANI–0326001 and CCF–0635203.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 222–233, 2008.

z belongs to A. A *convex bipartite graph* is a bipartite graph together with a bipartition (U, V) of the vertices, and a total order $\leq$ on the vertices of V, such that the neighbors of any vertex u in U form a convex subset of V.

Glover presented a simple greedy algorithm [7] for maximum-cardinality convex bipartite matching that admits an $O(|V| + |U| \log |U|)$-time implementation using an elementary priority queue data structure. Later, van Emde Boas used a fast priority queue to obtain an $O(|V| + |U| \log \log |U|)$-time implementation of Glover's algorithm [15]. Lipski and Preparata [11] used Tarjan's fast union-find data structure [14] to devise a different algorithm running in time $O(|U|+|V|\alpha(|V|))$, where α is a functional inverse of Ackermann's algorithm. Gabow and Tarjan [5] show that this application of union-find falls into a category admitting a linear-time implementation, thereby reducing the Lipski-Preparata time bound to $O(|U| + |V|)$. Another line of work focused on eliminating the dependence of the running time on $|V|$ [6,12]. This research culminated in the $O(|U|)$-time algorithm of Steiner and Yeomans [13].

The scheduling problem considered in this paper corresponds to the weighted variant of convex bipartite matching in which each vertex of U has an associated weight, and the weight of any edge is given by that of its endpoint in U. It is worth remarking that other weighted variants of the convex bipartite matching problem can be contemplated. One possibility is to allow each edge to have an arbitrary weight, but since the complete bipartite graph is convex, this weighted variant is equivalent to general weighted bipartite matching. A weighted variant of convex bipartite matching that is incomparable to the one considered in the present paper is obtained by associating a weight with each vertex in V, and defining the weight of each edge as that of its endpoint in V. For this "right-weighted" (i.e., V-weighted) variant, Katriel [10] has recently obtained an $O(|E| + |V| \log |U|)$-time algorithm to find a maximum-weight matching. Since the input size is $\Theta(|U| + |V|)$ words, and $|E|$ could be as large as $\Theta(|U| \cdot |V|)$, this algorithm has quadratic complexity. Another weighted variant of convex bipartite matching — this time more general than the one considered in the present paper — is obtained by associating a weight with each vertex in $U \cup V$, and defining the weight of each edge as the sum of the weights of its two endpoints. If the input graph admits a U-perfect matching, Katriel's $O(|E|+|V| \log |U|)$-time algorithm can be used to find a maximum-weight $|U|$-perfect matching. Since many bipartite graphs — including, for example, all bipartite graphs such that $|U| > |V|$ — do not admit a U-perfect matching, Katriel's algorithm addresses only a special case of this weighted variant.

Throughout the remainder of this paper, we focus on the "left-weighted" (i.e., U-weighted) variant of convex bipartite matching that corresponds to scheduling weighted unit jobs with release times and deadlines. Any left-weighted (or right-weighted) bipartite matching instance is well-known to form a matroid, where the independent sets of the matroid are those subsets U' of vertices on the left such that there exists a matching M that matches every vertex in U'. Thus, left-weighted convex bipartite matching can be addressed using the framework of the matroid greedy algorithm. Indeed, all efficient algorithms for left-weighted convex bipartite matching that have been proposed to date, including the algorithm presented in this paper, exploit this framework. In the next paragraph, we describe a simple $O(|U| \log |U| + |U| \cdot |V|)$-time algorithm of this sort. This algorithm, which we refer to in this paper as the *greedy algorithm*, produces a matching

that we call the *greedy matching*. The fast matching algorithm presented in this paper, which we refer to as the *hierarchically greedy algorithm*, maintains a hierarchical representation of a collection of matchings that includes the greedy matching. We establish the correctness of the hierarchically greedy algorithm by relating its behavior to that of the greedy algorithm.

Here is a description of the greedy algorithm. First, we sort the vertices in U in nonincreasing order of weight, initialize an independent set Z to the empty set, and initialize a matching M to the empty matching. Then, we iteratively attempt to grow the independent set Z by considering each successive vertex u in U (according to the sorted order previously determined) and adding it to Z if the resulting set remains independent. In order to simplify the task of determining whether u can be added to Z, we maintain the invariant that M is the greedy matching of Z that is produced by Glover's unweighted convex bipartite matching algorithm. Glover's algorithm attempts to match the vertices v in $|V|$, from lowest to highest, using a natural "earliest deadline" rule that the vertex in Z matched to v is the as yet unmatched vertex adjacent to v (if any) that has the smallest number of remaining opportunities to be matched. (If a tie occurs, it is broken according to a fixed ordering of the vertices in U.) Using a naive representation of the greedy matching M in the form of an array of length V, in $O(|V|)$ time we can determine whether Glover's algorithm successfully matches all vertices in $I \cup \{u\}$, and if so, update Z and M appropriately. Upon termination of the greedy algorithm, the greedy matching M is a maximum-weight matching.

Lipski and Preparata [11] use the matroid greedy framework to develop a left-weighted convex bipartite matching algorithm which, while somewhat different from the greedy algorithm described above, has a similar time complexity of $O(|U|^2 + |U| \cdot |V|)$. Dekel and Sahni [4] present a parallel algorithm for left-weighted convex bipartite matching that uses $O(|U|^2)$ processors and $O(\log^2 |U|)$ time, and which is based on a sequential algorithm with $O(|U|^2)$ complexity.

In this paper, we introduce a data structure that maintains a hierarchical representation of a collection of matchings that includes the greedy matching. This data structure allows us to implement each iteration of the matroid greedy algorithm in amortized polylogarithmic time. As a result, we obtain a nearly linear time algorithm for left-weighted convex bipartite matching. The remainder of this paper is organized as follows. Section 2 contains some preliminary definitions. Section 3 presents an efficient dynamic data structure for a special class of convex bipartite graphs. Section 4 presents our "hierarchically greedy" matching algorithm. Section 5 discusses the time complexity of this algorithm. Section 6 offers some concluding remarks.

2 Preliminaries

In this section, we specify the formal representation of a convex bipartite graph, or CBG, that will be used throughout the remainder of the paper. We also define certain special kinds of CBGs.

Instead of working with "jobs" and "time slots", we find it convenient to introduce more abstract types "ping" and "pong", which we define as follows. A *pong* is an element of some totally ordered universe, such as the integers. A *ping* u is characterized by

four attributes: pongs $u.\mathit{first}$ and $u.\mathit{last}$, a positive weight $u.\mathit{weight}$, and a unique integer ID $u.\mathit{id}$. We define three total orders over the set of all pings: under the *first-ID total order*, ping u is at most u' if $(u.\mathit{first}, u.\mathit{id})$ is lexicographically at most $(u'.\mathit{first}, u'.\mathit{id})$; under the *last-ID total order*, ping u is at most u' if $(u.\mathit{last}, u.\mathit{id})$ is lexicographically at most $(u'.\mathit{last}, u'.\mathit{id})$; under the *weight-ID total order*, ping u is at most u' if $(u.\mathit{weight}, u.\mathit{id})$ is lexicographically at most $(u'.\mathit{weight}, u'.\mathit{id})$. We primarily make use of the last-ID total order. For this reason, we adopt the convention that all ping comparisons are resolved according to the last-ID total order unless stated otherwise.

A pair (U, V), where U is a set of pings and V is a set of pongs, represents a CBG as follows: (1) we identify each ping in U with a vertex on the LHS; (2) we identify each pong in V with a vertex on the RHS; (3) there is an edge from ping u to pong v if and only if $u.\mathit{first} \leq v \leq u.\mathit{last}$.

A CBG (U, V) is *proper* if $\{u.\mathit{first}, u.\mathit{last}\} \subseteq V$ for all pings u in U. A CBG (U, V) is *simple* if $|U| \geq |V|$ and $v \leq u.\mathit{last}$ for all pings u in U and all pongs v in V. A CBG (U, V) is *nice* if $|U| \leq |V| + 1$ and (U, V) is simple and admits a matching of cardinality $|V|$. A nice CBG (U, V) with $|U| = |V|$ is said to be *in-kilter*; otherwise, it is *out-of-kilter*.

3 A Dynamic Data Structure for Nice CBGs

In this section, we develop a dynamic data structure for maintaining a nice CBG subject to a collection of six operations. Three of these operations are applicable when the nice CBG is in an in-kilter state; the other three are applicable in out-of-kilter states.

The three in-kilter operations are as follows. The first is *pingAdd*(u), where ping u does not belong to U and $(U \cup \{u\}, V)$ is simple; this operation adds the ping u to U. The second is *pongDrop*(v), where pong v belongs to V; this operation removes the pong v from V. The third is *print*$()$, which prints out a perfect matching of (U, V).

The three out-of-kilter operations are as follows. The first is *pongAdd*(v), where pong v does not belong to V and $(U, V \cup \{v\})$ is simple and admits a perfect matching; this operation adds the pong v to V. The second is *pingDrop*(u), where ping u belongs to U and $(U \setminus \{u\}, V)$ admits a perfect matching; this operation removes the ping u from U. The third is *pingDrop*$()$, which takes no arguments; this operation removes from U the maximum ping u in U such that $(U \setminus \{u\}, V)$ admits a perfect matching.

Notice that the precondition of each of the above operations ensures that the CBG remains nice. Our goal is to implement the *print*$()$ operation in linear time, and each of the other five operations in logarithmic time. The only significant challenge is to implement the *pingDrop*$()$ operation efficiently. To do so, we first find it useful to introduce a few definitions and lemmas related to simple CBGs.

For any simple CBG (U, V), we define the following auxiliary functions: (1) let $A(U, V)$ denote $\{u.\mathit{first} \mid u \in U\} \cup V$; (2) for any pong v in $A(U, V)$, let $f(U, V, v)$ denote the number of pings u in U such that $v \leq u.\mathit{first}$ minus the number of pongs v' in V such that $v \leq v'$; (3) let $g(U, V)$ denote the maximum, over all pongs v in $A(U, V)$, of $f(U, V, v)$ (if $A(U, V)$ is empty, then $g(U, V)$ is zero). The proof of the following lemma is straightforward and is omitted.

Lemma 1. *For any simple CBG (U, V), the size of a maximum cardinality matching is $|U| - g(U, V)$.*

For any simple CBG (U, V) such that $A(U, V)$ is nonempty, let us define $h(U, V)$ as the maximum pong in $A(U, V)$ such that $g(U, V) = f(U, V, v)$. The proof of the next lemma follows easily from Lemma 1.

Lemma 2. *For any simple CBG (U, V) such that $|U| > |V|$, and any ping u in U, the size of a maximum cardinality matching of (U, V) is equal to that of $(U \setminus \{u\}, V)$ if and only if $h(U, V) \leq u.$*first.

Lemma 2 leads to the following useful characterization of the set of pings over which the maximization occurs in the definition of *pingDrop*().

Lemma 3. *For any out-of-kilter nice CBG (U, V), and any ping u in U, $(U \setminus \{u\}, V)$ admits a perfect matching if and only if $h(U, V) \leq u.$*first.

Lemma 3 suggests a two-phase approach for implementing *pingDrop*(): (1) compute the pong $h(U, V)$; (2) compute the maximum ping u in U such that $h(U, V) \leq u.$*first*.

We now discuss how to implement the first phase in logarithmic time. By Lemma 1, for any out-of-kilter nice CBG (U, V), we have $g(U, V) = 1$, and hence $h(U, V)$ is the maximum pong in $A(U, V)$ such that $f(U, V, v) = 1$. To implement the first phase, we maintain an augmented red-black tree with a node for each pong v in $A(U, V)$; the key of each node is the associated pong. We augment a node x by maintaining the following three auxiliary fields: an integer "count" field equal to $|\{u \in U \mid u.\mathit{first} = v\}| - \Delta$, where v is the key of node x and Δ is equal to 1 if v belongs to V, and 0 otherwise; an integer "sum" field that is equal to the sum of all count fields in the nodes of the subtree rooted at x; an integer "maximum suffix sum" field that is equal to the maximum, over all suffixes (including the empty suffix) of the key-ordered sequence of nodes in the subtree rooted at x, of the sum of the counts in the suffix. It is straightforward to argue that all of these fields can be maintained in logarithmic time whenever a ping is added to or removed from U, and whenever a pong is added to or removed from V. Furthermore, given the augmented red-black tree structure that we have just described, it is straightforward to determine the maximum pong in $A(U, V)$ such that $f(U, V, v) = 1$ in logarithmic time, and hence to implement the first phase in logarithmic time.

To implement the second phase, we maintain a second augmented red-black tree with a node for each ping in U. The nodes are sorted according to the first-ID ordering, that is, the key associated with the node for a ping u is $(u.\mathit{first}, u.\mathit{id})$. We augment each red-black tree node with a "max" field equal to the maximum ping (with respect to the last-ID ordering) in the nodes of the corresponding subtree. It is straightforward to maintain the max fields in logarithmic time whenever a ping is added to or removed from U. Furthermore, given this tree, and the pong threshold $h(U, V)$ determined in the first phase, it is straightforward to implement the second phase in logarithmic time.

As remarked in the preceding paragraphs, it is straightforward to maintain our two augmented red-black trees in logarithmic time whenever the nice CBG is modified via an operation that adds or drops a ping or pong. It remains to describe how to implement the *print*() operation in linear time. One simple approach is to maintain a third red-black

tree containing the pongs of V. A perfect matching of an in-kilter nice CBG can then be obtained by matching each ping in the second augmented red-black tree described above (the one sorted by the first-ID ordering) to the pong of equal rank in the third tree. In fact, it is not necessary to maintain such a third red-black tree, because we can use inorder traversals of the first and second red-black trees to produce a sorted list of the pongs in V in linear time. Thus our data structure for maintaining a dynamic nice CBG consists of just two augmented red-black trees. (Remark: The implementation of the *print*() operation described above does not, in general, produce the greedy matching. If we wish to produce the greedy matching, we can do so in $O(|U| \log |U|)$ time via a suitable linear sequence of calls to the *pongDrop*(v) and *pingDrop*() operations. To avoid modifying the data structure, we can first create a copy in linear time.)

4 A Hierarchically Greedy Algorithm

In this section we present a hierarchically greedy algorithm for computing a maximum-weight matching of a given CBG (U, V). It is convenient to assume that (U, V) is proper. In the context of establishing the upper bound of Theorem 1, our assumption that (U, V) is proper is made without loss of generality, since we can easily preprocess (U, V) in $O(|U| \log |U| + |V| \log |V|)$ time to obtain an equivalent CBG — with respect to the maximum-weight matchings — that is proper. The preprocessing phase removes all pings with degree zero, and for each of the remaining pings u, assigns $u.\mathit{first}$ to the minimum pong v in V such that $u.\mathit{first} \leq v$, and assigns $u.\mathit{last}$ to the maximum pong v in V such that $v \leq u.\mathit{last}$.

Like the greedy algorithm described in Section 1, our hierarchically greedy algorithm is based on the framework of the matroid greedy algorithm. As such, the algorithm iterates through the pings in decreasing order with respect to the weight-ID ordering. While the greedy algorithm maintains a specific matching — the greedy matching — at each iteration, our hierarchically greedy algorithm maintains a representation of a collection of matchings that includes the greedy matching. We say that an iteration of the greedy algorithm is successful if it adds a ping to the greedy matching; otherwise, it is unsuccessful. It turns out that, at any given iteration, all of the matchings in the collection maintained by the hierarchically greedy algorithm induce the same set of matched pings and pongs. It follows that an iteration of the hierarchically greedy algorithm successfully inserts the current ping into the set of matched pings if and only if the corresponding iteration of the greedy algorithm is successful.

It remains to describe how the hierarchically greedy algorithm performs the insertion attempt associated with a general iteration of the matroid greedy algorithm. It is straightforward to prove that the attempt to insert a ping u in a given iteration is unsuccessful if and only if there exist pongs v and v' in V such that $v \leq u.\mathit{first} \leq u.\mathit{last} \leq v'$ and the number of pings u' previously successfully inserted for which $v \leq u.\mathit{first} \leq u.\mathit{last} \leq v'$ is equal to the number of pongs in V that belong to the interval $[v, v']$. We refer to such an interval of pongs $[v, v']$ as a *tight interval*, and we say that a pong in V is *tight* if it belongs to some tight interval. Initially, none of the pongs in V are tight, but as more pings are successfully inserted, certain pongs become (and remain) tight.

Our hierarchically greedy algorithm maintains a conservative estimate (i.e., a subset) of the current set of tight pongs in V. The pongs associated with this estimate are said to be *marked tight*. Once a pong is marked tight, it continues to be marked tight thereafter. When we attempt to insert a ping u, we first determine whether all of the pongs v in V such that $u.first \leq v \leq u.last$ are marked tight. If so, we conclude that the insertion is unsuccessful, and proceed to the next iteration. We call such an unsuccessful insertion *good* because it is relatively inexpensive to process; other unsuccessful insertions are said to be *bad*.

4.1 An Augmented Binary Search Tree

Our hierarchical greedy algorithm makes use of an augmented BST with $|V|$ nodes, one for each pong in V. The key of each node α in the augmented BST, which is denoted $\alpha.key$, is the associated pong. Since V is not updated through the course of the algorithm, the structure of the augmented BST is static. In analyzing the performance of the algorithm, we assume only that the augmented BST has depth $O(\log |V|)$. We maintain several additional fields for each node α in the augmented BST: $\alpha.left$, which is a pointer to the left child of α ($\alpha.left = 0$ if there is no left child); $\alpha.right$, which is a pointer to the right child of α ($\alpha.right = 0$ if there is no right child); $\alpha.parent$, which is a pointer to the parent of α ($\alpha.parent = 0$ if α is the root); $\alpha.keyMin$, which is equal to the minimum, over all nodes β in the subtree rooted at α, of $\beta.key$; $\alpha.size$, which is equal to the total number of nodes in the subtree rooted at α; $\alpha.occupant$, which is of type "pointer to node", and which either points to some ancestor of α, or takes on one of two special values 0 and $+\infty$; $\alpha.occupantMax$, which is equal to the "maximum", over all nodes β in the subtree rooted at α, of $\beta.occupant$. In order to make the latter definition precise, we need to specify how to determine the maximum of two node pointers. We consider the special pointer value $+\infty$ to be the highest possible pointer value, and the special pointer value 0 to be the lowest possible pointer value. To compare two pointers to actual nodes α and β, we instead compare $\alpha.size$ with $\beta.size$, breaking ties arbitrarily.

Each node of the augmented BST also contains a nice CBG that is represented by two augmented red-black trees as discussed in Section 3. These nice CBGs are such that the total size of the augmented BST remains linear throughout.

As indicated earlier, the structure of the augmented BST is static. All occupant fields (and hence also the occupantMax fields) are initialized to $+\infty$ to indicate that all of the pongs are unmatched. The nice CBG associated with each node is initialized to the empty CBG. After initialization, the only attributes of a node that are subject to modification are the occupant and occupantMax fields, and the associated nice CBG.

4.2 Key Invariant

Due to space limitations, our proof of correctness is not included in this extended abstract. Instead we simply state a key invariant of the data structure, which holds in any *quiescent state*, that is, before and after any insertion attempt. The invariant completely characterizes the state of the augmented BST in terms of the set of pongs in V that have been marked tight and the state of the greedy algorithm after the same number of insertion attempts. To define the invariant, we consider executing the greedy and hierarchically greedy algorithms side by side, one insertion attempt at a time. In any quiescent

state, we define a mapping from the edges of the greedy matching to the nodes of the augmented BST as follows: Map edge (u, v) to the LCA of the node with key v and the node with key $u.last$. We claim that the set of pings (resp., pongs) of the nice CBG associated with any node α is exactly the set of ping (resp., pong) endpoints of the edges of the greedy matching that are mapped to node α. The preceding claim characterizes the set of $|V|$ nice CBGs associated with the nodes of our augmented BST in this quiescent state. It remains only to characterize the values of the occupant fields, since the occupant values determine the occupantMax values, and all of the other augmented BST fields are static. For any pong v in V, let α denote the augmented BST node with $\alpha.key = v$. If pong v is unmatched in the greedy matching, then $\alpha.occupant = +\infty$. If pong v is matched in the greedy matching and is marked tight, then $\alpha.occupant = 0$. If v is matched to ping u in the greedy matching and is not marked tight, then $\alpha.occupant$ is a pointer to the augmented BST node to which edge (u, v) is mapped, i.e., to the LCA of the node with key v and the node with key $u.last$.

The aforementioned invariant asserts a close correspondence between the successive quiescent states of the greedy and hierarchically greedy algorithms. The main idea underlying our full proof of correctness is to extend this correspondence to non-quiescent states. We do so by starting with the greedy algorithm and transforming it into the hierarchically greedy algorithm via a sequence of code transformations. A suitable correspondence is established between each pair of successive algorithms in this sequence.

4.3 Methods

In this section we describe the methods supported by the node object of the augmented BST introduced in Section 4.1.

The print method is invoked on the root node at the end of the hierarchically greedy algorithm in order to output a maximum-weight matching. When invoked on the root node, the print method traverses each node α of the augmented BST, and invokes the print operation of Section 3 on the nice CBG associated with α. Each of these $|V|$ print operations produces a piece of the overall output matching. The invariant stated in Section 4.2 ensures that the nice CBG associated with any augmented BST node is in-kilter in any quiescent state, and hence the nice CBG print operation is applicable in any such state. The overall running time of the print method is $O(|U| + |V|)$. (If we wish to produce the greedy matching, then we need to print the greedy matching of each nice CBG; by the analysis of the print operation in Section 3, this increases the running time by an $O(\log k)$ factor, where $k \leq \min\{|U|, |V\}$ denotes the maximum number of pings/pongs in any of the nice CBGs.)

The second major node method is insert. Pseudocode for the insert method is provided below. Each of the $|U|$ insertion attempts performed by the hierarchically greedy algorithm corresponds to an invocation of this method at the root of the augmented BST. The lone argument of the insert method is the ping associated with the current insertion attempt. The insert method is defined in terms of three other node methods: add, tight, and tighten. We discuss each of these three methods below.

insert(Ping u)
  Pong $v := u.first$

```
if tight(v, u.last) = 0 then
    Node *p := add(u, v)
    if p ≠ 0 then
        tighten(v, p →key)
```

The tight method takes two pong arguments v and v', and returns a boolean value indicating whether every pong v'' in V such that $v \leq v'' \leq v'$ has been marked tight. By exploiting the occupantMax field of the augmented BST, this method is easy to implement in $O(\log |V|)$ time. But to establish our best time bound for left-weighted convex bipartite matching, we need a more efficient implementation of the tight method, so we maintain a separate union-find data structure [14]. The idea is to break the sorted sequence of pongs in V into maximal contiguous subsequences of pongs in which either: (1) all of the pongs in a subsequence have been marked tight, or (2) none of the pongs in a subsequence have been marked tight. Each such subsequence corresponds to a single set in our union-find data structure. Along with each set, we store a bit indicating whether the pongs in the set have been marked tight. The roots of the sets are linked together in a doubly-linked list, which is kept in sorted order based on the pong values. To implement the tight method, we simply look up the two pong arguments in the union-find data structure to determine whether they both belong to the same set, and if so, whether that set consists of pongs that have been marked tight. In the foregoing pseudocode for the insert method, we use a call to the tight method to determine whether to immediately reject a given insertion attempt.

The tighten method takes two pong arguments v and v'. For every pong v'' in V such that $v \leq v'' \leq v'$, this method marks v'' tight by ensuring that the occupant field in the node with key v'' is equal to zero (i.e., if it was not already zero, this method sets it to zero), and makes any necessary adjustments to the occupantMax fields. This method also performs corresponding unions to the union-find data structure mentioned in the preceding paragraph. (The sorted doubly-linked list of roots in the union-find data structure enables us to perform these union operations efficiently.) By exploiting the occupantMax field (i.e., there is no need for this method to descend into a subtree rooted at a node with occupantMax field equal to zero), we can easily execute an invocation of the tighten method in $O((k+1)\log |V|)$ time, where k denotes the number of nonzero occupant fields that are set to zero as a result of the invocation. In the above pseudocode for the insert method, we invoke the tighten method on the root node of the augmented BST to update our data structures after a bad unsuccessful insertion attempt. Accordingly, the quantity k in the preceding expression is guaranteed to be positive on any invocation of the tighten method. Furthermore, once we set the occupant field of some node to zero in tighten, it remains zero thereafter. It follows from the foregoing remarks that the total cost of all invocations of the tighten method is $O(|V| \log |V|)$.

The pseudocode for the add method is given below. If the insertion attempt associated with the add invocation is successful, add returns 0. Otherwise, add indicates failure by returning a pointer to a node in the augmented BST; this pointer is used to determine the second pong argument of the subsequent invocation of the tighten method. Recall that each augmented BST node has an associated nice CBG. In the pseudocode for the add method, the calls to *pingAdd*(u) and *pingDrop*(u) act on the associated nice CBG. The same holds for the calls to *pongAdd*(v), *pingDrop*$()$, and *pingAdd*(u) appearing

in the pseudocode for the resolve method below, and for the calls to *pongDrop*(v) and *pongAdd*(v) appearing in the pseudocode for the drop method below. Our proof of correctness establishes that each call acting on the associated nice CBG satisfies the required precondition specified in Section 3. The add method is defined in terms of the node method resolve, which we discuss next.

```
Node *add(Ping u, Pong v)
  if occupantMax = 0 then
    return this
  else if v > key then
    return right→add(u, v)
  else if u.last < key then
    return left→add(u, v)
  pingAdd(u)
  p := resolve(v)
  if p ≠ 0 then
    pingDrop(u)
  return p
```

The pseudocode for the resolve method is given below. Like the add method, resolve returns 0 on success, and a pointer to a node on failure. The resolve method is defined in terms of the node methods add, drop, occupy, and search. We have already defined the add method; drop, occupy, and search are defined below.

```
Node *resolve(Pong v)
  Node *p, *q := search(v)
  if q ≠ 0 then
    Pong v := q →key
    Node *r := q →occupant
    p := if r = +∞ then 0 else r →drop(v)
    if p = 0 then
      pongAdd(v)
      occupy(q)
  else
    Ping u := pingDrop()
    p := if key = u.last then this else right→add(u, right→keyMin)
    if p ≠ 0 then
      pingAdd(u)
  return p
```

When the search method is invoked at a node α, it takes as argument a pong v that is either equal to $\alpha.key$, or is equal to $\beta.key$ for some node β in the left subtree of α. The search method returns a pointer to a node, determined as follows. Let S denote the set of all nodes β in the subtree rooted at α such that $v \leq \beta.key \leq \alpha.key$, and $\beta.occupant$ is either $+\infty$ or the address of a proper ancestor of α. If S is empty, then 0 is returned. Otherwise, a pointer to the node β in S minimizing $\beta.key$ is returned. By making use

of the occupantMax field, it is straightforward to implement the search method in time proportional to the depth of the subtree rooted at node α.

When the occupy method is invoked at a node α, it takes as argument a nonzero pointer to a node returned by a just-completed call to the search method at node α. As such, the argument of the occupy method is a pointer to some β in the subtree rooted at node α such that $\beta.occupant$ is either $+\infty$ or the address of a proper ancestor of α. The occupy method sets $\beta.occupant$ to point to α, and then updates occupantMax fields as necessary, starting at β and repeatedly "bubbling up" to the parent until the occupantMax field is unchanged or we attempt to move to the parent of the root. The running time is proportional to the depth of the subtree rooted at node α.

The pseudocode for the drop method is given below. Like the add method, drop returns 0 on success, and a pointer to a node on failure. The drop method is defined in terms of the resolve method discussed above.

```
Node *drop(Pong v)
    pongDrop(v)
    Node *p := resolve(v)
    if p ≠ 0 then
        pongAdd(v)
    return p
```

5 Analysis

Due to space limitations, we are not able to include our analysis of the running time of the hierarchically greedy algorithm in this extended abstract. In the full version of the paper, we prove the following theorem. The proof uses separate arguments to bound the total cost of the successful, good unsuccessful, and bad unsuccessful insertions.

Theorem 1. *The hierarchically greedy algorithm of Section 4 computes a maximum-weight matching of a given CBG (U, V) in $O(|U| \log |U| + |V| \log^2 |V|)$ time using $O(|U| + |V|)$ space.*

With some extra preprocessing work, we can further improve the running time of the algorithm of Section 4. By employing a preprocessing phase based on the $O(|U|)$-time unweighted convex bipartite matching algorithm of Steiner and Yeomans [13], the $O(|V| \log^2 |V|)$ term can be improved to $O(k \log^2 k)$, where $k \leq \min\{|U|, |V|\}$ denotes the size of a maximum-cardinality matching. The following theorem can then be obtained via an $O(|U| + k \log^2 k)$-time preprocessing phase that discards all but $O(k \log k)$ of the pings in U, while ensuring that we can still produce a maximum-weight matching using the remaining pings. The complete details of the two preprocessing phases are nontrivial and are provided in the full paper.

Theorem 2. *A maximum-weight matching of a proper CBG (U, V) can be computed in $O(|U| + k \log^2 k)$ time and $O(|U| + |V|)$ space, where $k \leq \min\{|U|, |V|\}$ denotes the size of a maximum-cardinality matching.*

Notice that in order to achieve the time bound of Theorem 2 we need to assume — as in the work of Steiner and Yeomans [13] — that the input CBG (U, V) is proper.

6 Concluding Remarks

Recently, Brodal et al. [2] have designed a data structure based on the Dekel-Sahni algorithm for the problem of maintaining an unweighted maximum matching in a dynamic convex bipartite graph. It allows for vertices to be inserted or deleted from either U or V in $O(\log^2 |U|)$ amortized time. The interface supported by the data structure includes a constant-time "status query" that can be used to determine whether a given vertex is in the current maximum matching. Also, a "pair query" is provided that takes as argument a matched vertex and returns the current match of that vertex. The amortized cost of a pair query is shown to be $O(\sqrt{|U|}\log^2 |U|)$ and $\Omega(\sqrt{|U|})$. We plan to investigate whether the techniques of the present paper can be used to obtain improved bounds for some of the dynamic operations considered in [2].

References

1. Brassard, G., Bratley, P.: Fundamentals of Algorithmics. Prentice Hall, Englewood Cliffs (1996)
2. Brodal, G.S., Georgiadis, L., Hansen, K.A., Katriel, I.: Dynamic matchings in convex bipartite graphs. In: Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science, pp. 406–417 (August 2007)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, McGraw-Hill, Cambridge (2001)
4. Dekel, E., Sahni, S.: A parallel matching algorithm for convex bipartite graphs and applications to scheduling. Journal of Parallel and Distributed Computing 1, 185–205 (1984)
5. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. Journal of Computer and System Sciences 30, 209–221 (1985)
6. Gallo, G.: An $O(n \log n)$ algorithm for the convex bipartite matching problem. Operations Research Letters 3, 313–316 (1984)
7. Glover, F.: Maximum matching in convex bipartite graphs. Naval Research Logistic Quarterly 14, 313–316 (1967)
8. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 287–326 (1979)
9. Horowitz, E., Sahni, S.: Fundamentals of Computer Algorithms. Computer Science Press, New York (1978)
10. Katriel, I.: Matchings in node-weighted convex bipartite graphs. INFORMS Journal on Computing (December 2007); Published online in Articles in Advance (print version to in appear, 2008)
11. Lipski Jr., W., Preparata, F.P.: Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. Acta Informatica 15, 329–346 (1981)
12. Scutellà, M.G., Scevola, G.: A modification of Lipski-Preparata's algorithm for the maximum matching problem on bipartite convex graphs. Ricerca Operativa 46, 63–77 (1988)
13. Steiner, G., Yeomans, J.S.: A linear time algorithm for determining maximum matchings in convex, bipartite graphs. Computers and Mathematics with Applications 31, 91–96 (1996)
14. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. Journal of the ACM 22, 215–225 (1975)
15. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. Information Processing Letters 6, 80–82 (1977)

Approximation Algorithms for Scheduling Parallel Jobs: Breaking the Approximation Ratio of 2*

Klaus Jansen and Ralf Thöle

Institut für Informatik, Universität zu Kiel, Germany
{kj,rth}@informatik.uni-kiel.de

Abstract. In this paper we study variants of the non-preemptive parallel job scheduling problem where the number of machines is polynomially bounded in the number of jobs. For this problem we show that a schedule with length at most $(1+\varepsilon)\,\mathrm{OPT}$ can be calculated in polynomial time, which is the best possible result (in the sense of approximation ratio), since the problem is strongly NP-hard.

For the case when all jobs must be allotted to a subset of machines with consecutive indices a schedule with length at most $(1.5+\varepsilon)\,\mathrm{OPT}$ can be calculated in polynomial time. The previously best known results are algorithms with absolute approximation ratio 2.

1 Introduction

In classical scheduling theory, each job is executed by only one processor at a time. In the last years however, due to the rapid development of parallel computer systems, new theoretical approaches have emerged to model scheduling on parallel architectures (for an overview about scheduling of multiprocessor jobs see [2,5,13]).

We study variants of the non-preemptive parallel job scheduling problem. An instance of this problem is given by a list $L := \{J_1, \ldots, J_n\}$ of jobs and for each job J_j an execution time p_j and the number of required machines q_j is given. A schedule $S = ((s_1, r_1), \ldots, (s_n, r_n))$ is a sequence of starting times $s_j \geq 0$ together with the set of assigned machines $r_j \subseteq \{1, \ldots, m\}$ $(|r_j| = q_j)$ for $j \in \{1, \ldots, n\}$. A schedule is feasible if each machine executes at most one job at the same time. The length of a schedule is defined as its latest job completion time $C_{\max} = \max\{s_j + p_j | j \in \{1, \ldots, n\}\}$. The objective is to find a feasible schedule of minimal length. This problem is denoted $P|\mathrm{size}_j|C_{\max}$ (for more information on this three field notation see [5]).

$P|\mathrm{size}_j|C_{\max}$ is strongly NP-hard, since already the problem $P5|\mathrm{size}_j|C_{\max}$, is strongly NP-hard [6], and, unless $P = NP$, there is no approximation algorithm

* Research supported by a PPP funding "Approximation algorithms for d-dimensional packing problems" 315/ab D/05/50457 granted by the DAAD, and by EU research project AEOLUS, Algorithmic Principles for Building Efficient Overlay Computers, EU contract number 015964.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 234–245, 2008.

with a performance ratio better than 1.5 for $P|\text{size}_j|C_{\max}$ [11]. Furthermore, Johannes [11] has shown that no list-scheduling algorithm can perform better than 2.

The best known algorithm with polynomial running time for this problem was implicitly given by Garey & Graham [7]. They proposed a list-based algorithm with approximation ratio 2 for a resource-constrained scheduling problem. In this scheduling problem one or more resources are given and for the duration of its execution time each job requires a certain amount of each resource. As pointed out by Ludwig & Tiwari [14] this resource-constrained scheduling problem can be used to model $P|\text{size}_j|C_{\max}$ by using the available processors as the single resource. The existence of a polynomial time approximation scheme (PTAS) for the case that the number of available processors is a constant, $Pm|\text{size}_j|C_{\max}$, was presented in [1,9].

A problem closely related to $P|\text{size}_j|C_{\max}$ is the *StripPacking*-problem (ie. packing of rectangles in a strip of width 1 while minimizing the packing heigth). The main difference is that machines assigned to a job need to be contiguous in a solution to the *StripPacking*-problem. We denote the corresponding *Scheduling*-problem by $P|line_j|C_{\max}$. Turek et al. [18] pointed out, that using contiguous machine assignments is desirable in some settings (for example to maintain a physical proximity of processors allotted to a job). One of the first results for the *StripPacking*-problem was given by Coffman et al. [3]. They proved that the level-based algorithm NFDH (Next Fit Decreasing Height) has approximation ratio 3. The currently best known algorithms with absolute approximation ratio were given independently by Schiermeyer [16] and Steinberg [17], who presented algorithms with ratio 2. Coffman et al. [3] also analyzed the asymptotic performance of NFDH, which is $2\,\text{OPT} + h_{\max}$, where $\text{OPT}, h_{\max}$ denote the height of an optimal solution and the height of the tallest rectangle, respectively. An AFPTAS for this problem was presented by Kenyon & Rémila [12]. Recently, Jansen & Solis-Oba [10] presented an asymptotic polynomial time approximation scheme (APTAS) with additive term 1 at the cost of a higher running time.

New results. In this paper we focus on the natural case when the number of machines is polynomial in the number of jobs (in most scenarios the number of machines will be even lower than the number of jobs). We will denote this problem by $P\text{poly}|\text{size}_j|C_{\max}$ or by $P\text{poly}|\text{line}_j|C_{\max}$ in the contiguous case. Using a reduction from 3-Partition [8], it is easy to see that these problems are strongly NP-hard.

For the case that all machines assigned to a job have contiguous addresses we show the existence of an algorithm with approximation ratio 1.5.

Theorem 1. *For every $\varepsilon > 0$ and every instance I of $P\text{poly}|line_j|C_{\max}$ there exists an algorithm A with $A(I) \leq (1.5 + \varepsilon)\,\text{OPT}(I)$ and running time polynomial in n, where $A(I)$ is the length of the schedule for instance I generated by algorithm A and $\text{OPT}(I)$ is the length of an optimal schedule for instance I.*

The previous best known result for this problem is the above mentioned 2-approximation algorithm for the resource-constrained problem by Ludwig & Tiwari [14].

In the general case (non-contiguous addresses) we show the existence of a polynomial time approximation scheme (PTAS).

Theorem 2. *For every $\varepsilon > 0$ and every instance I of $Ppoly|size_j|C_{\max}$ there exists an algorithm A with $A(I) \leq (1+\varepsilon)\,\mathrm{OPT}(I)$ and running time polynomial in n, where $A(I)$ is the length of the schedule for instance I generated by algorithm A and $\mathrm{OPT}(I)$ is the length of an optimal schedule for instance I.*

The previous best known result for this problem is a 2-approximation algorithm based on list scheduling by Garey & Graham [7]. This is the best possible result (in the sense of approximation ratio), since the problem is strongly NP-hard.

A similar problem is the scheduling of so called *malleable* jobs, where the number of required machines for each job is not known a priori; the execution time of each job depends on the number of allotted machines. That is, instead of p_j, q_j each job J_j has an associated function $p_j : \{1, \ldots, m\} \to \mathbb{Q}^+$ that gives the execution time $p_j(\ell)$ of J_j in terms of the number ℓ of processors that are assigned to J_j. For this *Scheduling*-problem Turek et al. [18] and Ludwig & Tiwari [14] presented algorithms with approximation ratio 2 for both cases (contiguous and non-contiguous) without additional properties. Mounié et al. [15] gave an algorithm for scheduling monotonic malleable tasks with approximation ratio $(1.5 + \varepsilon)$. For the special case of identical malleable tasks Decker et al. [4] presented an approximation algorithm with ratio 1.25. In the full version of this paper we will show how the algorithms for both non-malleable cases can be extended to solve the corresponding malleable versions with the same approximation ratios (without monotonic properties of the processing times).

Structure. In Sect. 2 we present the algorithm for scheduling jobs on machines with contiguous addresses. In Sect. 2.1 we show how an optimal solution can be transformed into a nearly optimal solution with simpler structure. After these preliminary steps we explain in Sect. 2.2 the pre-positioning of a subset of the rectangles and introduce a linear program (LP) formulation for the assignment of the remaining rectangles. In Sect. 2.3 we outline how to pack the rectangles according to the fractional solution of the LP. In Sect. 3 we present the scheduling algorithm for the case that the machines allotted to each job are not required to have contiguous addresses.

2 Contiguous Parallel Job Scheduling

Since each job is required to be executed on contiguous machines, an instance of the *Scheduling*-problem can be translated directly into a *StripPacking*-instance; create for each job J_i a rectangle $R_i = (w_i, h_i)$ with width $w_i = q_i/m$ and height $h_i = p_i$ (where q_i is the number of required machines and p_i is the execution time of J_i). The objective is then to find an orthogonal, axis parallel arrangement of all rectangles into a strip of width 1 and minimal height (without rotations). Thus, the StripPacking- and the Scheduling-notation can be used synonymously in the contiguous case. In this paper we use the StripPacking-notation since this notation is more descriptive.

2.1 Near-Optimal Packing with Simple Structure

The following construction of a nearly optimal solution with simple structure based on a given optimal solution is similar to the solution constructed in [10].

Let $0 < \varepsilon' \leq 1$ and let $L = \{R_1, \ldots, R_n\}$ be a *Scheduling*-instance and for each rectangle (job) R_i let w_i be its width and h_i be its height. A packing (schedule) P for instance L is given as a set of pairs $P = \{(x_1, y_1), \ldots, (x_n, y_m)\}$, where each pair $(x_i, y_i) \in \mathbb{R}^2_{\geq 0}$ denotes the position of the lower left corner of rectangle R_i in the strip. Note that in this case the representation of the schedule by a packing is sufficient since the subset of assigned processors is well-defined by the first assigned processor. We assume that the lower left corner of the strip coincides with the origin of a Cartesian system of coordinates. A packing P is valid if the rectangles do not overlap and $x_i + w_i \leq 1$ for all $i \in \{1, \ldots, n\}$. The height of packing P is given by $\mathrm{h}(P) := \max_{i \in \{1,\ldots,n\}}(y_i + h_i)$.

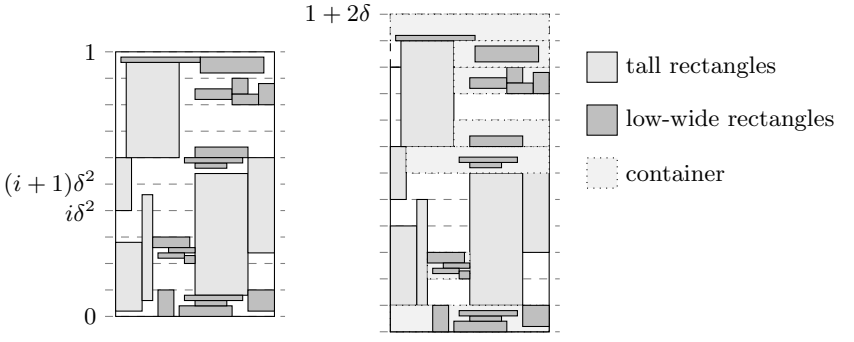
Bounded height. Since we want to divide the solution into a constant number of *slots*, we need to know the height of an optimal solution, at least up to the required accuracy ε'. By using the *StripPacking*-algorithm of Steinberg [17], we can find a solution for the *StripPacking*-instance of height $v \leq 2\,\mathrm{OPT}$, where OPT is the height of an optimal solution. Let $C := \{(1 + 0\varepsilon')^v/2, (1 + 1\varepsilon')^v/2, \ldots, (1 + \lceil 1/\varepsilon' \rceil \varepsilon')^v/2\}$, then there exists a value $v^* \in C$ such that $\mathrm{OPT} \leq v^* \leq (1 + \varepsilon')\,\mathrm{OPT}$. Obviously we only have to consider $\lceil 1/\varepsilon' \rceil + 1$ different candidates to find the right one. For simplicity we divide the height of each rectangle by v^*, such that the height of an optimal solution for the scaled instance is bounded by 1. In the following we show the existence of an algorithm that packs all rectangles of a scaled instance into a strip of height at most $(1.5 + \varepsilon')$. This height bound is sufficient to prove Theorem 1, since rescaling yields $v^*\,(1.5 + \varepsilon') \leq (1.5 + 4\varepsilon')\,\mathrm{OPT}$.

Partitioning the set of rectangles. In the following we assume that the instance is already scaled such that an optimal packing P^* has height $\mathrm{h}(P^*)$, where $(1 - \varepsilon') < \mathrm{h}(P^*) \leq 1$. Let ε be the largest value of the form $\varepsilon = 1/(2a)$ for an integer a such that $\varepsilon \leq \varepsilon'/13$. First we create a gap in size between *tall* and *low* rectangles and between *wide* and *narrow* restangles by removing *middle-sized* rectangles with small total area. Let $\sigma_0 := 1, \sigma_1 := \varepsilon$, and $\sigma_k := (\sigma_{k-1})^{8/\sigma_{k-1}^3}$ for all $k \geq 2$. Define $L^{>1/2} := \left\{R_i \in L | h_i > (1+\varepsilon)/2\right\}$, and $L_k := \{R_i \in L \setminus L^{>1/2} | w_i \in (\sigma_k, \sigma_{k-1}]$ or $h_i \in (\sigma_k, \sigma_{k-1}]\}$. Define for each subset $L' \subseteq L$ the total area of L' by $A(L') = \sum_{R_i \in L'}(w_i h_i)$.

Lemma 1. *There exists* $k \in \{2, \ldots, 4/\varepsilon + 1\}$ *such that* $A(L_k) \leq \varepsilon/2 A(L)$.

Choose the smallest value k satisfying the conditions of Lemma 1 and define $\delta := \sigma_{k-1}$ and $s := 8/\delta^3$. For simplicity we define the following sets and call rectangles belonging to each set accordingly: $L_{\mathrm{ta}} := \{R_i \in L | h_i > \delta\}$ *tall* rectangles, $L_{\mathrm{lo}} := \{R_i \in L | h_i \leq \delta^s\}$ *low* rectangles, $L_{\mathrm{wi}} := \{R_i \in L | w_i > \delta\}$ *wide* rectangles, and $L_{\mathrm{na}} := \{R_i \in L | w_i \leq \delta^s\}$ *narrow* rectangles. We will denote the subset of low-wide rectangles in the following with $L_{\text{lo-wi}} := L_{\mathrm{lo}} \cap L_{\mathrm{wi}}$.

Rounding and shifting tall rectangles. A crucial part of the simple structure are the positions and heights of the tall rectangles. Let P be an optimal



(a) Original packing (b) Packing with simpler structure

Fig. 1. Rounding and shifting rectangles

packing for all rectangles. First we increase the height of each tall rectangle $R_i \in L_{\text{ta}}$ to the nearest multiple of δ^2. Then, we shift the rectangles up until all rectangles $R_i \in L_{\text{ta}}$ have their corners placed at points (x'_i, y'_i), such that there exists some integer k_i with $x'_i = x_i$ and $y'_i = k_i\delta^2$. Since tall rectangles have height at least δ, these transformations increase the total height of packing P by at most $1/\delta(2\delta^2) = 2\delta$ (see Fig. 1).

Containers for low rectangles. Since we want to increase only the height but not the width of the packing, we cannot round up the width of the wide rectangles. Instead we introduce *containers*, into which all low-wide ($L_{\text{lo-wi}}$) and a subset of the low-narrow rectangles will be packed. Consider a scaled and shifted packing P. Draw horizontal lines spaced by a distance δ^2 across the strip (due to the rounding and shifting, the lower and upper sides of the tall rectangles lie along two of these lines). These lines split the strip into at most $(1+2\delta)/\delta^2$ horizontal rectangular regions that we call *slots* (see Fig. 1). A container is a rectangular region inside a slot whose left boundary is either the right side of a tall rectangle or the left side of the strip, and whose right boundary is either the left side of a tall rectangle or the right side of the strip (see Fig. 1). In the following we consider only containers that contain at least one low-wide rectangle. As in [10], we can show, that a packing can be transformed such that all non-empty containers have width $w_{\max}(C) + i\delta^s$ for $i \in \mathbb{N}$, where $w_{\max}(C)$ is the sum of the width of at most $1/\delta$ low-wide rectangles. This limits the number of possible widths for the containers; however, some of the rectangles previously packed in C might not fit anymore, but the total area of all non-fitting rectangles is bounded by δ.

2.2 Pre-positioning

The next step is to determine the positions of the containers and of a subset of the tall rectangles (*critical* rectangles). On the one hand we have to make sure that all discarded rectangles have height bounded by $1/2$. On the other hand we have to

make sure that the pre-positioning has a polynomial running time. In particular, we can only enumerate the positions of a constant number of tall rectangles and containers. From here on let $\mathcal{C}$ be the set of containers and let $L'_{\text{ta}} \subseteq L_{\text{ta}} \setminus L^{>1/2}$ be the subset of K tall rectangles with largest area for some constant K (see full version), and let $L' = \mathcal{C} \cup L'_{\text{ta}}$. Note that $|\mathcal{C}| \leq {}^{(1+2\delta)}/_{\delta^3} \leq 2\delta^{-3}$ and thus $|L'| \leq K + 2\delta^{-3}$. In order to determine the positions of the *critical* rectangles, first we enumerate assignments of the K tall rectangles L'_{ta} and of the containers $\mathcal{C}$ to *slots* and *snapshots*. Then we describe a dynamic program that assigns the tall rectangles from $L^{>1/2}$ to snapshots without enumerating all possibilities. Using these assignments we set up a linear program (LP). If this LP has a solution, we have found a fractional solution for the packing problem. Furthermore, if almost all low-wide rectangles fit into the containers, we show that the fractional solution can be transformed into a feasible integral solution by discarding some rectangles with small total area.

Slot assignment. We split again the strip into horizontal slots of height δ^2. A slot assignment for L' is a mapping $f : L' \to M$ where $M = \{1, \ldots, {}^{(1+2\delta)}/_{\delta^2}\}$ corresponds to the set of slots. For a given slot assignment f the set of slots that will be used for packing a rectangle $R_j \in L'$ is given by $\{f(R_j), \ldots, f(R_j) + \gamma_j - 1\}$, where $\gamma_j \delta^2 = h_j$ is the height of rectangle R_j (in particular $\gamma_j = 1$ if R_j is a container). Since the number of different mappings f is bounded by $O(n^{(1+2\delta)/\delta^2})$, we can consider all mappings f in polynomial time and for each mapping try to find a packing for L that is consistent with f.

Snapshots. In order to handle the x position of the rectangles we introduce snapshots. We use the snapshots to model the relative horizontal positions of all rectangles in L'. Consider a packing for L'. Trace vertical lines extending the sides of the rectangles in L' (see Fig. 2). The region between two of these adjacent lines is called a snapshot. If we index all snapshots from left to right, every rectangle $R_j \in L'$ appears in a sequence of consecutive snapshots $\mathrm{S}_{\alpha_j}, \ldots, \mathrm{S}_{\beta_j}$, where α_j denotes the index of the first snapshot in which rectangles R_j occurs and β_j denotes the index of the last snapshot. In Fig. 2 rectangle R_1 is contained in snapshot S_1, while R_2 is contained in snapshots $\mathrm{S}_2, \mathrm{S}_3, \mathrm{S}_4$, thus $\alpha_1 = 1, \beta_1 = 1, \alpha_2 = 2$ and $\beta_2 = 4$. More formally, an assignment of rectangles in L' to snapshots is given by two functions $\alpha, \beta : L' \to \{1, \ldots, g\}$, where g denotes the number of snapshots. Since $|L'| \leq K + 2\delta^{-3}$ the maximum number of snapshots g in any packing for L' is at most $g \leq 2(K + 2\delta^{-3})$, and thus the number of different assignments of L' to snapshots is polynomial, $O(g^{2|L'|})$.

Dynamic program for $L^{>1/2}$-rectangles. In general we cannot consider all assignments of rectangles in $L^{>1/2}$ to snapshots, because there might be up to n rectangles in $L^{>1/2}$. In the following we introduce an algorithm that allows us to enumerate a subset of all snapshot assignments for $L^{>1/2}$ such that the size of the subset is polynomially bounded in n and there exists one snapshot assignment in this subset that is *equivalent* (up to renaming) to a snapshot assignment induced by an optimal packing. Rectangles in $L^{>1/2}$ that intersect more than one snapshot are handled separately, since our packing algorithm can only be used for tall rectangles that do not intersect more than one snapshot.

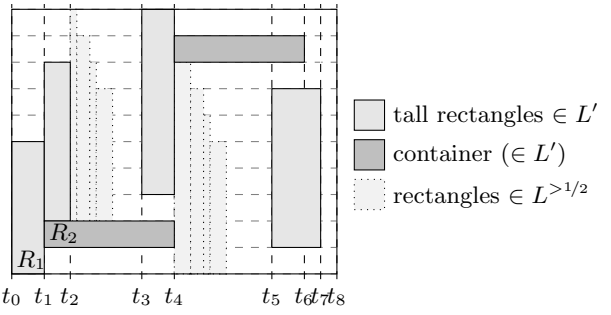


Fig. 2. Packing of rectangles and containers and induced snapshots

We guess the set of tall rectangles intersecting snapshot boundaries, which can be done in polynomial time since there are at most g of these rectangles. In order to pack these rectangles we simply add them to L' (the set of K tall rectangles and containers). This modification increases the size of L' such that $|L'| \leq 3(K + 2\delta^{-3})$, but this does not increase the vector sizes, that we define in the following, since the snapshots introduced by these added rectangles obviously do not allow further $L^{>1/2}$ rectangles to be packed in them (height $> 1/2$). Note that, due to the scaling of all tall rectangles and due to the height bound of 1 for all rectangles, $L^{>1/2}$ can be partitioned into $(1-\varepsilon)/(2\delta^2) < 1/(2\delta^2)$ sets $L_i^{>1/2}$ ($i \in I := \{(1+\varepsilon)/(2\delta^2) + 1, \ldots, 1/\delta^2\}$), where each set $L_i^{>1/2}$ contains all rectangles of height $s_i := i\delta^2$, that is $L_i^{>1/2} := \{R_j \in L^{>1/2} | h_j = s_i\}$. Consider a packing of all rectangles and snapshots as defined above. Then we can define a vector $v^i = (v_1^i, \ldots, v_g^i)$ for each height $s_i, i \in I$, where $v_j^i \in \{0, \ldots, m\}$ is chosen such that $v_j^i \cdot 1/m$ is the sum of widths of all rectangles of height s_i contained in snapshot S_j and height s_i. We assume that a rectangle is contained in a snapshot, if its left side is contained in it. Since the width of the strip is bounded by 1, the value of every component is bounded by m. Thus, a rough upper bound for the number of feasible vectors for each height s_i is given by $(m+1)^g$. The algorithm to calculate all feasible vectors for a given height s_i works as follows.

Assume that $L_i^{>1/2} = \{R_1, \ldots, R_{k_i}\}$. Starting with the set $V := \{(0, \ldots, 0)\}$ we replace in step $l \in \{1, \ldots, k_i\}$ each vector $v \in V$ with all vectors that can be generated by adding $\gamma_l := w_l \cdot m$ to one of its components. After each step remove all duplicate vectors. Removing all duplicates ensures that after each step the number of vectors is bounded by $(m+1)^g$. Using this bound it is easy to see that the overall running time for height s_i is bounded by $O(g^2(m+1)^{g+2})$, since $k_i = |L_i^{>1/2}| \leq |L^{>1/2}| \leq m$, and for each step l at most $g \cdot (m+1)^g$ vectors are generated; furthermore, at most $g^2(m+1)^g \log(m+1)$ comparisons are needed for the removal of duplicate vectors. Obviously the vector v^i induced by the given packing can be found among the generated vectors. Let V^i denote the set of vectors generated for this height class $L_i^{>1/2}$.

Repeating this computation for every height class $L_i^{>1/2}$ $(i \in I)$ leads to $|I| \leq 1/(2\delta^2)$ sets of at most $(m+1)^g$ vectors. Since each set contains at least the null vector and thus each set is not empty, we can build the direct product $V := \prod_{i \in I} V^i$ of these sets. V contains at most $((m+1)^g)^{1/(2\delta^2)}$ elements and each of these elements consists of one vector for each height class. One element $v = (v^1, \ldots, v^{|I|}) \in V$ corresponds to the vectors induced by the given packing.

In our packing algorithm we try each vector $v \in V$ consisting of components $v^i, i \in I$ and use these vectors v^i to pack the tall rectangles into the snapshots. Using the described version of the dynamic program results only in (many) vectors, but for our packing algorithm we also need to know what combination of rectangles leads to the given width per snapshot. This can be achieved by extending the dynamic program such that for each vector a component consists not only of the current width, but also of a set of rectangles. During the vector generation step, a rectangle is added to this set, if its width is added to the corresponding width component. Due to this modification the space needed to store the vectors increases but is still polynomial in n; the runtime of the dynamic program is not affected significantly.

Linear Program. In this subsection we present a linear program (LP), which allows us to calculate the width of all snapshots, and thus determine the positions of all rectangles in L'. We now assume that we have chosen a slot assignment f, functions α, β, and $v \in V$ consisting of vectors v^i of widths for each height class. Since all low-wide rectangles and a subset of the low-narrow rectangles get packed into the containers, we do not need to consider them in the LP. For convenience we call the subset of the low-narrow rectangles that are packed into the containers $L^C_{\text{lo-na}}$, and the remaining low-narrow rectangles $L_{\text{lo-na}}$. We construct $L^C_{\text{lo-na}}$ by greedily taking low-narrow rectangles as long as their total area plus the total area of all low-wide rectangles does not exceed the total area of the containers. We discard the first low-narrow rectangle that exceeds the total area in order to assure that enough space can be reserved for the remaining rectangles in the following LP. This discarded rectangle has area at most δ^{2s}, in fact we will show, that this discarded rectangle can be packed along with the rectangles from $L_{\text{lo-na}}$ (see Sect. 2.3). We will denote the total area of the remaining rectangles from $L_{\text{lo-na}}$ as $\mathrm{A}(L_{\text{lo-na}})$. Since f, α, β are fixed, we can calculate the set of *free* slots (i.e. the slots not occupied by L' rectangles) for each snapshot. These free slots will be used for packing the remaining tall rectangles and for the small rectangles from $L_{\text{lo-na}}$. In order to formulate constraints to ensure that enough space is reserved for these rectangles we introduce configurations. We define a configuration as a pair (SN, Π), where SN is a subset of the free slots reserved for rectangles from $L_{\text{lo-na}}$ and Π is a partition of the remaining free slots into sets of consecutive slots reserved for rectangles from $L_{\text{ta}} \setminus L'$. Every subset $F \in \Pi$ of cardinality $l = |F|$ is reserved to pack rectangles from L_{ta} of height $l\delta^2$. Let n_j denote the number of different configurations for each snapshot S_j and let $c_i^j := (\mathrm{SN}_i^j, \Pi_i^j)$ denote the different configurations for snapshot S_j, $i \in \{1, \ldots, n_j\}$. Let $n_i^j(l) := |\{F \in \Pi_i^j : |F| = l\}|$ denote the number of sets of cardinality l in Π_i^j. The total width of all rectangles in $L_{\text{ta}} \setminus L'$ of height l is denoted as

W_l. The variables $x_i^j, j \in \{1,\dots,g\}, i \in \{1,\dots,n_j\}$ are used to determine the width of each configuration c_i^j. Additional variables $t_j, j \in \{1,\dots,g\}$, are used to determine the width of each snapshot S_j.

$$
\begin{aligned}
\mathrm{LP}(f,\alpha,\beta,v): \quad & t_0 = 0, t_g \le 1 && \\
& t_j \ge t_{j-1} && \forall j \in \{1,\dots,g\} \\
& t_{\beta_j} - t_{\alpha_j} = w_j && \forall R_j \in L' \\
& \sum_{i=1}^{n_j} n_i^j(l) x_i^j \ge \frac{1}{m} v_j^l && \forall j \in \{1,\dots,g\}, l \in I \\
& \sum_{j=1}^{g} \sum_{i=1}^{n_j} n_i^j(l) x_i^j \ge W_l && \forall l \in M \\
& \sum_{j=1}^{g} \sum_{i=1}^{n_j} x_i^j |\mathrm{SN}_i^j| \delta^2 \ge \mathrm{A}(L_{\text{lo-na}}) && \\
& \sum_{i=1}^{n_j} x_i^j \le t_j - t_{j-1} && \forall j \in \{1,\dots,g\} \\
& x_1^j, \dots, x_{n_j}^j \ge 0 && \forall j \in \{1,\dots,g\}
\end{aligned}
$$

Since $g, n_j, |L'|, |M|, |I|$ are independent of n, this linear program can be solved in polynomial time. If $\mathrm{LP}(f,\alpha,\beta,v)$ has no feasible solution we construct a new LP with a new combination of f,α,β,v.

2.3 Packing the Rectangles

Due to the page limit we present here only a brief outline of the actual packing of the rectangles. A complete description is given in the full version of this paper. A crucial step is to sort and adopt the configurations. On the one hand, this allows us to pack the tall rectangles $L^{>1/2}$ and on the other hand, this assures that the fragmentation of the space reserved for low-narrow rectangles is *limited*. The pre-positioned rectangles are packed according to the LP-solution. The remaining tall rectangles get fractionally packed in a greedy manner. We pack the containers using a modified version of the algorithm by Kenyon & Rémila [12]. The remaining low-narrow rectangles are packed using a NFDH approach. Overall we can show that almost all rectangles can be packed. In particular, the remaining rectangles have a small total area and maximum height bounded by $(1+\varepsilon)/2$ and thus can be packed in an additional strip with height bounded by $\varepsilon + 9\delta + (1+\varepsilon)/2$. Stacking these strips on top of each other leads to a strip with total height bounded by $1.5 + \varepsilon'$.

3 Non-contiguous Parallel Job Scheduling

In the following we construct a polynomial time approximation scheme (PTAS) for the non-contiguous machine indices case. The first steps for this algorithm are basically the same as before. We transform the problem into a packing problem, guess the height of an optimal schedule in order to scale the instance, such that the height of an optimal solution for the scaled instance is bounded by 1.

Instead of the 2-approximation algorithm for the *StripPacking*-problem, we use a 2-approximation algorithm for this *Scheduling*-problem by Garey & Graham [7]. We again partition the rectangles, but for this setting it is sufficient to partition the set into tall, low-narrow, and low-wide rectangles, and we reduce the search space by scaling and shifting the tall rectangles in the same manner as before (see Sect. 2.1). After this step the algorithms differ significantly. We use a dynamic program to find a distribution of the tall rectangles among the slots. Then we pack the tall rectangles according to the distribution in a canonical way. The remaining space is merged into one bin per slot. We pack the low rectangles into these bins using again the modified Kenyon & Rémila algorithm. Then, creating a feasible packing can be done by a simple greedy algorithm.

Partitioning the set of rectangles. Let ε' denote the requested accuracy and let $\varepsilon \leq \varepsilon'/9$ be the largest value of the form $\varepsilon = 1/a$ for some integer value a. Let $\sigma_0 := 1, \sigma_1 := \varepsilon$, and $\sigma_k := \sigma_{k-1}^3/2 \cdot 7^2$ for all $k \geq 2$. Define $L_k := \{R_i \in L | h_i \in (\sigma_k, \sigma_{k-1}]$ or $w_i \in (\sigma_k, \sigma_{k-1}]\}$. It is easy to see, that there exists $k \in \{2, \ldots, 2/\varepsilon + 1\}$ such that $A(L_k) \leq \varepsilon A(L)$ (see Lemma 1). Choose the smallest value $k \in \{2, \ldots, 2/\varepsilon + 1\}$ such that $A(L_k) \leq \varepsilon A(L)$, and let $\delta := \sigma_{k-1}$, and $\gamma := \sigma_k$. Define $L_{\text{ta}} := \{R_i \in L | h_i > \delta\}$ *tall* rectangles, $L_{\text{lo-wi}} := \{R_i \in L | h_i \leq \gamma, w_i > \delta\}$ *low-wide* rectangles, and $L_{\text{lo-na}} := \{R_i \in L | h_i \leq \gamma, w_i \leq \gamma\}$ *low-narrow* rectangles.

Shifting and Rounding. Shifting and rounding is done the same way as before. Again this leads to the existence of a packing with height bounded by $1 + 2\delta$ (see Sect. 2.1).

Dynamic program for tall rectangles. Draw horizontal lines spaced by a distance δ^2 across the strip starting with the x-axis as first such line. Note that, due to the rounding and shifting the lower side of each tall rectangle corresponds to one of these horizontal lines. We say that rectangle R_i is in slot j if its lower bound corresponds to the jth horizontal line. Since we cannot enumerate all possible assignments of tall rectangles to slots in polynomial time, we use a dynamic programming approach. Due to the height bound of 1 and the rounding, we can partition the set of tall rectangles L_{ta} into height classes $L_{\text{ta}}^i := \{R_j \in L_{\text{ta}} | h_j = i\delta^2\}, i \in I := \{1, \ldots, 1/\delta^2\}$. Note that we partition all tall rectangles in this case, not only $L^{>1/2}$. For each height class $i \in I$ we define a vector $v^i = (v_1^i, \ldots, v_{(1+2\delta)/\delta^2}^i)$, such that each entry v_j^i denotes the total width of all tall rectangles with height $i\delta^2$ in slot $j \in J := \{1, \ldots, (1+2\delta)/\delta^2\}$. The algorithm to calculate all feasible vectors for a given height class i works as follows. Assume that $L_{\text{ta}}^i = \{R_1, \ldots, R_{k_i}\}$. Starting with a set $V^i := \{(0, \ldots, 0)\}$ we replace in step $l \in \{1, \ldots, k_i\}$ each vector $v \in V^i$ with all vectors that can be generated by adding w_l to one of its components. After each step remove all duplicate vectors. Similar arguments as in Sect. 2.2 show that the number of vectors generated and the running time of the algorithm is polynomially bounded in m. After repeating this computation for each height class L_{ta}^i, we can again build the direct product of all sets of vectors $V := \prod_{i \in I} V^i$ and again there is an element $v \in V$ that corresponds to the vectors of widths induced by an optimal solution. Analogous to the first case, we extend our dynamic program such that for each component

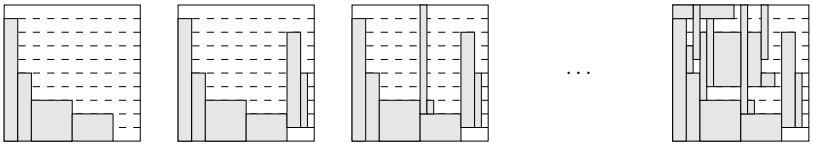


Fig. 3. Canonical packing for tall rectangles

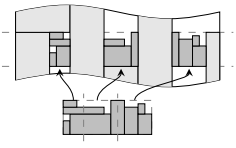


Fig. 4. Split bins to pack slots

of each vector a set of associated rectangles is stored. Let $L(v_j^i)$ be this set of rectangles associated with v_j^i. We call an element $v \in V$ feasible, if for each slot the total width of all rectangles intersecting this slot and of all rectangles in this slot is not greater than m. In contrast to the previous case not all elements $v \in V$ are feasible.

Canonical packing for the tall rectangles. Given a feasible vector $v \in V$, we can pack all tall rectangles using the following simple algorithm (see Fig. 3). The algorithm starts with the first slot and packs left aligned all tall rectangles $L(v_1^i), i \in I$ into it. This is obviously possible, since the vector is feasible. Now assume that we have packed all slots prior to slot j. The free space in slot j is sufficient to pack all rectangles assigned by $v_j^i, i \in I$, since v is feasible, and furthermore, the free space in this slot is also free in all following slots. This allows us to pack all rectangles given by v_j^i left aligned into slot j.

Bins for the low rectangles. Given a feasible combination of vectors $v \in V$, the total width for each slot that is not occupied by tall rectangles is fixed. Let w_j^f denote the total width of the free space for slot j and define for each slot j a bin C_j of width w_j^f and height δ^2. If we find a packing of all low rectangles into these bins, and know the positions of the tall rectangles, we can simply split the bins into slices of width $1/m$ and add them successively left aligned to the free space of each corresponding slot (see Fig. 4). Since there exists a packing for all rectangles into the strip, the total area of the bins is not smaller than the total area of the remaining low-wide rectangles together with the low-narrow rectangles. We can show, that almost all rectangles are packable using a modified version of the algorithm by Kenyon & Rémila [12].

Analysis. Overall we discarded rectangles with total area bounded by $\varepsilon + 2\delta$. Packing these rectangles using NFDH and due to the fact the all these rectangles have height bounded by δ yields an additional height of $2(\varepsilon + 2\delta) + \delta$. Thus,

the height of the resulting strip is bounded by $(1+2\delta)+(2\varepsilon+5\delta) \leq 1+9\varepsilon \leq 1+\varepsilon'$, if we stack these to strips on top of each other. Thus, we can prove Theorem 2, since rescaling yields $v^*(1+\varepsilon') \leq (1+3\varepsilon')\,\mathrm{OPT}$.

References

1. Amoura, A.K., Bampis, E., Kenyon, C., Manoussakis, Y.: Scheduling independent multiprocessor tasks. Algorithmica 32(2), 247–261 (2007)
2. Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Weglarz, J.: Handbook on Scheduling: From Theory to Applications. Springer, Heidelberg (2007)
3. Coffman Jr., E.G., Garey, M.R., Johnson, D.S., Tarjan, R.E.: Performance bounds for level-oriented two-dimensional packing algorithms. SIAM Journal on Computing 9(4), 808–826 (1980)
4. Decker, T., Lücking, T., Monien, B.: A 5/4-approximation algorithm for scheduling identical malleable tasks. Theor. Comput. Sci. 361(2), 226–240 (2006)
5. Drozdowski, M.: Scheduling multiprocessor tasks – an overview. European Journal of Operational Research 94(2), 215–230 (1996)
6. Du, J., Leung, J.Y.-T.: Complexity of scheduling parallel task systems. SIAM J. Disc. Math. 2(4), 473–487 (1989)
7. Garey, M.R., Graham, R.L.: Complexity results for multiprocessor scheduling under resource constraints. SIAM Journal on Computing 4(4), 397–411 (1975)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
9. Jansen, K., Porkolab, L.: Linear-time approximation schemes for scheduling malleable parallel tasks. Algorithmica 32(3), 507–520 (2002)
10. Jansen, K., Solis-Oba, R.: New Approximability Results for 2-Dimensional Packing Problems. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 103–114. Springer, Heidelberg (2007)
11. Johannes, B.: Scheduling parallel jobs to minimize the makespan. Journal of Scheduling 9(5), 433–452 (2006)
12. Kenyon, C., Rémila, E.: A near optimal solution to a two-dimensional cutting stock problem. Mathematics of Operations Research 25, 645–656 (2000)
13. Leung, J.Y.-T. (ed.): Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC (2004)
14. Ludwig, W., Tiwari, P.: Scheduling malleable and nonmalleable parallel tasks. In: Proc. 5th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 167–176 (1994)
15. Mounie, G., Rapine, C., Trystram, D.: A $\frac{3}{2}$-approximation algorithm for scheduling independent monotonic malleable tasks. SIAM Journal on Computing 37(2), 401–412 (2007)
16. Schiermeyer, I.: Reverse-fit: A 2-optimal algorithm for packing rectangles. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 290–299. Springer, Heidelberg (1994)
17. Steinberg, A.: A strip-packing algorithm with absolute performance bound 2. SIAM Journal of Computing 26(2), 401–409 (1997)
18. Turek, J., Wolf, J.L., Yu, P.S.: Approximate algorithms for scheduling parallelizable tasks. In: Proc. 4th ACM Symp. on Parallel Alg. and Architectures (SPAA), pp. 323–332 (1992)

A PTAS for Static Priority Real-Time Scheduling with Resource Augmentation

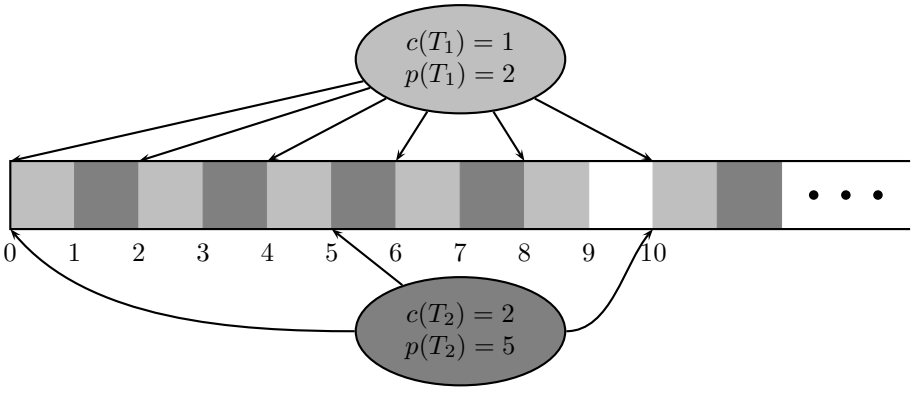
Friedrich Eisenbrand* and Thomas Rothvoß

Institute of Mathematics
École Polytechnique Féderale de Lausanne, 1015 Lausanne, Switzerland
{friedrich.eisenbrand,thomas.rothvoss}@epfl.ch

Abstract. We present a polynomial time approximation scheme for the *real-time scheduling problem with fixed priorities* when resource augmentation is allowed. For a fixed $\varepsilon > 0$, our algorithm computes an assignment using at most $(1+\varepsilon)\cdot OPT+1$ processors in polynomial time, which is feasible if the processors have speed $1+\varepsilon$. We also show that, unless $P = NP$, there does not exist an asymptotic FPTAS for this problem.

1 Introduction

In this paper, we are concerned with a scheduling problem described by Liu and Layland [11], which has received considerable attention in the real-time and embedded-systems community. Here one is given a set of *tasks* $\mathcal{T} = \{T_1, \ldots, T_n\}$, where each task T is characterized by two positive values, its *period* $p(T)$ and its *running time* $c(T)$. The task T releases a *job* requiring running time $c(T)$ at each integer multiple of its period.



If several tasks $\mathcal{T}' \subseteq \mathcal{T}$ are assigned to one processor, then this assignment is *feasible* if each job, being released by some task T at time $i \cdot p(T)$ is finished at time $(i+1)\cdot p(T)$, whereby jobs stemming from tasks with smaller period preempt

* Supported by Deutsche Forschungsgemeinschaft (DFG) within Priority Programme 1307 "Algorithm Engineering".

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 246–257, 2008.

those stemming from tasks with larger period. Ties are broken in an arbitrary but fixed way. In this case, we also speak about an assignment in which each *task is feasible* itself. Liu and Layland [11] have shown that this *rate-monotonic scheduling* is optimal, meaning if there is a feasible priority assignment, then the one in which the priority of a task T equals $1/p(T)$ is also feasible.

The picture above shows a feasible set $\mathcal{T}' = \{T_1, T_2\}$ of tasks. The arrows indicate the points in time, where the two tasks T_1 and T_2 release jobs. At time 0, the first job of T_1 as well as the first job of T_2 are released. Since the period of T_1 is smaller than the period of T_2, the first job of T_1 is executed, until it is finished at time 1. Now the first job of T_2 is executed, but interrupted by the second job of T_1 at time 2. The execution of the first job of T_2 is resumed at time 3 and finished at time 4. Notice that the processor is idle for one time unit at time 9 and that the schedule repeats at the least common multiple of the periods which is 10. All jobs finish in time. The set $\mathcal{T}'$ is feasible.

The *static-priority real-time scheduling problem* is now to determine a partitioning of a task-set $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_k$, such that each $\mathcal{T}_i$ is a feasible set of tasks for one processor and the number k of processors is minimized. In the real-time literature, this problem is also known as the static-priority real-time scheduling problem *with implicit deadlines*, since the deadlines are implicitly given by the periods of the tasks.

Related Work. If the periods $p(T)$ of all tasks in $\mathcal{T}$ are one, then the scheduling problem is simply the well known *bin packing problem*. This is because a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ would be feasible on one processor if and only if the sum of their running times is bounded by one. Recall that for bin packing an asymptotic PTAS [4] and even an asymptotic FPTAS exists [8].

The *utilization* of $\mathcal{T}'$ is defined as $\text{util}(\mathcal{T}') = \sum_{T \in \mathcal{T}'} c(T)/p(T)$. If $\mathcal{T}'$ is feasible, then the utilization $\text{util}(\mathcal{T}')$ is at most 1. However, $\mathcal{T}'$ can be infeasible, even if $\text{util}(\mathcal{T}') < 1$. Consider, for example, again the task system $\mathcal{T}'$ depicted on the cover page. If we increase the running time of T_1 by any $\varepsilon > 0$, then the set $\mathcal{T}'$ is no longer feasible and its utilization is $\text{util}(\mathcal{T}') = (9 + 5 \cdot \varepsilon)/10$. Liu and Layland [11] have shown that $\mathcal{T}'$ is feasible, if $\text{util}(\mathcal{T}')$ is bounded by $n'(2^{1/n'} - 1)$, where $n' = |\mathcal{T}'|$. This bound tends to $\ln 2$ and the condition is not necessary for feasibility, as the example with all periods equal to one shows. Stronger, but still not necessary conditions for feasibility are given in [10,2,12].

It is a longstanding open problem, whether there exists a polynomial time algorithm which decides whether a set $\mathcal{T}'$ of tasks is *feasible on one processor*. A first result in this direction using resource augmentation was presented by Fisher and Baruah [5]. In their paper, the authors show that one can efficiently decide whether a set of tasks is feasible, or infeasible on a faster processor of speed $1 + \varepsilon$. Our approximation scheme can be understood as an extension of their algorithm, which additionally approximates the task-distribution problem.

The sufficient condition $\text{util}(\mathcal{T}') \leqslant n'(2^{1/n'} - 1)$ allows to use first-fit and next-fit algorithms as in the case of bin packing. The currently best ratio for such strategies is 7/4 due to [10]. We refer to [3] for a survey of approximation algorithms based on first-fit and next-fit and to the article [1] for an overview

on complexity issues of real-time scheduling. The literature on approximation schemes, especially in scheduling, is extensive. We refer to [13] for a recent account.

Results. We show that, for each $\varepsilon > 0$ there exists a polynomial time algorithm which computes a partitioning using at most $(1+\varepsilon) \cdot OPT(\mathcal{T}) + 1$ subsets. Each subset is feasible on a processor of speed $1+\varepsilon$. Here $OPT(\mathcal{T})$ denotes the minimum number of processors to feasibly schedule $\mathcal{T}$. Our result is the first PTAS for the real-time scheduling problem with resource augmentation. Furthermore we show that real-time scheduling is harder to approximate than bin packing. Unless $P = NP$, there does not exist an algorithm which has an additive gap of $O(n^{1-\varepsilon})$ for any fixed $\varepsilon > 0$. This implies that there does not exist an asymptotic FPTAS for real-time scheduling without resource augmentation.

The main insights which lead to our PTAS with resource augmentation are twofold.

i) Apart from the standard *rounding of the instance*, we describe the concept of *local feasibility*. The effect of far-scattered periods prevents the application of bin packing techniques. The concept of local feasibility considers these effects only for those tasks, whose periods are close or *local* to the period of the task in consideration. A local feasible schedule is feasible on a slightly faster machine.
ii) In bin packing, small items are first discarded from the instance and then distributed with first-fit. Since the utilization is not a good lower bound for the real-time scheduling problem, a similar approach does not work. We provide a much different technique to treat small tasks. We re-set periods and group small tasks with the same period into one large task. A probabilistic argument shows that the optimum of the modified instance does not grow to much.

2 Preliminaries and Simplifying Assumptions

In [11] it is shown that a set $\mathcal{T}$ of tasks is feasible on one processor, if the first job of each task $T \in \mathcal{T}$ finishes before its period $p(T)$, or in other words, if the *response time* of each task is smaller than its period.

This response time r of a task T is calculated as follows. A task T' with higher priority interrupts T exactly $\lceil r/p(T') \rceil$ many times. Each time, this task consumes its processing time $c(T')$. Therefore r is the smallest fix-point of the *response function*

$$f_T(r) = c(T) + \sum_{T' \in \mathcal{T} \setminus \{T\} : p(T') \leqslant p(T)} \lceil r/p(T') \rceil \cdot c(T'). \tag{1}$$

The task system is feasible if there exists for each T a number $r_T^* \leqslant p(T)$ with $f_T(r_T^*) \leqslant r_T^*$. Notice that one has for each $a \in \mathbb{N}_{>0}$ $f_T(a \cdot r) \leqslant a \cdot f_T(r)$. This shows that the task system $\mathcal{T}$ is feasible if and only if there exists an r_T^* for each $T \in \mathcal{T}$

with $p(T)/2 \leqslant r_T^* \leqslant p(T)$ and $f_T(r_T^*) \leqslant r_T^*$. The vector $r^* = (r_{T_1}^*, \ldots, r_{T_n}^*)$ is a *certificate of feasibility* of the task-system $\mathcal{T} = \{T_1, \ldots, T_n\}$. Similarly, we say that the task-system is *feasible on a processor of speed* $\beta > 0$ if there exists a vector $r^* = (r_{T_1}^*, \ldots, r_{T_n}^*)$ with $p(T)/2 \leqslant r_T^* \leqslant p(T)$ and $f_T(r_T^*) \leqslant \beta \cdot r_T^*$. The next Lemma will be used several times in the sequel. A proof can be found in the full version of this paper.

Lemma 1. *Let $\mathcal{T}$ be a set of tasks, then the following holds.*

i) If $\text{util}(\mathcal{T}) \leqslant \gamma$ *with* $\gamma > 0$, *then* $\mathcal{T}$ *is feasible on a processor of speed* $(1/\ln(2)) \cdot \gamma$.
ii) $\text{util}(\mathcal{T}) \leqslant OPT(\mathcal{T}) \leqslant (2/\ln(2)) \cdot \text{util}(\mathcal{T}) + 1$.
iii) If $\mathcal{T}$ *is feasible on a processor of speed* β *and a second set* $\mathcal{T}'$ *has utilization at most* ε, *then* $\mathcal{T} \cup \mathcal{T}'$ *is feasible on a processor of speed* $\beta + 2\varepsilon$.

Simplifying Assumptions. The number $1/\varepsilon$ can be assumed to be an integer. Furthermore, we round each period up to the next power of $(1+\varepsilon)$. If a solution of this rounded instance is feasible, then it is also feasible for the original instance on a processor of speed $(1+\varepsilon)$.

Next, choose $k \in \{0, \ldots, (1/\varepsilon)-1\}$ such that the utilization u_k of tasks, having their period in an interval $[(1/\varepsilon)^i, (1/\varepsilon)^{i+1}[$ with $i \equiv k \pmod{1/\varepsilon}$, is minimized. Clearly $u_k \leqslant \varepsilon \cdot \text{util}(\mathcal{T})$. Thus we may remove all tasks, contributing to u_k and schedule them in a first-fit manner on $(2/\ln 2) \cdot OPT + 1$ additional processors, using Lemma 1(ii). This process yields a partitioning of the tasks into *blocks* $\mathcal{B}_1, \ldots, \mathcal{B}_\mu$ with the following properties.

i) If p_i and p_j are periods of tasks in $\mathcal{B}_i$ and $\mathcal{B}_j$ with $i < j$, then $(1/\varepsilon) \cdot p_i \leqslant p_j$.
ii) The number of different periods of tasks in one block $\mathcal{B}_i$ is bounded by $((1/\varepsilon) - 1) \cdot \log_{1+\varepsilon}(1/\varepsilon) \leqslant 1/\varepsilon^3$ which is a constant.

3 Real-Time Scheduling Is Harder Than Bin Packing

Due to its relation to bin packing, a natural question to ask at this point is whether real-time scheduling can be approximated as well as bin packing. The algorithm of Karmarkar and Karp [8] computes a solution to the bin packing problem in polynomial time, which has an *additive* approximation guarantee. More precisely, given an instance I the algorithm computes a solution $APX(I)$ with $APX(I) - OPT(I) \leqslant O(\log^2(OPT(I)))$. An analogous result cannot hold for real-time scheduling unless $P = NP$.

Theorem 2. *If $P \neq NP$, there is no $\varepsilon > 0$ such that there exists a polynomial algorithm which computes an approximate solution $APX(\mathcal{T})$ for each instance $\mathcal{T}$ with*

$$APX(\mathcal{T}) - OPT(\mathcal{T}) \leqslant |\mathcal{T}|^{1-\varepsilon}.$$

Proof. The proof of this theorem is by reduction from 3-PARTITION. An instance of 3-PARTITION is a multiset of $3 \cdot n$ numbers $a_1, \ldots, a_{3n} \in \mathbb{R}_+$. The problem is to decide, whether this set can be partitioned into triples, such that the sum of the numbers of each triple is exactly one. 3-PARTITION is strongly NP-complete see [6]. The idea is now to construct a set of tasks $\mathcal{T} = \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ such that the following holds.

a) All tasks in $\mathcal{T}_j$ have the same period and $\mathcal{T}_j$ consists of $3n$ tasks with utilization $a_1, \ldots . a_{3n}$ respectively.
b) If a subset $\mathcal{T}' \subseteq \mathcal{T}$ contains 3 tasks and the periods of the tasks in $\mathcal{T}'$ are not all equal, then $\mathcal{T}'$ is infeasible.

With such a construction at hand one needs $k \cdot n$ processors if 3-PARTITION has a solution while one needs at least $n \cdot k + k/2$ processors if 3-PARTITION does not have a solution. If there exists an algorithm which computes a solution $APX(\mathcal{T})$ with $APX(\mathcal{T}) - OPT(\mathcal{T}) \leqslant (3 \cdot k \cdot n)^{1-\varepsilon}$ for some $\varepsilon > 0$, then one could use it to test whether 3-PARTITION has a solution, since $(3 \cdot k \cdot n)^{1-\epsilon} < k/2$ for $k = \Omega(n^{1/\varepsilon})$.

What remains to show, is how to construct such an instance $\mathcal{T} = \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ as above. If we define new weights $a'_i = \frac{m/3 + a_i}{m+1}$, then this new instance of 3-PARTITION is equivalent to $a_1, \ldots, a_{3n}$, since three new weights sum up to one if and only if the corresponding old weights sum up to one. This shows that we can assume that the weights $a_1, \ldots, a_{3n}$ are between $1/3 - 1/m$ and $1/3 + 1/m$ for an arbitrary $m \in \mathbb{Z}^+$.

Next we consider the periods $p_j = 1 + j/(4 \cdot k)$ for $j = 1, \ldots, k$. Those periods are between $1 + 1/(4 \cdot k)$ and $1 + 1/4$. The group $\mathcal{T}_j$ consists now of tasks having period p_j and utilization $a_1, \ldots, a_{3n}$ respectively, which implies a). To show b), consider a set $\mathcal{T}' = \{T_1, T_2, T_3\}$ of three tasks, where the period of T_3 is strictly larger than the period of T_1. We argue that T_3 is infeasible, if $\mathcal{T}'$ is scheduled on one processor. Consider a fix-point r of the response function of T_3

$$r = c(T_3) + \left\lceil \frac{r}{p(T_1)} \right\rceil c(T_1) + \left\lceil \frac{r}{p(T_2)} \right\rceil c(T_2). \tag{2}$$

Since $c(T_3) = \text{util}(T_3) \cdot p(T_3)$ one has $r \geqslant c(T_3)/(1 - \text{util}\{T_1, T_2\})$ which implies that $r \geqslant p(T_3)(1 - 9/(m+6))$. Notice that $p(T_3) \geqslant p(T_1) + 1/(4k)$, thus $p(T_3)/p(T_1) \geqslant 1 + 1/(4k \cdot p(T_1)) \geqslant 1 + 1/(5k)$. If one chooses $m = 90k$, then $r/p(T_1) > 1$ and it follows that $\lceil \frac{r}{p(T_1)} \rceil$ in (2) is at least 2. This means that $r \geqslant c(T_3) + 2c(T_1) + c(T_2) \geqslant 4(1/3 - 1/m)$ which is larger than $5/4 \geqslant p(T_3)$. This implies that T_3 is infeasible. □

Corollary 3. *Unless $P = NP$, there does not exist an asymptotic FPTAS for real-time scheduling.*

Proof. An asymptotic FPTAS [7] is an algorithm, whose running time is polynomial in n and $1/\varepsilon$ and which yields a solution of cost at most $(1+\varepsilon) \cdot OPT + p(1/\varepsilon)$ for some polynomial p. Assume that such an asymptotic FPTAS exists. We

assume w.l.o.g. that $p(1/\varepsilon) = \varepsilon^{-\alpha}$ for a fixed exponent $\alpha > 0$. Then with $\varepsilon = (1/n)^{1/(2\alpha)}$ the algorithm computes a solution with

$$APX - OPT \leqslant \varepsilon \cdot OPT + (1/\varepsilon)^{\alpha} \leqslant n^{1-1/(2\alpha)} + n^{1/2},$$

which is a contradiction to Theorem 2. □

4 Local Feasibility and an Algorithm to Schedule Large Tasks

Consider the response function (1) for a task T. For local feasibility of T, the tasks T' with $p(T') \leqslant \varepsilon \cdot p(T)$ contribute only with their utilization to the response function and the rounding operation in (1) is ignored. Thus the *local response function* $f_T^{local}(r)$ is defined as

$$c(T) + r \cdot \operatorname{util}(\{T' \colon p(T') \leqslant \varepsilon \cdot p(T)\}) + \sum_{\substack{T' \in \mathcal{T} \setminus \{T\} \\ \varepsilon \cdot p(T) < p(T') \leqslant p(T)}} \lceil r/p(T') \rceil \cdot c(T'). \quad (3)$$

The task T is *local feasible*, if there exists a number $p(T)/2 \leqslant r_T^* \leqslant p(T)$ with $f_T^{local}(r_T^*) \leqslant r_T^*$. In other words, the contribution of the rounding operation is only taken into account for tasks which are *close* or *local* to the task in consideration. The other tasks contribute only with their utilization.

We now show that, if an assignment is locally feasible (each task is locally feasible), then it is feasible on processors of speed $1 + 2\varepsilon$. We can therefore relax feasibility to local feasibility, which will later allow us to optimally distribute large tasks.

Lemma 4. *If a set of tasks $\mathcal{T}$ is local feasible on one processor, then it is feasible on a processor of speed $1 + 2\varepsilon$.*

Proof. Let r_T^* be the certificate for local feasibility of $T \in \mathcal{T}$, i.e., one has $p(T)/2 \leqslant r_T^* \leqslant p(T)$ and $f_T^{local}(r_T^*) \leqslant r_T^*$. It is enough to show that $f_T(r_T^*) \leqslant (1 + 2\,\varepsilon) f_T^{local}(r_T^*)$ holds. The difference between $f_T(r_T^*)$ and $f_T^{local}(r_T^*)$ can be bounded by

$$\sum_{T' \colon p(T') \leqslant \varepsilon \cdot p(T)} c(T').$$

Since $1 \leqslant 2\,\varepsilon \cdot r_T^*/p(T')$ this difference is bounded by

$$2\,\varepsilon\, r_T^* \cdot \sum_{T' \in \mathcal{T} : p(T') \leqslant \varepsilon \cdot p(T)} c(T')/p(T') \quad \leqslant \quad 2\,\varepsilon \cdot f_T^{local}(r_T^*)$$

and the result follows. □

4.1 A Dynamic Program to Schedule Large Tasks

We describe now an algorithm which optimally distributes a set of tasks in in polynomial time if we additionally assume that each utilization is bounded from below by the constant ε and an increase of speed by $1 + O(\varepsilon)$ is allowed.

If we round all running times $c(T)$ down such that the utilization of T becomes the nearest integer multiple of ε^2, then due to the reason that each $c(T)/p(T)$ is at least ε, a feasible schedule for the new task system yields a feasible assignment for the original task system, if the machines have speed $1 + O(\varepsilon)$. Therefore we can also assume that each task T has utilization $c(T)/p(T) \in \varepsilon^2\mathbb{Z}$.

Let $\mathcal{B}_1, \ldots, \mathcal{B}_\mu$ be the block-partitioning of the task system $\mathcal{T} = \{T_1, \ldots, T_n\}$ (see section 2). How many different types of tasks, can be present in one block $\mathcal{B}_i$? The number of different periods of $\mathcal{B}_i$ is bounded by $1/\varepsilon^3$. The number of different utilization-values of tasks in $\mathcal{T}$ is bounded by $1/\varepsilon^2$. Therefore, the number of different types of tasks in each block is bounded by a constant. The tasks are distributed with a dynamic programming algorithm to compute an optimal assignment of $\mathcal{T}$ such that each task is locally feasible. This is done, block-wise.

A vector $a = (a_0, ..., a_{1/\varepsilon^2})$ with $a_i \in \mathbb{Z}$ is called a *configuration*, whereby a_i denotes the number of processors whose utilization is exactly $i \cdot \varepsilon^2$. We require that $\sum_i a_i = n$. Consider the following table entries.

$$A(a, \ell) = \begin{cases} 1 \text{ if tasks in } \mathcal{B}_1, ..., \mathcal{B}_\ell \text{ can be scheduled in a locally feasible way} \\ \quad \text{such that utilization bounds of configuration } a \text{ are met} \\ 0 \text{ otherwise} \end{cases}$$

Note that a has fixed dimension, thus the table has a polynomial number of entries. We now describe, how to compute $A(a, \ell)$ efficiently. Let $b = (b_0, \ldots, b_{1/\varepsilon^2})$ be a processor configuration from a distribution of the tasks $\mathcal{B}_1, \ldots, \mathcal{B}_{\ell-1}$. Then $\text{LocalRTS}(\mathcal{B}_\ell, b, a)$ is defined to be 1, if the tasks in block $\mathcal{B}_\ell$ can be additionally distributed among the processors, such that the bounds of configuration a are met. The base cases are

$$A(a, 1) = \text{LocalRTS}(\mathcal{B}_1, (n, 0, ..., 0), a)$$

For all $\ell > 1$ note that $A(a, \ell) = 1$ if and only if there exists a $b \in \mathbb{Z}^{1/\varepsilon^2+1}$ with $0 \leqslant b_i \leqslant a_i$ for all i and

$$A(b, \ell - 1) = 1 \quad \text{and} \quad \text{LocalRTS}(\mathcal{B}_\ell, b, a) = 1$$

After computing all entries, the optimal number of processors can be read out of the table.

It remains to show, how to determine LocalRTS efficiently. The block $\mathcal{B}_\ell$ has only a constant number of different task-types, each having a utilization, which is lower-bounded by a constant. Suppose that $\mathcal{B}_\ell$ has tasks, whose running-time and period are from the tuples $(c_1, p_1), \ldots, (c_k, p_k)$. A *pattern* is a vector $(x_1, \ldots, x_k) \in \mathbb{N}_0^k$ which represents a set of tasks with these types of total utilization at most 1 (the set, defined by the pattern, contains x_i times task type

(c_i, p_i)). There is only a constant number of patterns, which can be used to distribute the tasks in $\mathcal{B}_\ell$. This shows that LocalRTS can be computed in polynomial time with integer programming in fixed dimension [9]. Details of the model are described in the full version of this paper. We have the following result.

Theorem 5. *Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a set of tasks and let $\varepsilon > 0$ be a constant such that $c(T)/p(T) \geqslant \varepsilon$ for all $T \in \mathcal{T}$. Then we can distribute the tasks using $OPT(\mathcal{T})$ many processors in polynomial time, such that the tasks on each processor are feasible if the processors have speed $1 + O(\varepsilon)$.*

5 Small Tasks

The well known approximation algorithms for bin packing [4,8] use the fact that small items of weight at most ε can first be discarded from the instance and then be added in a first-fit way after the larger items have been packed. If a new bin had to be opened to pack the small items, then the weight of each bin, except possibly the last bin, exceeds $1 - \varepsilon$. If m bins are then open, then $(m-1)(1-\varepsilon)$ is a lower bound on $OPT(I)$ which implies that $m \leqslant (1 + 2\varepsilon)\, OPT(I) + 1$.

For the real-time scheduling problem, an analogous approach to deal with tasks having small utilization does not work. This is again because a subset of tasks might be infeasible, even if its utilization only slightly exceeds ln(2). In this section we describe a tailored procedure to eliminate small tasks. It has two steps.

I) In a first step, we discard tasks and re-set periods such that the utilization of each period is at least ε^6. Here, the utilization of a period p is the sum of the utilization of the tasks having period p. The total utilization of the discarded tasks is bounded by $O(\varepsilon) \cdot \text{util}(\mathcal{T})$.
II) In a second step we cluster small tasks of the same period into groups, each of which will be identified into one single task having utilization ε^6.

After these discards, re-setting of periods and identification of small tasks, we obtain a new instance $\widetilde{\mathcal{T}}$. If OPT denotes the minimum number of processors to feasibly schedule $\mathcal{T}$, then $\widetilde{\mathcal{T}}$ can be scheduled using $(1+O(\varepsilon))\cdot OPT+1$ processors of speed $1 + O(\varepsilon)$. The next sections describe these two steps in detail.

Periods with Small Utilization

Let p be a period of a task in $\mathcal{T}$. The *utilization* of this period is the sum of the utilizations of tasks, having period p

$$\text{util}(p) = \sum_{T \in \mathcal{T} : p(T) = p} c(T)/p.$$

Suppose now that $\mathcal{B}_1, \ldots, \mathcal{B}_\mu$ is the partitioning of $\mathcal{T}$ into blocks and let $\mathcal{B}_i$ be the first block having utilization $\leqslant \varepsilon^2$. Let j be minimal such that $\text{util}(\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j) \geqslant$

ε^2. If this utilization is larger than ε, then we discard $\mathcal{B}_i, \ldots, \mathcal{B}_{j-1}$ from $\mathcal{T}$. Otherwise, we re-set the period of each task to an arbitrary value sandwiched between the smallest and the largest period of a task in $\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j$. Thereby the utilization of this period is at least ε^2. We repeat this procedure until such a block $\mathcal{B}_i$ having utilization ε^2 cannot be found anymore. The utilization of the tasks which are discarded with this procedure is bounded by $\varepsilon \cdot \text{util}(\mathcal{T})$. With first-fit, these tasks can be scheduled on $O(\varepsilon) \cdot OPT + 1$ additional processors.

Define $p_{\min}(\mathcal{T}) = \min\{p(T) \mid T \in \mathcal{T}\}$ and $p_{\max}(\mathcal{T}) = \max\{p(T) \mid T \in \mathcal{T}\}$. The next lemma shows that re-setting the periods of the tasks in $\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j$ to an arbitrary period in $[p_{\min}(\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j), p_{\max}(\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j)]$ is a feasible operation, if we are to run the tasks on machines of speed $1 + O(\varepsilon)$. More precisely, the lemma implies that, if the tasks could be scheduled on k machines before the re-setting operation, then they can be scheduled on k machines of speed $1+O(\varepsilon)$ after the re-setting operation.

Lemma 6. *Suppose that $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ is a feasible task system with the property that $p_{\max}(\mathcal{T}_i) \leqslant \varepsilon \cdot p_{\min}(\mathcal{T}_j)$ whenever $i < j$. Let $I \subseteq \{1, \ldots, k\}$ be a set of indices i with $\text{util}(\mathcal{T}_i) \leqslant \varepsilon$ and let $\mathcal{T}^*$ be an instance emerging from $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ by assigning for each $i \in I$ to each task $T \in \mathcal{T}_i$ an arbitrary period in $[p_{\min}(\mathcal{T}_i), p_{\max}(\mathcal{T}_i)]$ while keeping the utilization of the tasks invariant. The tasks $\mathcal{T}^*$ are feasible on a processor with speed $1 + O(\varepsilon)$.*

Proof. By Lemma 4 it is enough to show that each such changed task is locally feasible on a processor of speed $1 + O(\varepsilon)$. For this purpose, suppose that $T \in \mathcal{T}_i$ and let T^* be the task stemming from T by changing its period. Furthermore let $\mathcal{T}_i^*$ be the changed tasks $\mathcal{T}_i$. Lemma 1(iii) shows that $(\mathcal{T} \setminus \mathcal{T}_i) \cup \mathcal{T}_i^*$ is feasible on a processor of speed $1 + O(\varepsilon)$. Thus, after changing the periods in $\mathcal{T}_i$ only, the system is feasible on a processor of speed $1 + O(\varepsilon)$.

In particular T^* is local feasible on a processor of speed $1 + O(\varepsilon)$. Scaling the periods in the other sets $\mathcal{T}_j$, $j \neq i$ leaves the local response function for T^* unchanged. This shows the claim. □

After applying the procedure described above, the situation is now as follows. Each block of the task system has utilization at least ε^2. Choose $\gamma = \varepsilon^6$ and remove the tasks of all periods having utilization less than γ. Recall that the number of periods in each block is bounded by $1/\varepsilon^3$, thus we remove a utilization of at most $\gamma/\varepsilon^3 = \varepsilon^3$ from each block. Comparing this to the total utilization of each block, one observes that this removed tasks can be scheduled, using again $O(\varepsilon) \cdot OPT + 1$ many extra processors.

Periods with Large Utilization

Each period has now a utilization of at least $\gamma = \varepsilon^6$. Next, we partition $\mathcal{T}$ into $\mathcal{T}_{large}, \mathcal{T}_1, \ldots, \mathcal{T}_q$ such that $\mathcal{T}_{large}$ contains all tasks with utilization at least γ, the tasks in $\mathcal{T}_i$ have the same period p_i and $\gamma \leqslant \text{util}(\mathcal{T}_i) \leqslant 3 \cdot \gamma$.

The idea is now to treat the tasks in the sets $\mathcal{T}_i$ with period p_i as one single task having period p_i and utilization $\text{util}(\mathcal{T}_i)$. By doing so, we lose some flexibility

to distribute small tasks. Those belonging to one group must be assigned to the same processor. The next theorem establishes that we do not loose too much in terms of optimality, if we again allow processors of speed $1+O(\varepsilon)$. The theorem is proved by applying a Chernoff-type argument.

Theorem 7. *Let $\gamma = \varepsilon^6$ and let $\mathcal{T}$ be a set of tasks which can be partitioned into subsets $\mathcal{T}_{large}, \mathcal{T}_1, \ldots, \mathcal{T}_q$ such that the following conditions hold.*

a) $\mathcal{T}_{large}$ contains all tasks with utilization at least γ.
b) The tasks in $\mathcal{T}_i$ have the same period p_i and $\gamma \leqslant \text{util}(\mathcal{T}_i) \leqslant 3 \cdot \gamma$.

If $\mathcal{T}'$ denotes the instance stemming from identifying each set $\mathcal{T}_i$ as one task with period p_i and running time $\sum_{T \in \mathcal{T}_i} c(T)$, then for $\varepsilon \leqslant \frac{1}{3}$ one can schedule $\mathcal{T}'$ on

$$(1 + O(\varepsilon)) \cdot OPT(\mathcal{T}) + 1$$

machines of speed $1 + O(\varepsilon)$.

Proof. We have to show that there exists a solution which uses at most $(1 + O(\varepsilon))OPT(\mathcal{T}) + 1$ processors of speed $1 + O(\varepsilon)$, in which the tasks of each $\mathcal{T}_i$ are scheduled together on one processor. To do so, consider an optimal schedule for $\mathcal{T}$ which uses the processors $P_1, \ldots, P_k$. Clearly, we can identify the tasks $S \subseteq \mathcal{T}_i$ which are assigned to the same processor P_j into one task with period p_i and processing time $\sum_{T \in S} c(T)$. Therefore, we can assume that each processor contains at most one task from each set $\mathcal{T}_i$. In the new solution the tasks in each set $\mathcal{T}_i$ are scheduled on one processor. This is done using randomization. If a processor does not contain a task from $\mathcal{T}_i$, then $\mathcal{T}_i$ will not be assigned to it. Otherwise suppose that $T \in \mathcal{T}_i$ is assigned to the processor P. The probability that all tasks in $\mathcal{T}_i$ are assigned to P will be $\text{util}(T)/\text{util}(\mathcal{T}_i)$.

For a task $T \in \mathcal{T}$ let E_T be event that $f_T(r_T^*)$ exceeds $(1 + 2\varepsilon) \cdot r_T^*$. We next show that the probability of E_T is bounded by ε. This means that the expected utilization of the tasks which exceed their deadline even on the faster processors is bounded by $\varepsilon \cdot \text{util}(\mathcal{T})$. By removing those tasks and by scheduling them on a set of new processors in a first-fit manner, we only require an additional number of at most $(2/\ln 2) \cdot \varepsilon \cdot OPT(\mathcal{T}) + 1$ processors and the result follows.

We show this first for a task $T \in \mathcal{T}_{large}$. Suppose $T \in \mathcal{T}_{large}$ is assigned to processor P. Let $I \subseteq \{1, \ldots, q\}$ be the set of indices corresponding to the sets $\mathcal{T}_i$ whose tasks T_i on P have higher priority than T. Let $\mathcal{T}'_{large}$ be the set of tasks in $\mathcal{T}_{large}$ that lie on P and have higher priority than T. To bound $\Pr[E_T]$ we inspect the response function

$$f_T(r) = c(T) + \sum_{T' \in \mathcal{T}'_{large}} \left\lceil \frac{r}{p(T')} \right\rceil \cdot c(T') + r \cdot \sum_{i \in I} \frac{p_i}{r} \cdot \left\lceil \frac{r}{p_i} \right\rceil \cdot c(T_i)/p_i.$$

Since T meets its deadline, there exists a number r_T^* with $p(T)/2 \leqslant r_T^* \leqslant p(T)$ and $f_T(r_T^*) \leqslant r_T^*$. From this, we can conclude that the number $a_i = \frac{p_i}{r_T^*} \cdot \lceil r_T^*/p_i \rceil$ satisfies $1 \leqslant a_i \leqslant 2$.

After randomly redistributing the tasks in $\mathcal{T}_1, \ldots, \mathcal{T}_q$ the evaluation of the response function at r_T^* is a random variable of the form

$$c(T) + \sum_{T' \in \mathcal{T}'_{large}} \left\lceil \frac{r_T^*}{p(T')} \right\rceil \cdot c(T') + r_T^* \cdot \sum_{i \in I} a_i \cdot X_i$$

where the $X_i \in \{0, \mathrm{util}(\mathcal{T}_i)\}$ are independent random variables. For $X := \sum a_j X_j$ one has $E[X] \leqslant 1$. It is sufficient to show that $\Pr[X \geqslant E[x] + \varepsilon] \leqslant \varepsilon$. This can be done with a variant of the Chernoff bound. Choose

$$\alpha := \max_i \{a_i \cdot \mathrm{util}(\mathcal{T}_i)\} \leqslant 2 \cdot 3\varepsilon^6 = 6\varepsilon^6.$$

A Chernoff-type argument yields, that

$$\Pr[X \geqslant E[X] + \varepsilon] = \Pr\left[X \geqslant \left(1 + \frac{\varepsilon}{E[X]}\right) E[X]\right] \leqslant e^{-\frac{1}{6\varepsilon^6} \frac{\varepsilon^2}{3E[X]^2} E[X]} \leqslant \varepsilon,$$

where the last inequality follows from $E[X] \leqslant 1$ and $\varepsilon \leqslant 1/3$.

If $T \in \mathcal{T}_i$ for some i, then the above analysis can be applied after the observation that $c(T)$ grows at most up to $3 \cdot \gamma \cdot p(T)$. This can be bounded by $6 \cdot \gamma \cdot r_T^*$ which is bounded by $\varepsilon \cdot r_T^*$. □

By combining the treatment of periods with small utilization and periods with large utilization, we obtain the main result of this section.

Theorem 8. *Let $\mathcal{T}$ be a set of tasks and $\varepsilon < 1/3$. There is a polynomial time algorithm which discards a subset $\mathcal{T}'$ with $\mathrm{util}(\mathcal{T}') \leqslant O(\varepsilon) \cdot \mathrm{util}(\mathcal{T})$, constructs an instance $\widetilde{\mathcal{T}}$ such that each task of $\widetilde{\mathcal{T}}$ has utilization of at least ε^6 and $\widetilde{\mathcal{T}}$ can be scheduled on $(1+O(\varepsilon)) \cdot OPT(\mathcal{T}) + 1$ processors of speed $1 + O(\varepsilon)$. Furthermore, each feasible packing of $\widetilde{\mathcal{T}} \cup \mathcal{T}'$ on k processors of speed $1 + O(\varepsilon)$ yields a feasible packing of the original task set $\mathcal{T}$ on k processors of speed $1 + O(\varepsilon)$.*

Notice that we had to discard tasks of utilization $O(\varepsilon) \cdot \mathrm{util}(\mathcal{T})$ processors three times in this paper. If we collect all discarded tasks and then schedule this tasks once in a first-fit manner, this requires at most $O(\varepsilon) \cdot OPT + 1$ processors. Thus by combining Theorem 8 with Theorem 5, we obtain the main result of this paper.

Theorem 9. *For each $\varepsilon > 0$ there exists a polynomial time algorithm, which partitions a set of tasks $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_k$ such that each $\mathcal{T}_i$ is feasible on a processor of speed $1 + O(\varepsilon)$ and $k \leqslant (1 + O(\varepsilon)) \cdot OPT + 1$.*

References

1. Baruah, S., Goossens, J.: Scheduling real-time tasks: Algorithms and complexity. In: Leung, J.Y.-T. (ed.) Handbook of Scheduling — Algorithms, Models, and Performance Analysis, ch. 28. Chapman & Hall/CRC, Boca Raton (2004)

2. Davari, S., Dhall, S.K.: On-line algorithms for allocating periodic-time-critical tasks on multiprocessor systems. Informatica (Slovenia) 19(1) (1995)
3. Dhall, S.K.: Approximation algorithms for scheduling time-critical jobs on multiprocessor systems. In: Leung, J.Y.-T. (ed.) Handbook of Scheduling — Algorithms, Models, and Performance Analysis, ch. 32, Chapman & Hall/CRC, Boca Raton (2004)
4. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica 1(4), 349–355 (1981)
5. Fisher, N., Baruah, S.: A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In: ECRTS 2005: Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS 2005), pp. 117–126. IEEE Computer Society, Los Alamitos (2005)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, San Francisco (1979)
7. Johnson, D.S.: The NP-completeness column: an ongoing guide. J. Algorithms 13(3), 502–524 (1992)
8. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: 23rd annual symposium on foundations of computer science, Chicago, Ill., pp. 312–320. IEEE, New York (1982)
9. Lenstra, H.W.: Integer programming with a fixed number of variables. Mathematics of Operations Research 8(4), 538–548 (1983)
10. Liebeherr, J., Burchard, A., Oh, Y., Son, S.H.: New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. Comput. 44(12), 1429–1442 (1995)
11. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20(1), 46–61 (1973)
12. Oh, Y., Son, S.H.: Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9(3), 207–239 (1995)
13. Schuurman, P., Woeginger, G.: Approximation schemes – a tutorial. In: Möhring, R.H., Potts, C.N., Schulz, A.S., Woeginger, G.J., Wolsey, L.A. (eds.) Lectures on Scheduling (to appear, 2007)

Optimal Monotone Encodings*

Noga Alon** and Rani Hod

Schools of Mathematics and Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel
{nogaa,ranihod}@post.tau.ac.il

Abstract. Moran, Naor and Segev have asked what is the minimal $r = r(n,k)$ for which there exists an (n,k)-*monotone encoding of length* r, i.e., a monotone injective function from subsets of size up to k of $\{1,2,\ldots,n\}$ to r bits. Monotone encodings are relevant to the study of tamper-proof data structures and arise also in the design of broadcast schemes in certain communication networks.

To answer this question, we develop a relaxation of k-superimposed families, which we call α-fraction k-multi-user tracing ((k,α)-FUT families). We show that $r(n,k) = \Theta(k\log(n/k))$ by proving tight asymptotic lower and upper bounds on the size of (k,α)-FUT families and by constructing an (n,k)-monotone encoding of length $O(k\log(n/k))$.

We also present an explicit construction of an $(n,2)$-monotone encoding of length $2\log n + O(1)$, which is optimal up to an additive constant.

1 Introduction

In their pursuit of history-independent schemes that use a write-once memory, motivated by cryptographic applications, Moran et al. [14] have considered monotone injective functions that map subsets of size up to k of $[n]$ into $2^{[r]}$ (all subsets of $[r]$), henceforth called (n,k)-monotone encodings of length r, or $\mathrm{ME}(n,k,r)$. They have shown the existence of an (n,k)-monotone encoding of length $O(k\log n\log(n/k))$ and raised the question of determining the minimal $r = r(n,k)$ for which an $\mathrm{ME}(n,k,r)$ exists.

A counting argument shows that $r(n,k) \geq \log\left(\sum_{i=0}^{k}\binom{n}{i}\right) = \Omega(k\log(n/k))$ is required for any injective encoding, without even considering monotonicity. In this paper, we show that a monotone encoding of length $O(k\log(n/k))$ exists, establishing that $r(n,k) = \Theta(k\log(n/k))$, thus settling the open problem raised in [14]. We limit ourselves to $k \leq n/2$ since the trivial identity encoding is optimal for $k > n/2$.

Throughout the paper we use $[n]$ to denote $\{1,2,\ldots,n\}$. We denote subsets of $[n]$ of size k and up to k by $\binom{[n]}{k}$ and $\binom{[n]}{\leq k}$, respectively. All logarithms are binary unless stated otherwise. Floor and ceiling signs are omitted whenever these are not crucial.

* Due to space limitations we refer the reader to a longer version available at http://www.math.tau.ac.il/∼nogaa/PDFS/publications.html

** Research supported in part by the Israel Science Foundation and by a USA-Israeli BSF grant.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 258–270, 2008.

Paper Organization. In the rest of this section we present our contribution and consider previous work. In Section 2 we present a probabilistic construction of FUT families and a deterministic construction of ME based on FUT families. In Section 3 we prove a lower bound on the length of FUT families and another lower bound on the length of ME. In Section 4 we present an explicit construction of an ME$(n, 2, 2\log n + O(1))$. Most proofs have been omitted from this extended abstract due to lack of space.

1.1 A First Attempt: Superimposed Families

A general representation of a monotone function f is $f(S) = \bigcup_{S' \subseteq S} g(S')$ for some function $g(S)$. For f to be injective as well, we need all the relevant unions to be distinct.

A family of subsets of $[r]$ is called k-superimposed if all the unions of up to k sets from it are distinct. Clearly, a k-superimposed family $\mathcal{F} = \{A_i\}_{i=1}^n$ of cardinality n translates to an ME(n, k, r) f defined by $f(S) = \bigcup_{i \in S} A_i$.

Probabilistic and explicit constructions of k-superimposed families of cardinality n are known for $r = O\left(k^2 \log(n/k)\right)$ (see, for example, [7,8,1]), yielding the same upper bound on the length of (n, k)-monotone encodings. However, in [7,16,9] it was shown that for $n > k^2$, k-superimposed families require $r = \Omega((k^2/\log k)\log n)$; Thus, an approach based solely on k-superimposed families will not achieve optimal monotone encodings.

Inspecting the monotone encoding induced by a k-superimposed family, we observe that only the "linear" terms $g(\{i\}) = A_i$ are non-empty. In a way, using "higher-order" terms can be regarded as a form of adaptive encoding (obtained in a non-adaptive fashion) since collisions in the unions of lower-order terms can be resolved by a higher-order term.

1.2 Our Contribution

A general monotone encoding does not need the strict distinct-unions requirement of superimposed families. We consider the following relaxation of superimposed families.

Definition 1. *Let $\mathcal{F} = \{A_i\}_{i=1}^n$ be a family of subsets of $[r]$ and let $S \subseteq [n]$. We denote $\bigcup_{i \in S} A_i$ by A_S. An element $j \in S$ is said to be $\mathcal{F}$-identifiable (with respect to S) if A_j has a unique element not present in any other subset $A_i \in \mathcal{F}$ that is covered by A_S; that is, if $A_j \not\subseteq \bigcup\{A_i \in \mathcal{F} : i \neq j, A_i \subseteq A_S\}$. An element $j \in S$ is said to be $\mathcal{F}$-obscured (with respect to S) if it is not $\mathcal{F}$-identifiable.*

Definition 2. *Let $k \geq 2$, $n \geq 2k$ and $0 < \alpha < 1$. A family $\mathcal{F} = \{A_i\}_{i=1}^n$ of subsets of $[r]$ is called α-fraction k-multi-user tracing, or (k, α)-FUT, if for any $S \in \binom{[n]}{\leq k}$, more than $\alpha|S|$ of its elements are $\mathcal{F}$-identifiable.*[1]

We prove almost tight upper and lower bounds on $(k, 1-\epsilon)$-FUT families.

[1] Or all of them, if $|S| \leq 1/\alpha$.

Theorem 1. *There exists a constant $c_1 > 0$ such that for all $k \geq 2$, $n \geq 2k$ and $\frac{1}{k} \leq \epsilon \leq \frac{1}{2}$ there exists a $(k, 1-\epsilon)$-FUT family of cardinality n, where $r = c_1(k/\epsilon)\log(n/k)$.*

Theorem 2. *There exists a constant $c_2 > 0$ such that for all $k \geq 2$, $\frac{1}{k} \leq \epsilon \leq \frac{1}{2}$, any $(k, 1-\epsilon)$-FUT family of cardinality $n \geq k/\epsilon$ must have $r \geq c_2 \frac{k/\epsilon}{\log(k/\epsilon)} \log n$.*

Note that for $0 < \epsilon \leq \frac{1}{k}$ any $(k, 1-\epsilon)$-FUT family is actually k-superimposed since the number of obscured elements is strictly less than one. Substituting $\epsilon = \frac{1}{k}$ in Theorems 1 and 2 yields the known asymptotic upper and lower bounds for k-superimposed families.

Back to monotone encodings, we form an optimal $\mathrm{ME}(n, k, O(k\log(n/k)))$ by chaining $\left(2^{-t}k, \frac{1}{2}\right)$-FUT families of cardinality n for $t = 0, 1, \ldots, \log k$. This yields the following theorem.

Theorem 3. *There exists a constant $c_3 > 0$ such that for all integers $n \geq 4$ and $2 \leq k \leq n/2$ there exists an (n, k)-monotone encoding of length $r = c_3 k \log(n/k)$.*

Definition 3. *For integers $0 \leq k \leq n$ we denote $\left\lceil \log\left(\sum_{i=0}^{k} \binom{n}{i}\right)\right\rceil$ by $\rho(n, k)$.*

We also present a lower bound on the length of monotone encodings.

Theorem 4. *There exists a constant $c_4 > 0$ such that $r(n, k) > (1 + c_4)\rho(n, k)$ for sufficiently large n and some $k = k(n)$.*

When k is small, constant factors may have a significant impact. In Section 4 we present an explicit construction for $k = 2$ that is optimal up to an additive constant, yielding the following theorem.

Theorem 5. *There exists a constant $c_5 > 0$ such that for all integers $n \geq 4$, there exists an explicit $(n, 2)$-monotone encoding of length $\rho(n, 2) + c_5$.*

1.3 Related Work

Single and Multi-user Tracing Families. Although we described (k, α)-FUT families as a relaxation of superimposed families, they can also be seen as an extension of single-user tracing (SUT) families, an even simpler relaxation of superimposed families introduced by Csűrös and Ruszinkó [6]. Given the union of up to k subsets of a SUT family, we are able to identify at least one of them. While the lower bound remains $\Omega(k\log(n/k))$, SUT families of cardinality n were shown by Alon and Asodi [2] to exist for $r = O(k\log(n/k))$.

Laczay and Ruszinkó [12] extended SUT families in another direction, considering j-out-of-k multi-user tracing (MUT_j) families, ensuring that given the union of up to k subsets we are able to identify at least j of them.[2] By definition, a MUT_1 family is equivalent to a SUT family; Alon and Asodi [3] proved that MUT_j families exist for $r = O\left((k + j^2)\log(n/k)\right)$, effectively creating $\mathrm{MUT}_{\sqrt{k}}$ families for the same cost as SUT families. Nevertheless, MUT_j families are also j-superimposed, hence we cannot use them for $j = \omega(\sqrt{k\log k})$ while maintaining linear dependence in k.

[2] Or all of them, if the given union is a union of less than j subsets.

Non-adaptive Conflict Resolution. Komlós and Greenberg [11] solved a problem similar to monotone encoding using techniques very similar to ours. They considered non-adaptive conflict resolution (NACR) in a multiple access OR channel. Here is a quick description of NACR.

> A communication channel is shared by n stations, some of which may want to broadcast a message. Multiple concurrent broadcasts cancel out and the stations involved are notified of this.
> Each station has a *scheme*, or a list of available time slots. Each active station will try to broadcast its message on every available time slot; it will deactivate if it succeeds (i.e., if it was the only station broadcasting on this time slot).
> An *epoch*, or a sequence of time-slots, is called successful if no active stations remain (due to successful broadcasting) until the epoch ends.
> A scheme set is valid if for all choices of initial k active stations, the epoch succeeds. What is the minimal length (in time slots) needed for a valid scheme set to exist?

Although the problem of monotone encodings can be reformulated in a similar language, two major differences exist between ME and NACR.

1. In NACR, stations are aware of their success/failure, i.e., they know whether there were 0, 1 or ≥ 2 concurrent broadcasts. In ME, an outside observer is required to identify active stations seeing only the channel activity indicator (0 or ≥ 1 broadcasts).
2. In NACR, an active station will stop once it has successfully broadcast its message. In ME, the situation is analogous to stations that remain active and cannot change their schemes. However, stations in ME are aware of each other, and are allowed to broadcast *more* if other stations are active.

For instance, the following valid NACR scheme set for $n = 3, k = 2$ uses three time slots: $S_1 = \{2, 3\}, S_2 = \{1, 3\}, S_3 = \{1, 2\}$. Nevertheless, the activity indicator of the channel gives no hint of *which* stations are active when any two of them are active!

Assume that the message each station broadcasts specifies its identifying number and consider the actual channel data rather than the channel activity indicator. This allows a successful broadcast to identify[3] the transmitting station. Thus, we may convert[4] an NACR solution to an ME at the cost of a factor of $\log n$. Although presented in a different perspective, the (n, k)-monotone encoding of length $O(k \log n \log(n/k)))$ shown in [14] proceeds essentially along these lines.

[3] Some action is needed to ensure that multiple concurrent broadcasts are not misinterpreted as valid messages. For instance, encode 0 as '01' and 1 as '10', doubling the length of the data.

[4] Further modifications are necessary to work out the second difference as well, but the length of the data remains unaffected.

Cryptographic Applications. In [14], monotone encodings are used to maintain a tamper-proof deterministic data structure that represents a subset of size up to k of $[n]$. Instead of relying on cryptographic assumptions, the data structure is made tamper-proof by storing it on a write-once memory, i.e., all bits are initially 0 and can only be turned to 1. This imposes the monotonicity requirement.

Since elements are inserted one by one, another security-motivated requirement is that the representation of the data structure is independent of the order in which elements are inserted (for example, to ensure privacy in voting schemes). This rules out "adaptive" solutions like writing down the elements sequentially using $\log n$ bits per element. This requirement is expressed in ME by taking $\binom{[n]}{\leq k}$ (that is, *unordered* subsets) as the domain of the encoding.

2 The Construction

2.1 FUT Families

The upper bound stated in Theorem 1 is implied by the following probabilistic construction.

Let $k \geq 2$ and $\frac{1}{k} \leq \epsilon \leq \frac{1}{2}$. Let $d = (2/\epsilon)\log(2en/k)$ and $r = 16kd = O\left((k/\epsilon)\log(n/k)\right)$. Let $h_1, \ldots, h_n$ be n random functions from $[d]$ to $[16k]$, i.e., the values $h_j(i)$ for $i \in [d]$ and $j \in [n]$ are chosen independently and equiprobably from $[16k]$ and let $\mathcal{F} = \{A_1, \ldots, A_n\} \subset 2^{[r]}$ be their representations as sets; that is, $A_j = \{16ki - h_j(i) + 1 : i \in [d]\} \subset [r]$.

Definition 4. *A family $\mathcal{F}$ of sets is said to have property A if for $t \in [2k]$ and for all distinct $A_1, \ldots, A_t \in \mathcal{F}$, A_j is covered by the union of $\{A_i : i \in [t], i \neq j\}$ for less than ϵt values of $j \in [t]$. In other words, more than $(1-\epsilon)t$ of the sets have a unique element in $\bigcup_{i=1}^{t} A_i$.*

Proposition 2.1. *With positive probability, property A holds for $\mathcal{F}$ as selected above.*

Proposition 2.2. *Any family for which property A holds is $(k, 1-\epsilon)$-FUT; Thus, with positive probability, $\mathcal{F}$ as selected above is $(k, 1-\epsilon)$-FUT.*

Proof. Let $S \subset [n], |S| \leq k$ and consider $I = \{i \in [n] : A_i \subseteq A_S\}$. Obviously, $S \subseteq I$. By the definition of I, all $i \in I \setminus S$ are $\mathcal{F}$-obscured since A_i is covered by A_S. Assume that $|I| \geq 2k$. By property A, applied to some subset $S \subset I' \subseteq I$ of cardinality $2k$, more than $(1-\epsilon)2k \geq k \geq |S|$ elements of I' are $\mathcal{F}$-identifiable, which is absurd as they all reside in S.[5] Thus, $|I| < 2k$. By property A, more than $(1-\epsilon)|I| \geq (1-\epsilon)|S|$ elements of I are $\mathcal{F}$-identifiable. Again, they all reside in S. □

[5] To be exact, these elements are $\mathcal{F}$-identifiable with respect to I' and not I, but it is all the same since $A_{I'} = A_S = A_I$.

2.2 Monotone Encodings

Equipped with the tool we have just developed, we move on to describing a function as stated by Theorem 3.

We construct $f : \binom{[n]}{\leq k} \to 2^{[r]}$ inductively. Initialize the construction[6] with the trivial case $k = 1$. Let $f' : \binom{[n]}{\leq k/2} \to 2^{[r']}$ be a monotone encoding for subsets of size up to $k/2$ and let $\mathcal{F} = \{A_i\}_{i=1}^{n} \subset 2^{[r'']}$ be a $(k, \frac{1}{2})$-FUT family. Shift $\mathcal{F}$ by r' to make its support disjoint from $[r']$. All involved sets are now subsets of the ground set $[r]$, where $r = r' + r''$. We define $f(S) = A_S \cup f'(S')$, where S' consists of all $\mathcal{F}$-obscured elements of S (note that S' is well-defined given $\mathcal{F}$).

Since a $(k, \frac{1}{2})$-FUT family exists for $r = 2c_1 k \log(n/k)$, the size of the ground set for the entire construction is

$$\sum_{t=0}^{\log k} 2c_1 2^{-t} k \log\left(\frac{n}{2^{-t}k}\right) \leq 2c_1 k \sum_{t=0}^{\infty} 2^{-t}\left(t + \log(n/k)\right) = O\left(k \log(n/k)\right) .$$

Proposition 2.3. *The function f is injective, i.e., $f(S) \neq f(T)$ for $S \neq T$.*

Proposition 2.4. *The function f is monotone, i.e., $f(S) \subseteq f(T)$ for $S \subseteq T$.*

3 Lower Bounds

3.1 FUT Families

First we show a lower bound of $\Omega(k \log(n/k))$ on the length of *constant* fraction user-tracing families.

Proposition 3.1. *For all $k \geq 2$, any $(k, \frac{1}{2})$-FUT family of cardinality $n \geq k$ must have $r \geq \frac{1}{4} k \log(n/k)$.*

Next, we establish the lower bound stated in Theorem 2 by using a modified version of a technique from [16]. As the bound of Proposition 3.1 holds for any $(k, 1-\epsilon)$-FUT family, $\epsilon \leq \frac{1}{2}$, we henceforth assume $\epsilon < \frac{1}{4}$. Let $k \geq 2$, $\frac{1}{k} \leq \epsilon < \frac{1}{4}$ and let $\mathcal{F} = \{A_i\}_{i=1}^{n} \subseteq 2^{[r]}$ be a $(k, 1-\epsilon)$-FUT family, where $n \geq k/\epsilon$. We modify $\mathcal{F}$ in two phases, as follows.

1. As long as $\mathcal{F}$ contains a set F of cardinality at least $\beta = 4r/k$, remove F from $\mathcal{F}$ and remove its elements from all other sets of $\mathcal{F}$. Call the resulting family $\mathcal{F}'$.
2. As long as $\mathcal{F}'$ contains a set F such that any 4ϵ-fraction of it is covered by some other set from $\mathcal{F}'$, and in particular, it is covered by the union of $t \leq 1/(4\epsilon)$ other sets $A_1, \ldots, A_t \in \mathcal{F}'$, remove F and $\{A_i\}_{i=1}^{t}$ from $\mathcal{F}'$. Call the resulting family $\mathcal{F}''$.

[6] Another way to initialize the construction is with an $\mathrm{ME}(n, \sqrt{k}, O(k \log(n/\sqrt{k})))$ induced by a $\sqrt{k}$-superimposed family as described in Section 1.1.

Claim 3.2. *Phase 1 stops after at most $k/4$ iterations.*

Proof. Every iteration discards at least β elements from the ground set. We begin with r elements, so we stop after at most $r/\beta = k/4$ iterations. □

Claim 3.3. *Phase 2 stops after less than ϵk iterations.*

We now have a family $\mathcal{F}''$ of cardinality greater than $n-k$ with the following property: every $F \in \mathcal{F}''$ has a subset of $4\epsilon|F|$ elements unique to F. Let $\gamma = 16\epsilon/k$. Every such unique subset is of cardinality $4\epsilon|F| \leq 4\epsilon\beta = \gamma r$; Thus,

$$n-k+1 \leq |\mathcal{F}''| \leq \left|\binom{[r]}{\leq \gamma r}\right| = \sum_{t=0}^{\gamma r}\binom{r}{t} \leq 1+\gamma r\binom{r}{\gamma r} \leq 1+\gamma r\left(\frac{re}{\gamma r}\right)^{\gamma r}.$$

Taking logarithms we get

$$\Omega(\log n) \leq (\log n)-1 \leq \log(n-k) \leq \gamma r \log\frac{e}{\gamma} + o(r) = O\left(r\frac{\epsilon}{k}\log\frac{k}{\epsilon}\right).$$

Therefore, $r = \Omega\left(\frac{k/\epsilon}{\log(k/\epsilon)}\log n\right)$.

3.2 Monotone Encodings

As we already stated, $r(n,k) \geq \rho(n,k)$ by a counting argument. The trivial identity encoding implies that $r(n,k) \leq n$. Obviously, $r(n,1) = \rho(n,1) = \lceil\log(1+n)\rceil$. Theorem 3 states that $r(n,k) = \Theta(\rho(n,k))$. In Section 4 we will prove that $r(n,2) \leq \rho(n,2)+O(1)$. The following simple proposition shows that sometimes $r(n,2) \geq \rho(n,2)+1$.

Proposition 3.4. $r(5,2) = 5 > 4 = \rho(5,2)$.

Proof. Assume that $r(5,2) \leq \rho(5,2) = \lceil\log(1+5+10)\rceil = 4$ for the sake of contradiction and let $f : \binom{[5]}{2} \to 2^{[4]}$ be an ME$(5,2,4)$. Without loss of generality, $f(\emptyset) = \emptyset$. There must be some $i \in [5]$ such that $|f(\{i\})| \geq 2$, since $\{f(\{i\})\}_{i=1}^{5}$ cannot all reside in $\binom{[4]}{1}$. Without loss of generality, assume that $\{1,2\} \subseteq f(\{1\})$.

We have reached a contradiction, as $f(\{1,2\}), f(\{1,3\}), f(\{1,4\}), f(\{1,5\})$ are 4 distinct subsets of $2^{[4]}$ that properly contain $\{1,2\}$. Thus, $r(5,2) \geq 5$. But obviously $r(5,2) \leq 5$, hence $r(5,2) = 5$. □

The proof of Proposition 3.4 extends to show that for some choices of n and k, $r(n,k)$ exceeds $\rho(n,k)$ by more than an additive constant.

Lemma 3.5. *Let $f : \binom{[n]}{\leq k} \to 2^{[n-1]}$ be an ME$(n,k,n-1)$ and let $0 \leq m \leq k$. Then, there exists $S \in \binom{[n]}{\leq m}$ such that $|f(S)| \geq 2m$.*

To establish the stronger result, we need the following corollary of Stirling's approximation formula, where $H(\alpha) = \alpha\log\frac{1}{\alpha} + (1-\alpha)\log\frac{1}{1-\alpha}$ is the binary entropy function.

Claim 3.6. $\log\binom{t}{\alpha t} = tH(\alpha) - \frac{1}{2}\log(2\pi\alpha(1-\alpha)t) + o(1)$.

Proposition 3.7. *There exists a constant $\delta > 0$ such that $r\left(n, \frac{1-\delta}{2}n\right) = n$ for sufficiently large n.*

Proof. Let $k = \frac{1-\delta}{2}n$ and let $m = \left(\frac{1}{2} - \delta\right)n = k - \frac{\delta}{2}n$. Assume for the sake of contradiction that $r(n,k) \leq n-1$ and let $f : \binom{[n]}{\leq k} \to 2^{[n-1]}$ be an ME$(n,k,n-1)$.

By Lemma 3.5, there exists some S of cardinality at most m such that $|f(S)| \geq 2m = (1-2\delta)n$. Consider $\{S \cup T : T \in \binom{[n]\setminus S}{(\delta/2)n}\}$. These are at least $\binom{n-m}{(\delta/2)n}$ sets in $\binom{[n]}{\leq k}$ whose images under f are all distinct and (properly) contain $f(S)$. Therefore, for sufficiently small δ and sufficiently large n,

$$\log\binom{n-m}{(\delta/2)n} = \log\binom{\left(\frac{1}{2}+\delta\right)n}{(\delta/2)n} > \left(\frac{1}{2}+\delta\right)nH\left(\frac{\delta}{1+2\delta}\right) - \frac{1}{2}\log n$$
$$> 2\delta n = n - 2m > |[n-1] \setminus f(S)|,$$

which is a contradiction. Thus, $r(n,k) \geq n$ and hence $r(n,k) = n$. □

Another useful entropy-related estimation we need is

Claim 3.8. $H\left(\frac{1-\delta}{2}\right) = 1 - \frac{\delta^2}{2\ln 2} + O(\delta^4)$.

We are now ready to prove the lower bound on $r(n,k)$ of Theorem 4.

Proof (of Theorem 4). Choose $k = k(n) = \frac{1-\delta}{2}n$, where δ is the constant from Proposition 3.7. By Claim 3.8,

$$\rho(n,k) = \log\left(\sum_{i=0}^{k}\binom{n}{i}\right) < nH\left(\frac{k}{n}\right) = nH\left(\frac{1-\delta}{2}\right) = n\left(1 - \frac{\delta^2}{2\ln 2} + O(\delta^4)\right)$$

Hence, $r(n,k) = n > (1+c_4)\rho(n,k)$ for large enough n and sufficiently small $c_4 > 0$. □

In Proposition 3.7 we needed δ to satisfy $\left(\frac{1}{2}+\delta\right)H(\frac{\delta}{1+2\delta}) > 2\delta$, e.g., pick $\delta = 0.2276$. Thus, the largest value of c_4 in Theorem 4 that follows from the proof is roughly 0.038.

4 Tighter Bounds for $k = 2$

The ME construction presented in Section 2 is optimal up to a constant factor. Yet, it is interesting to see how small this constant can get and whether we can beat MEs induced by superimposed families, even for small values of k where asymptotic superiority still does not apply.

Trivially, $r(n,1) = \lceil\log(n+1)\rceil$, as we just need n different non-empty subsets of $[r]$. Hence, the first interesting case is $k = 2$. The obvious lower bound[7] is

[7] As Proposition 3.4 demonstrated, $\rho(n,2)$ is not tight for some values of n.

$\rho(n,2) = \lceil 2\log n \rceil - 1 + o(1)$. We prove Theorem 5 by explicitly constructing an $(n,2)$-monotone encoding of length $\rho(n,2) + O(1)$.

In contrast to the $r(n,2) \leq 2\log n + O(1)$ bound of Theorem 5, Coppersmith and Shearer [5] have shown that for $r < (2.0008 - o(1))\log n$, no family $\{A_i\}_{i=1}^{n}$ of n subsets of $[r]$ exists for which $\{A_i \cup A_j\}_{1 \leq i < j \leq n}$ are all different.

4.1 Construction Time Again

A monotone function f over the domain $\binom{[n]}{\leq 2}$ can be defined by[8] $A_i = f(\{i\})$ for $i \in [n]$ and $A_{ij} = A_{ji} = f(\{i,j\}) \setminus (A_i \cup A_j)$ for $\{i,j\} \in \binom{[n]}{2}$. Assuming that $\{A_i : i \in [n]\}$ and $\{A_{ij} : \{i,j\} \in \binom{[n]}{2}\}$ have disjoint supports, it is sufficient to require three conditions for the function to be injective:

(*i*) $A_i \neq A_j$ for $i \neq j$.
(*ii*) $A_{ij} \neq \emptyset$ for $i \neq j$ if $\exists i'$ such that $A_i \cup A_j = A_{i'}$.
(*iii*) $A_{ij} \neq A_{i'j'}$ for $\{i,j\} \neq \{i',j'\}$ satisfying $A_i \cup A_j = A_{i'} \cup A_{j'}$.

As we now strive for a result optimal up to an additive constant, we cannot continue neglecting the effects of rounding.

Definition 5. *For $x \in \mathbb{R}$, define $\lfloor x \rceil = \{\lfloor x \rfloor, \lceil x \rceil\}$.*

Let $p = 1 - \frac{1}{\sqrt{2}} \approx 0.29$ and select minimal a and $b \in \lfloor pa \rceil$ subject to $\binom{a}{b} \geq n$. Select distinct $A_1, \ldots, A_n \in \binom{[a]}{b}$. Obviously, these satisfy condition (*i*). In addition, $|A_i \cup A_j| > b$ for $i \neq j$, so condition (*ii*) is satisfied as well, albeit in the null sense. Condition (*iii*) will be satisfied by an appropriate selection of $A_{ij} \in 2^{[s]}$, where $s = \lceil \log B_{max} \rceil$, $B_{max} = \max_{A \subseteq [a]} B(A)$ and $B(A) = \left| \left\{ \{i,j\} \in \binom{[n]}{2} : A_i \cup A_j = A \right\} \right|$. In simple words, we differentiate between the $B(A)$ pairs colliding at A by labeling each one with a unique number between 1 and $B(A) \leq B_{max} \leq 2^s$.

Thus, we only need to determine B_{max} to get an $(n,2)$-monotone encoding of length $r = a + s$. By symmetry,[9] $B(A)$ depends only on $|A|$. For $0 \leq m \leq a$, denote the value of $B(\lfloor m \rceil)$ by $B(m)$. Clearly,

$$B(m) = \begin{cases} \frac{1}{2}\binom{m}{b}\binom{b}{m-b} = \frac{m!}{2(m-b)!^2(2b-m)!}, & b < m \leq 2b, \\ 0, & \text{otherwise.} \end{cases}$$

Proposition 4.1. *$B(m)$ has a single maximum, achieved at $m^* \in \lfloor b/2p \rceil$.*

A Rough Estimate. We now bound the difference between r and the lower bound $\rho(n,2)$. First, we use Claim 3.6 to get a quick estimate, neglecting $o(1)$ terms and the effects of rounding. Let $n = \binom{a}{b}$ and recall that $m^* \approx b/2p \approx pa/2p = a/2$. Note also that $1 + \frac{1}{2}H(2p) = (2-p)H(p)$ for our choice of p. Hence,

[8] Without loss of generality we may assume $f(\emptyset) = \emptyset$.
[9] For this analysis we assume that $n = \binom{a}{b}$. If $n < \binom{a}{b}$, $B(A)$ will decrease for some values of A, but the maximum B_{max} should remain unaffected.

$$
\begin{aligned}
r - \rho(n,2) &\approx (a + \log B_{max}) - (2\log n - 1) \\
&= a + \log \binom{m^*}{b} + \log \binom{b}{m^* - b} - 1 - 2\log \binom{a}{b} + 1 \\
&\approx a + \log \binom{a/2}{pa} + \log \binom{pa}{(1-p)pa} - 2\log \binom{a}{pa} \\
&\approx a + \left[(a/2)H(2p) - \tfrac{1}{2}\log\left(2\pi 2p(1-2p)(a/2)\right)\right] \\
&\qquad + \left[paH(1-p) - \tfrac{1}{2}\log\left(2\pi(1-p)p^2a\right)\right] - 2\left[aH(p) - \tfrac{1}{2}\log\left(2\pi p(1-p)a\right)\right] \\
&= a\left(1 + \tfrac{1}{2}H(2p) + pH(p) - 2H(p)\right) + \tfrac{1}{2}\log\frac{(2\pi p(1-p)a)^2}{(2\pi p(1-2p)a)(2\pi(1-p)p^2a)} \\
&= \frac{1}{2}\log\frac{1-p}{p(1-2p)} = \log(1+\sqrt{2}) \approx 1.272\,.
\end{aligned}
$$

Next we delve into details to check where the estimation above is inaccurate. We lose a little due to the following reasons: (1) While $p \approx \frac{b}{a}$ is irrational, a and b must be integers; (2) If n is just a little bigger than $\binom{a}{b}$, we are forced to increase either a or b. It can be verified for small values of a and b that the $o(1)$ terms cause no further loss.

Proposition 4.2. *Let $n = \binom{a}{b}$. Then, the first loss is bounded by 10 bits.*

Proposition 4.3. *The second loss is bounded by one bit.*

Thus, we have proved that our construction is optimal up to an additive constant $c = 11$. Empirical results show that the first loss is always close to $\log(1+\sqrt{2}) \approx 1.272$, so the correct value of this constant is 3, but to avoid further complication in the proof we settled for the above estimate.

5 Concluding Remarks and Open Problems

5.1 Encoding and Decoding Algorithms

Proposition 2.3 fuels a recursive algorithm to decode S from $f(S)$:

1. Separate $f(S)$ to A_S and $f'(S')$.
2. Determine S' by running the algorithm recursively on $f'(S')$.
3. Find all sets $A_i \subseteq A_S$.
4. Add j to S'' if some $x \in A_S$ is present solely in A_j.
5. Return $S' \cup S''$.

A quick calculation shows that the running time of the whole decoding algorithm is $O(nk\log(n/k))$. This is rather expensive as it is exponential in $r = O(k\log(n/k))$ for $k = \operatorname{poly}\log n$. The encoding algorithm suffers from the same behaviour, as basically it determines S' similarly and encodes it recursively.

Our explicit construction for $k = 2$, however, has polynomial-time encoding and decoding algorithms. We will use the following algorithms as subroutines.

Claim 5.1. *Fix integers $a \geq b \geq 0$. Let φ_b^a be the lexicographic isomorphism from $\binom{[a]}{b}$ to $[\binom{a}{b}]$ and let $\psi_b^a : [\binom{a}{b}] \to \binom{[a]}{b}$ be its inverse. There exist* $\mathrm{poly}(a,b)$*-time algorithms computing $\varphi_b^a(S)$ given S and $\psi_b^a(m)$ given m.*

Proposition 5.2. *The $(n,2)$-monotone encoding presented in Section 4 and its inverse can be computed in* $\mathrm{poly}\log n$*-time (per input).*

5.2 Open Problems

Exact Constructions. In spite of Proposition 3.4, we believe that usually $r(n,2) = \rho(n,2)$. For a fixed n, the following method can be used to check if $r(n,2) = \rho(n,2)$. First, we assign $f(\emptyset)$ to $\emptyset$ and all singletons $\{f(\{i\})\}_{i=1}^n$ to small subsets of $2^{[\rho(n,2)]}$. Next, we build the bipartite constraints graph:

- On one side $U = \binom{[n]}{2}$ we have all pairs,
- On the other side $V \subset 2^{[\rho(n,2)]}$ we have all unassigned targets;
- An edge connects $\{i,j\} \in U$ and $A \in V$ iff $f(\{i\}) \cup f(\{j\}) \subseteq A$.

A matching that saturates U in this graph translates into an $\mathrm{ME}(n,2)$.[10]

Using Hopcroft-Karp's maximum-cardinality bipartite matching algorithm, we verified that a saturating matching exists for $23 \leq n \leq 250$. Especially interesting is $n = 90$ since $1 + 90 + \binom{90}{2} = 2^{12}$, rendering the monotone encoding surjective as well. This suggests the following conjecture.

Conjecture 1. *For all $n \geq 23$, $r(n,2) = \rho(n,2)$.*

Maybe the following stronger version is true as well.

Conjecture 2. *For every fixed $k \geq 2$, $r(n,k) \neq \rho(n,k)$ for only a finite number of values of n.*

Note that in the notation above, almost always $|U| < |V|$, i.e., there is some 'extra' space. Indeed, this is a simple consequence of the ABC Conjecture, as we explain next.

Masser and Oesterlé conjectured in 1985 that for any $\epsilon > 0$ there exists a constant $K_\epsilon > 0$ such that for every triple of coprime positive integers a, b, c satisfying $a + b = c$ we have $c \leq K_\epsilon(\mathrm{rad}(abc))^{1+\epsilon}$, where $\mathrm{rad}(m)$ is defined as the product of all distinct prime divisors of m. This is known as the ABC Conjecture (see [13,15]) and has numerous number-theoretic consequences including the following one.

Claim 5.3. *For any fixed M we have $2^{\rho(n,2)} = 1 + n + \binom{n}{2} + M$ for only a finite number of values of n, under the assumption that the ABC Conjecture holds.*

Corollary 5.4. *For any fixed M we have $2^{\rho(n,2)} \geq 1 + n + \binom{n}{2} + M$, i.e., $|V| \geq |U| + M$ for all but a finite number of values of n, under the same assumption.*

[10] It is possible that $r(n,2) = \rho(n,2)$ and still the graph does not contain a matching saturating U, as the values of $\{f(\{i\})\}_{i=1}^n$ we have chosen are not necessarily those leading to an optimal encoding.

In other words, the matching is almost never required to be nearly perfect. It seems likely that the assertion of the last two claims can be proved without relying on any unproven conjectures, using the theory of imaginary quadratic fields, but as this is not very essential for our purpose in this paper, we include only the conditional simple proof above.

Explicit Constructions, General Case. Although the ME construction of Theorem 3 is explicit, it relies on using FUT families of various sizes as building blocks, for which we only presented a probabilistic construction. A bipartite graph $G = (U, V, E)$ in which the degree of every vertex $u \in U$ is s is called *a (k, δ)-expander* if any $U' \subset U$ of size at most k has at least $\delta s|U'|$ neighbors in V. $(k, 1-\epsilon)$-expanders for some small $\epsilon > 0$, called *lossless expanders*, may assist us in building FUT families as any $(k, 1-\epsilon)$-expander yields a $(k, 1-2\epsilon)$-FUT family; However, the best known explicit constructions of these (see [4,10]) do not suffice for the recursive chaining procedure of Theorem 3.

References

1. Alon, N.: Explicit construction of exponential sized families of k-independent sets. Discrete Mathematics 58(2), 191–193 (1986)
2. Alon, N., Asodi, V.: Tracing a single user. European Journal of Combinatorics 27(8), 1227–1234 (2006)
3. Alon, N., Asodi, V.: Tracing many users with almost no rate penalty. IEEE Transactions on Information Theory 53(1), 437–439 (2007)
4. Capalbo, M., Reingold, O., Vadhan, S., Wigderson, A.: Randomness conductors and constant-degree lossless expanders. In: Proceedings of the 34th Annual ACM STOC, pp. 659–668 (2002)
5. Coppersmith, D., Shearer, J.: New bounds for union-free families of sets. Electronic Journal of Combinatorics 5(1), 39 (1998)
6. Csűrös, M., Ruszinkó, M.: Single user tracing and disjointly superimposed codes. IEEE Transactions on Information Theory 51(4), 1606–1611 (2005)
7. Dyachkov, A.G., Rykov, V.V.: Bounds on the length of disjunctive codes. Problemy Peredachi Informatsii 18(3), 158–166 (1982)
8. Erdős, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of r others. Israel Journal of Mathematics 51(1-2), 79–89 (1985)
9. Füredi, Z.: A note on r-cover-free families. Journal of Combinatorial Theory Series A 73(1), 172–173 (1996)
10. Guruswami, V., Umans, C., Vadhan, S.: Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. In: Proceedings of the 22nd IEEE CCC, pp. 96–108 (2007)
11. Komlós, J., Greenberg, A.: An asymptotically fast nonadaptive algorithm for conflict resolution in multiple-access channels. IEEE Transactions on Information Theory 31(2), 302–306 (1985)
12. Laczay, B., Ruszinkó, M.: Multiple user tracing codes. In: Proceedings of IEEE ISIT 2006, pp. 1900–1904 (2006)
13. Masser, D.W.: Note on a conjecture of Szpiro. Astérisque 183, 19–23 (1990)

14. Moran, T., Naor, M., Segev, G.: Deterministic History-Independent Strategies for Storing Information on Write-Once Memories. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 303–315. Springer, Heidelberg (2007)
15. Oesterlé, J.: Nouvelles approches du "théorème" de Fermat. Astérisque 161/162, 165–186 (1988)
16. Ruszinkó, M.: On the upper bound of the size of the r-cover-free families. Journal of Combinatorial Theory Series A 66(2), 302–310 (1994)

Polynomial-Time Construction of Linear Network Coding

Kazuo Iwama[1,⋆], Harumichi Nishimura[2,⋆⋆], Mike Paterson[3,⋆⋆⋆], Rudy Raymond[4], and Shigeru Yamashita[5,†]

[1] School of Informatics, Kyoto University, Japan
iwama@kuis.kyoto-u.ac.jp
[2] School of Science, Osaka Prefecture University, Japan
hnishimura@mi.s.osakafu-u.ac.jp
[3] Department of Computer Science and DIMAP, University of Warwick, UK
msp@dcs.warwick.ac.uk
[4] Tokyo Research Laboratory, IBM Japan, Japan
raymond@jp.ibm.com
[5] Graduate School of Information Science, Nara Inst. of Science & Technology, Japan
ger@is.naist.jp

Abstract. Constructing k independent sessions between k source-sink pairs with the help of a linear operation at each vertex is one of the most standard problems in network coding. For an unbounded k, this is known to be NP-hard. Very recently, a polynomial-time algorithm was given for $k = 2$ [Wang and Shroff, ISIT 2007], but was open for a general (constant) k. This paper gives a polynomial-time algorithm for this problem under the assumption that the size of the finite field for the linear operations is bounded by a fixed constant.

1 Introduction

The max-flow min-cut theorem is a fundamental law for communication networks. So, it was remarkable when Ahlswede, Cai, Li and Yeung showed that this law can be bypassed by *network coding* [2]. They gave a small and nice example called the Butterfly network. As shown in Figure 1, the Butterfly network has two source-sink pairs (s_1, t_1) and (s_2, t_2), and it is easily seen that if we remove a single link, i.e., the one from s_0 to t_0, then there is no path from s_1 to t_1 or from s_2 to t_2 any longer. Therefore, we cannot achieve two disjoint paths for the two source-sink pairs in order to send two bits, x from s_1 to t_1 and y

⋆ Supported in part by Scientific Research Grant, Ministry of Japan, 16092101 and 19200001.
⋆⋆ Supported in part by Scientific Research Grant, Ministry of Japan, 19700011.
⋆⋆⋆ Supported in part by DIMAP (Centre for Discrete Mathematics and its Applications), EPSRC, UK, and by Scientific Research Grant, Ministry of Japan, 16092101.
† Supported in part by Scientific Research Grant, Ministry of Japan, 16092218 and 19700010.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 271–282, 2008.

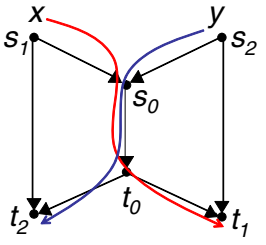


Fig. 1. Butterfly network

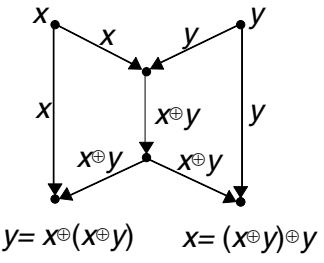


Fig. 2. Coding scheme

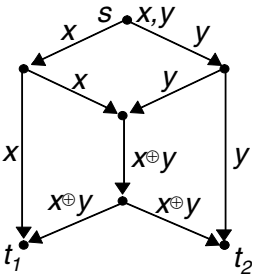


Fig. 3. Multicast example

from s_2 to t_2, simultaneously. However, if we allow "coding" at each vertex, then we can consider, for instance, the protocol in Figure 2 and two bits x and y can now be transmitted simultaneously! After [2], network coding became instantly popular and the large literature has investigated the possibility and applications of network coding (see the network coding home page [11]).

Network coding has several different models, but the two major ones are (i) the *k-source-sink pair model* (also called the multiple-source unicast model) and (ii) the *multicast model*. A typical example of (i) is the butterfly network, where $k = 2$. The latter model has only one source (and many sinks) but usually the source has multiple inputs. For example, the model in Figure 3 has one source s with two inputs x, y, and two sinks t_1 and t_2 which both require x and y in a single round. (An obvious necessary condition is that there must be a sufficient number of links from the source vertex, two in the case of Figure 3, and similarly for each sink.) There is a large literature for this model: for example, (A) Li, Yeung and Cai [18] showed that a network coding exists if and only if a *linear* network coding exists (i.e., the operation at every vertex is a linear combination of inputs on a finite field alphabet), (B) the maximum size of an alphabet in the linear coding can be bounded by a relatively small number [9,10], and (C) such a network coding (if any) can be found in polynomial time [10,7,13].

The k-source-sink pair model, requiring independent sessions between sources and sinks as in the Butterfly, is much more difficult. In fact, this model is essentially the same as the most general model where (not necessarily pairwise) sources and sinks may have multiple inputs and multiple outputs, respectively [4]. Compared to the multicast model, (A′) linear coding is not sufficient, i.e., there exist graphs that do not admit linear network coding but admit vector-linear network coding (vector-linear operations instead of linear operations are allowed at each vertex) [16,19], (B′) the maximum alphabet size is not known even if we only consider linear coding, and (C′) if k is not bounded, then whether a given graph has a linear network coding is NP-complete for any fixed alphabet size [16].

Although (A′) shows a limit of linear coding, linear network coding is still popular because of its simplicity and in fact admits several nice features (e.g., exponentially larger bandwidth) compared to conventional routing (e.g., [1,8]). Therefore, considering (C′) also, our natural question for the k-pair model is

whether there is a polynomial-time algorithm for deciding the possibility of linear network coding for a small constant k, which has been one of the most important open questions in the field.

Very recently there was some important progress toward this goal; Wang and Shroff [22,23] showed that the problem is solvable in polynomial time if $k = 2$. The proof exploits nontrivial graph theoretic properties, which furthermore implies that (i) linear coding is enough and (ii) two is enough for the size of its alphabet. Thus the question was answered almost completely for $k = 2$, but at the same time, the above two facts suggest that the case of $k = 2$ is somewhat special and it is hard to extend their approach to a general (constant) k.

1.1 Our Contribution

In this paper we present an algorithm that decides, given a directed acyclic graph and k source-sink pairs, whether or not a linear network coding exists. It runs in time $O(n^{g(a,k)})$ where n is the number of vertices in the graph and $g(a, k)$ is a function depending only on k and a, the maximum size of the field $\mathbb{F}$ we can use for linear coding. Thus, if both k and a are constants, then this is a polynomial-time algorithm. Unfortunately it is not known if such a (i.e., depending only on k and not on n) is enough for a linear network coding.

Here is the basic idea: we first show, using a similar idea to that in [14], that the given graph can be transformed, with a sacrifice of a quadratic increase in its size, into another graph whose maximum indegree is two. Then the number of different linear operations at each vertex is at most $|\mathbb{F}|^2$, which is constant since we assumed that $|\mathbb{F}|$ is bounded by a constant. Thus far, we have a trivial exponential-time algorithm by considering $(|\mathbb{F}|^2)^{n^2}$ combinations of linear operations for all vertices. To decrease this complexity, we first prove the key lemma stating that the number of "real" coding vertices can be bounded by a constant (only dependent on k). "Real" means that the output state of the gate (vertex) depends on both input states. In other words, in all the other gates the output state depends on only one of the two input states.

Roughly speaking, what remains to be done in those gates is to select one of the two inputs and connect it to the output. It turns out that this is equivalent to finding vertex-disjoint paths between h (again bounded by a constant) pairs of vertices, which is known as the Disjoint Paths Problem. For directed acyclic graphs, the general problem for unrestricted h is still NP-hard, but fortunately polynomial-time algorithms for constant h were found by Fortune et al. [5].

1.2 Related Work

As mentioned in [15], there are many variables to consider in network coding. Here we list some of them which are closely related to this paper.

Alphabet size. The complexity of the problem with respect to the size of the alphabet (the size of the finite field in the case of linear network coding)

is subtle, since if the size is larger we can use more powerful operations but at the same time we need to transmit more information in each round. However, from a practical viewpoint, it is obviously better to use smaller alphabets. In the multicast model, an upper bound of the alphabet size for linear network coding, $O(h)$ where h is the number of sinks, was shown by Ho et al. [9] based on the algebraic framework in [12]. This upper bound (and the algebraic argument in [12,9]) was used in a polynomial-time algorithm by Harvey et al. [7]. Another (in fact, the first) polynomial-time algorithm, also with alphabet size bounded by $O(h)$, was given by Jaggi et al. [10]. On the other hand, the lower bound on the alphabet size of $\Omega(\sqrt{h})$ is shown in [16,21], and deciding the smallest alphabet size is NP-hard [16].

Much less is known for the k-source-sink pair model. There is a network coding that has a doubly-exponential lower bound (in the graph size n) on alphabet size [17]. However, this lower bound example does not admit linear network coding (but admits a vector-linear coding), and its k increases with n. So, the maximum alphabet size, especially whether it depends only on k, is still open for both linear and general network codings. Interestingly, there is a graph which admits network coding for the (unique) finite field of size 4 but not for any finite field whose size is not a square number (for example, 5, 7, 8) [17].

Coding operations. Linear coding over a finite-field alphabet is one of the most useful and popular settings. As mentioned before, linear coding is enough for any graph in the multicast model and for many "natural" graphs in other models. However, there exist counter examples: in [19] it is shown that some graph class from [16] and a simple graph by Koetter do not have a linear network coding even if their alphabet sizes are arbitrarily large, but do have vector-linear network coding over an alphabet of size four (actually $\mathbb{F}_2^2$). From simply transforming Koetter's graph to a graph in the k-pair model by the method in [4], it also turns out that this situation, i.e., insufficiency of linear coding, happens as early as $k = 8$. (Recall that linear coding is sufficient when k=2.) Dougherty et al. [3] found an example that does not have a vector linear network coding over any alphabet, but has a network coding over a size-four alphabet if we allow some non-linear operations.

The number of encoding vertices. Recall that this issue is also very important in this paper. Several studies on reducing the number of encoding vertices do exist, but all of them focus on the multicast model. For the multicast model with two inputs (as in Figure 3), Tavory et al. [21] showed that the number of encoding vertices to construct a network coding is independent of the size of the graph, and Fragouli and Soljanin [6] independently proved that it is bounded by the number of the sinks. For the case of three or more inputs, Langberg et al. [14,13] gave an efficient construction of network coding in which the number of encoding vertices is independent of the size of the graph (only dependent on the numbers of inputs and sinks). This might seem to be closely related to our present result but the property specific to the multicast model plays a key role in their proof.

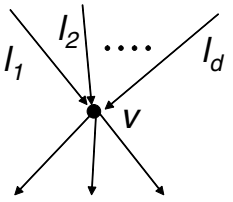


Fig. 4. Vertex v

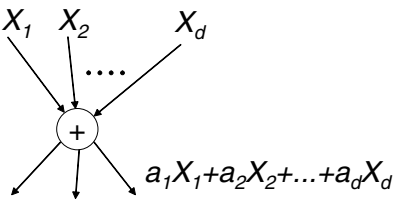


Fig. 5. Linear gate replacing v

2 Network Coding

There are only a finite number of different finite fields if their size is fixed. Therefore we do not lose generality if we fix a finite field itself. For a finite field $\mathbb{F}$, *Linear Network Coding over* $\mathbb{F}$ (LNC($\mathbb{F}$)) is the following decision problem: we are given a directed acyclic graph (DAG) $G = (V, E)$ ($|V| = n$, $|E| = m$) and a *requirement* R. In this paper we do not discuss specific values of exponents for polynomials representing the complexity, and thus we simply use n as the size of the graph though it is $n+m$ precisely. For each vertex $v \in V$, its incoming edges have labels $l_1, l_2, \ldots, l_d$ where d is the indegree of v. R is a set of *source-sink pairs*, given as $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, where, for all i, j, (i) $s_i, t_i \in V$, $s_i \neq s_j$ (for $i \neq j$) and $t_i \neq t_j$ (for $i \neq j$), and (ii) the indegree of s_i and the outdegree of t_i are both zero. Consider a mapping σ, called a *gate assignment mapping*, such that for a vertex v of indegree d, $\sigma(v)$ is given as $\sigma(v) = (a_1, \ldots, a_d)$ where each a_i is in $\mathbb{F}$.

Then σ maps the graph G to a linear circuit in the following sense: suppose that $\sigma(v) = (a_1, \ldots, a_d)$ for a vertex v. Then as shown in Figures 4 and 5, we replace the vertex v by the linear gate computing $a_1X_1 + a_2X_2 + \cdots + a_dX_d$, where X_i is the state (in $\mathbb{F}$) of the incoming edge with label l_i. So, we can consider that a_i is assigned to the edge labeled by l_i. In this paper, we assume that all outgoing edges from a single vertex have the same output state. However, this assumption can be removed by increasing (polynomially) the number of vertices by a method similar to Lemma 1 given later. Thus, this restriction does not lose generality. Source vertices receive global inputs $(x_1, x_2, \ldots, x_k) \in \mathbb{F}^k$ of the circuit and sink vertices hold global outputs $(y_1, \ldots, y_k) \in \mathbb{F}^k$ of the circuit. Note that y_i is a function, denoted by $f_i(x_1, \ldots, x_k)$, depending on the k global input values in general. LNC($\mathbb{F}$) asks whether some gate assignment mapping σ can yield $f_i(x_1, \ldots, x_k) = x_i$ for all $1 \leq i \leq k$. If such σ exists, we say that σ *realizes* the requirement R and also that a *network coding exists* for the graph G and the requirement R. To summarize:

Linear Network Coding over $\mathbb{F}$ (LNC($\mathbb{F}$))
Instance: A pair (G, R) where G is a DAG and R is a set of k source-sink pairs.
Question: Is there a network coding, i.e., is there a gate assignment mapping (over $\mathbb{F}$) which realizes (G, R)?

Assumption: We assume that the number k of source-sink pairs is a constant.

Now we introduce a restriction to the graph which does not lose generality: a DAG G is said to be 2/1-restricted if the (indegree, outdegree) pair for each vertex is only either $(2,1)$ or $(1,2)$, except for sources and sinks.

Lemma 1. *([14]) If LNC($\mathbb{F}$) is solvable for 2/1-restricted graphs in polynomial time, then it is solvable for general graphs in polynomial time.*

Thus, without loss of generality we can assume that our graph contains k $(0,1)$ vertices (sources), k $(1,0)$ vertices (sinks) and all the others are $(2,1)$ or $(1,2)$ vertices. For such a G, a gate assignment mapping is given as $\sigma(v) = (a_1, a_2)$ (where $a_1, a_2 \in \mathbb{F}$) for $(2,1)$ vertices v and $\sigma(v) = a$ (where $a \in \mathbb{F}$) for $(1,2)$ and $(1,0)$ vertices v. Hence the number of different operations at each vertex is at most $|\mathbb{F}|^2$ and so the number of all different mappings is $O(|\mathbb{F}|^{2n})$, which is a trivial upper bound for the complexity of the problem (it is straightforward to check for each assignment whether it actually realizes R).

In the following two sections, we shall explain how to reduce the time complexity to polynomial. In the next section, we first prove that we need only a constant number, C, of "real" $(2,1)$ vertices. If $\sigma(v) = (a_1, a_2)$, "real" means neither a_1 nor a_2 is zero, and we call such a vertex a *coding vertex*. Note that if (at least) one of them is zero, then the corresponding incoming edge can be "cut" and the vertex effectively becomes a $(1,1)$ vertex. Thus the number of different cases to determine the positions of those coding vertices is at most $\binom{n}{C}$, which is bounded by a polynomial. Since the number of different coding operations at each vertex is also constant (at most $|\mathbb{F}|^2$), we can check all the possibilities just by increasing the complexity by a constant factor. For each set of fixed positions and coding operations for those coding vertices, all we have to do further is to decide how to change the other $(2,1)$ vertices to (effective) $(1,1)$ vertices. An exhaustive search would still need exponential time for this purpose, but as shown in Section 4, we can use the disjoint-paths problem for which polynomial-time algorithms are known.

3 Main Lemma

We prove the key lemma which bounds the number of coding vertices.

Lemma 2. *If R is realized by some gate assignment mapping, then it is realized by another mapping for which the number of coding vertices is at most $|\mathbb{F}|^{3k}$.*

Proof. We need several new definitions. Consider the linear circuit T that is induced from the requirement $R = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ and a gate assignment mapping σ realizing R. Let $x_1, \ldots, x_k \in \mathbb{F}$ be the global inputs (at the sources) of T. Then the input and output values of each gate g (which corresponds to a coding vertex) can be written in the form $\sum_{i=1}^{k} a_i x_i$ where $a_i \in \mathbb{F}$. First, we define the following change of functionality of a gate. Let g be a gate which

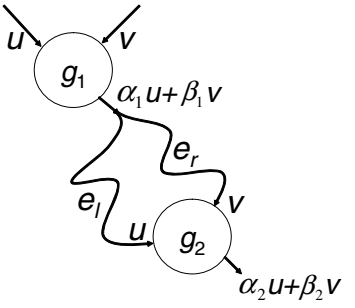


Fig. 6. Gates g_1 and g_2 of the same type

produces output $\alpha X + \beta Y$ for left input X and right input Y. The *left-side γ-change* of g means that we change g to the gate which produces output $(\alpha + \gamma)X + \beta Y$. In particular, if $\gamma = -\alpha$ we call the change the *left-side cut* of g. Similarly we define the *right-side γ-change* of g and the *right-side cut* of g. Note that by the left-side γ-change, the output value O_g of g changes to $O_g + \gamma X$ (that is, changes by $+\gamma X$). Secondly, we define the *effect* of a gate g on the i-th global output y_i (at sink t_i) of T. This is defined as the value of y_i (in $\mathbb{F}$) when all the inputs of T are set to 0 while the output value of g is forced to 1 (regardless of the input of g), or equivalently, is defined as the sum of products of all the σ values assigned to the edges on the paths from g to the i-th global output. Similarly we can define the effect of g on the left input or the right input of another gate. The following lemma follows easily from the linearity of gates.

Lemma 3. *Let $e \in \mathbb{F}$ be the effect of a gate g on the i-th output y_i of T. If the output value of g is changed (due to a change of its functionality) by $+\mu$, then the value of y_i is changed by $+e\mu$. A similar statement holds for the left and the right inputs of a gate, instead of y_i.*

Finally, we define the notion of a type of each gate. Let $a = (a_1, \ldots, a_k)$, $b = (b_1, \ldots, b_k)$ and $e = (e_1, \ldots, e_k)$, where all a_i, b_i, e_i are in $\mathbb{F}$. A gate g is of *type* (a, b, e) if the left-side and right-side inputs of g are $\sum_{i=1}^{k} a_i x_i$ and $\sum_{i=1}^{k} b_i x_i$, respectively, and the effect of g on the j-th output of T is e_j. The following lemma is straightforward from Lemma 3.

Lemma 4. *Fix arbitrary global input values and let g_1 and g_2 be two gates of the same type where g_1 is not a descendant of g_2 (see Figure 6). Suppose that the output value of g_1 is changed (again due to a change of its functionality) by $-\mu$ and suppose that then the output value of g_2 is similarly changed by $+\mu$. Then the outputs of T do not change their values.*

Now suppose that the number of gates in T is larger than $|\mathbb{F}|^{3k}$. Then there are two gates g_1 and g_2 whose types are the same, (a, b, e) say. Let $g_1(X, Y) = \alpha_1 X + \beta_1 Y$ and $g_2(X, Y) = \alpha_2 X + \beta_2 Y$ (without loss of generality, we can assume that none of $\alpha_1, \beta_1, \alpha_2, \beta_2$ are 0). Again assume that g_1 is not a descendant of g_2

(see Figure 6). Then we have the following lemma (Lemma 5), which completes the proof of Lemma 2.

Lemma 5. *Let g_1 and g_2 be as above. Then there must be a left-side cut (and/or a right-side cut) of g_1 and a left-side α_2'-change (and/or a right-side β_2'-change) of g_2 such that the functionality of the circuit does not change.*

Proof. Let e_l and e_r be the effects of g_1 on the left and right inputs of g_2, respectively (see Figure 6). Then, we consider the following three cases: (i) $\alpha_1 e_l \neq 1$ (ii) $\beta_1 e_r \neq 1$ (iii) $\alpha_1 e_l = \beta_1 e_r = 1$. We only analyze cases (i) and (iii) since the analysis of (ii) is similar to (i). Fix arbitrary global input values and let u and v be the values of (both) g_i's left and right inputs, respectively.

Case (i): $\alpha_1 e_l \neq 1$. We show that after the left-side cut of g_1 the output value of g_1 is changed by $-\alpha_1 u$ and there is a choice of α_2' such that the output value of g_2 is changed by $+\alpha_1 u$ after the left-side α_2'-change of g_2. Then, the lemma follows from Lemma 4. In fact, by the left-side cut of g_1 the output value of g_1 is changed from $\alpha_1 u + \beta_1 v$ to $\beta_1 v$. Thus, the output of g_1 is changed by $-\alpha_1 u$. Then, by Lemma 3, the left and right input values of g_2 are changed by $-e_l\alpha_1 u$ and $-e_r\alpha_1 u$, respectively. This means that the left and right inputs of g_2 take the values $u - e_l\alpha_1 u$ and $v - e_r\alpha_1 u$, respectively. (Recall that g_2 is of the same type as g_1, which means that its previous input values were u and v.) By the left-side α_2'-change of g_2, the output value of g_2 then changes by

$$\alpha_2'(u - e_l\alpha_1 u) = \alpha_2'(1 - e_l\alpha_1)u. \tag{1}$$

By setting $\alpha_2' = \alpha_1(1 - e_l\alpha_1)^{-1}$, the right-hand side of Equation (1) is $+\alpha_1 u$. Thus the output value of g_2 changes by $+\alpha_1 u$.

Case (iii): $\alpha_1 e_l = \beta_1 e_r = 1$. First we carry out the left-side and the right-side cuts of g_1. Then, the output value of g_1 is changed by $-(\alpha_1 u + \beta_1 v)$. By Lemma 3 the left and right input values of g_2 are changed by

$$-e_l(\alpha_1 u + \beta_1 v) = -\frac{1}{\alpha_1}(\alpha_1 u + \beta_1 v) = -u - \frac{\beta_1}{\alpha_1} v$$

and

$$-e_r(\alpha_1 u + \beta_1 v) = -\frac{1}{\beta_1}(\alpha_1 u + \beta_1 v) = -v - \frac{\alpha_1}{\beta_1} u,$$

respectively. Thus, the left and right inputs of g_2 take values $u - u - \frac{\beta_1}{\alpha_1} v = -\frac{\beta_1}{\alpha_1} v$ and $v - v - \frac{\alpha_1}{\beta_1} u = -\frac{\alpha_1}{\beta_1} u$, respectively. Then by the left-side α_2'-change and the right-side β_2'-change of g_2, the output value of g_2 changes by

$$-\alpha_2' \frac{\beta_1}{\alpha_1} v - \beta_2' \frac{\alpha_1}{\beta_1} u.$$

By setting $\alpha_2' = -\alpha_1$ and $\beta_2' = -\beta_1$ of g_2, we can verify that the output value of g_2 changes by $+(\alpha_1 u + \beta_1 v)$. Therefore, the lemma follows from Lemma 4 in this case also.

This concludes the proof of Lemma 2. □

4 Polynomial-Time Algorithms

Recall that our graph includes only $(2,1)$ and $(1,2)$ vertices excepting sources (i.e., $(0,1)$ vertices) and sinks (i.e., $(1,0)$ vertices). Fix a gate assignment mapping σ. Then σ defines a linear circuit, T, as described in Section 2. Now we introduce a procedure which simplifies T:

Call an edge *dead* if it is assigned 0 by σ.
(i) Remove all dead edges.
(ii) If all outgoing edges of a vertex v are removed, then remove all its incoming edges, too.
(iii) Repeat (ii) until no further removal is possible.

The resulting circuit is said to be *reduced* and includes the following vertices: (i) k source vertices (all of them should remain if σ realizes the requirement R), (ii) k sink vertices (similarly as above), (iii) vertices having two remaining incoming edges, called *coding vertices*, (iv) vertices having two remaining outgoing edges, called *fork vertices*, and (v) all other vertices, called *path vertices*, such that for each of them there remains a single incoming and a single outgoing edge. We say that a mapping σ is *reduced* if it directly defines a reduced circuit (i.e., if 0-assigned edges are removed, then no further simplification is possible). Proofs for the following two lemmas are straightforward and omitted.

Lemma 6. *Suppose that there exists a gate assignment mapping σ which realizes a requirement R. Then there exists a reduced mapping σ' such that: (i) σ' realizes R; and (ii) the number of coding vertices of σ' is less than or equal to the number of coding vertices of σ.*

Lemma 7. *If σ is reduced, then the circuit defined by σ has the same numbers of coding vertices and fork vertices.*

Now we are ready to prove our main result.

Theorem 1. *LNC($\mathbb{F}$) can be solved in polynomial time.*

Proof. For a given (G,R), suppose that there is a gate assignment mapping which realizes R. Then, by Lemma 2, there is such a mapping σ which includes only a finite number C of coding vertices. This implies, by Lemma 6, that there is a reduced mapping σ' which realizes R and includes at most C coding vertices. Lemma 7 guarantees that the circuit defined by σ' includes at most C fork vertices whose two outgoing edges are assigned non-zero values by σ'. Thus to prove the existence of σ, it suffices to find a reduced mapping σ' which (i) includes at most C coding vertices and at most C fork vertices and (ii) realizes R. For (i), our basic strategy is enumeration in polynomial time of all such mappings without considering the requirement R. Checking (ii) can obviously be done in polynomial time.

For the enumeration, we first select from V a set of (at most) C positions (vertices) for C coding vertices and (at most) C positions for C fork vertices.

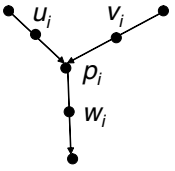


Fig. 7. Inserting u_i, v_i, w_i at a code vertex

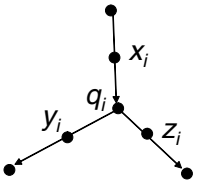


Fig. 8. Inserting x_i, y_i, z_i at a fork vertex

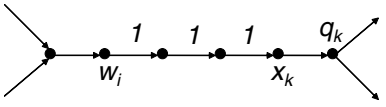


Fig. 9. Assigning 1 to all vertices on the path

Note that the total number of such selections is $O(\binom{n}{C} \cdot \binom{n}{C}) = O(n^{2C})$, and is bounded by a polynomial since C is constant. Suppose that a single selection consists of $(2,1)$ vertices $p_1, \ldots, p_C$ for the coding vertices and $(1,2)$ vertices $q_1, \ldots, q_C$ for the fork vertices. Then we make another change to the graph G as follows: insert vertices u_i, v_i, w_i into the three edges of p_i and x_i, y_i, z_i into the three edges of q_i as shown in Figures 7 and 8. After this modification, let

$$\text{OUT} = \{s_1, \ldots, s_k, w_1, \ldots, w_C, y_1, \ldots, y_C, z_1, \ldots, z_C\}$$
$$\text{IN} = \{t_1, \ldots, t_k, u_1, \ldots, u_C, v_1, \ldots, v_C, x_1, \ldots, x_C\}.$$

The key observation is that a sequence of vertices of path vertices in (v) defined by σ' constitutes a "path" in graph G which starts from some vertex in OUT and ends with some vertex in IN. Furthermore, any two such paths are obviously vertex-disjoint, i.e., these paths define a matching between OUT and IN. Conversely, suppose that we are given fixed OUT and IN (let $|\text{OUT}| = |\text{IN}| = L$) and a (complete) matching $M = \{(a_1, b_1), (a_2, b_2), \ldots, (a_L, b_L)\} \subseteq \text{OUT} \times \text{IN}$. Then this matching (together with the set of fork vertices and a concrete operation at each of the coding vertices) defines a linear circuit, and whether or not such a circuit is defined by some σ' is equivalent to whether or not there are vertex-disjoint paths connecting each a_j to b_j, without going through any p_i or q_i (where $1 \leq i \leq C$). If L is constant, then this test can be done in time polynomial in n [5]. If such paths exist, then we can construct the circuit and the corresponding gate assignment mapping σ'. It is straightforward to check whether σ' realizes the requirement R of network coding. Note that without loss of generality we can assign 1 to the vertices on the path as shown in Figure 9, since all the multiplicative constants on the path can be accumulated into the constant on edge (x_k, q_k). Algorithm 1 shows the formal description of our algorithm.

Since C is constant, the number of repetitions of Line 2 is bounded by a polynomial. For fixed CODE and FORK, the number of repetitions of Line 4

Algorithm 1. Our Algorithm for Linear Network Coding of k Source-sink Pairs.

1: Compute the maximum number C of coding vertices by Lemma 2.
2: **for each** CODE $\subseteq V$, FORK $\subseteq V$ such that $|\text{CODE}| = |\text{FORK}| \leq C$ **do**
3: Insert vertices as in Figures 7 and 8 for vertices in CODE and FORK and compute OUT and IN.
4: **for each** matching $M \subseteq \text{OUT} \times \text{IN}$ **do**
5: Check if there are disjoint paths for M in G' where G' is the graph obtained by removing all vertices (and their incoming and outgoing edges) in CODE $\cup$ FORK (but vertices in IN and OUT and all vertices in $V - \text{CODE} \cup \text{FORK}$).
6: **if** No **then**
7: Go to **end**.
8: **else**
9: **for each** gate assignment mapping σ for G' such that σ assigns (i) 1 to all the edges in the selected paths above, (ii) any $a \in \mathbb{F} - \{0\}$ to each of the edges to or from vertices in CODE$\cup$FORK, and (iii) 0 to all the other edges **do**
10: Check if σ realizes R.
11: **if** Yes **then**
12: Answer YES and halt.
13: **end if**
14: **end for**
15: **end if**
16: **end for**
17: **end for**
18: Answer NO and halt.

is bounded by a constant. The number of repetitions of Line 9 is also bounded by a constant. Line 5 can be done in polynomial time by [5], and all the other instructions can obviously be executed in polynomial time. Thus the total running time is also polynomial. The correctness of the algorithm follows from the observation given in the previous pages.

5 Concluding Remarks

Our algorithm runs in polynomial time. However, its exponent is obviously not small. Seeking more efficient algorithms is an obvious future task. Fixed parameter tractability of the problem also seems interesting, but it is known that the disjoint paths problem for DAGs is $W[1]$-hard [20]. Therefore, in order to design FPT algorithms, we have to bypass the main subroutine solving this problem, which does not seem easy.

References

1. Adler, M., Harvey, N.J., Jain, K., Kleinberg, R.D., Lehman, A.R.: On the capacity of information networks. In: Proc. 17th ACM-SIAM SODA, pp. 241–250 (2006)
2. Ahlswede, R., Cai, N., Li, S.-Y.R., Yeung, R.W.: Network information flow. IEEE Transactions on Information Theory 46, 1204–1216 (2000)

3. Dougherty, R., Freiling, C., Zeger, K.: Insufficiency of linear coding in network information flow. IEEE Transactions on Information Theory 51, 2745–2759 (2005)
4. Dougherty, R., Zeger, K.: Nonreversibility and equivalent constructions of multiple-unicast networks. IEEE Transactions on Information Theory 52, 5067–5077 (2006)
5. Fortune, S., Hopcroft, J., Wyllie, J.: The directed subgraph homeomorphism problem. Theoret. Comput. Sci. 10, 111–121 (1980)
6. Fragouli, C., Soljanin, E.: Information flow decomposition for network coding. IEEE Transactions on Information Theory 52, 829–848 (2006)
7. Harvey, N.J., Karger, D.R., Murota, K.: Deterministic network coding by matrix completion. In: Proc. 16th ACM-SIAM SODA, pp. 489–498 (2005)
8. Harvey, N.J., Kleinberg, R.D., Lehman, A.R.: Comparing network coding with multicommodity flow for the k-pairs communication problem. MIT LCS Technical Report 964 (September 2004)
9. Ho, T., Karger, D.R., Médard, M., Koetter, R.: Network coding from a network flow perspective. In: Proc. IEEE International Symposium on Information Theory (2003)
10. Jaggi, S., Sanders, P., Chou, P.A., Effros, M., Egner, S., Jain, K., Tolhuizen, L.M.G.M.: Polynomial time algorithms for multicast network code construction. IEEE Transactions on Information Theory 51, 1973–1982 (2005)
11. Koetter, R.: Network coding home page, http://tesla.csl.uiuc.edu/~koetter/NWC/
12. Koetter, R., Médard, M.: Beyond routing: An algebraic approach to network coding. In: Proc. 21st Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 122–130 (2002)
13. Langberg, M., Sprintson, A., Bruck, J.: Network coding: A computational perspective. In: Proc. 40th Conference on Information Sciences and Systems (2006)
14. Langberg, M., Sprintson, A., Bruck, J.: The encoding complexity of network coding. IEEE Transactions on Information Theory 52, 2386–2397 (2006)
15. Lehman, A.R.: Network Coding. PhD thesis. MIT, Cambridge (2005)
16. Lehman, A.R., Lehman, E.: Complexity classification of network information flow problems. In: Proc. 15th ACM-SIAM SODA, pp. 142–150 (2004)
17. Lehman, A.R., Lehman, E.: Network coding: Does the model need tuning? In: Proc. 16th ACM-SIAM SODA, pp. 499–504 (2005)
18. Li, S.-Y.R., Yeung, R.W., Cai, N.: Linear network coding. IEEE Transactions on Information Theory 49, 371–381 (2003)
19. Médard, M., Effros, M., Ho, T., Karger, D.: On coding for non-multicast networks. In: Proc. 41st Annual Allerton Conference on Communication, Control and Computing (2003)
20. Slivkins, A.: Parameterized tractability of edge-disjoint paths on directed acyclic graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 482–493. Springer, Heidelberg (2003)
21. Tavory, A., Feder, M., Ron, D.: Bounds on linear codes for network multicast. ECCC Technical Report 33 (2003)
22. Wang, C.-C., Shroff, N.B.: Beyond the butterfly – A graph-theoretic characterization of the feasibility of network coding with two simple unicast sessions. In: Proc. IEEE International Symposium on Information Theory (2007)
23. Wang, C.-C., Shroff, N.B.: Intersession network coding for two simple multicast sessions. In: Proc. 45th Annual Allerton Conference on Communication, Control and Computing (2007)

Complexity of Decoding Positive-Rate Reed-Solomon Codes

Qi Cheng[1] and Daqing Wan[2]

[1] School of Computer Science
The University of Oklahoma
Norman, OK73019
qcheng@cs.ou.edu
[2] Department of Mathematics
University of California
Irvine, CA 92697-3875
dwan@math.uci.edu

Abstract. The complexity of maximum likelihood decoding of the Reed-Solomon codes $[q-1,k]_q$ is a well known open problem. The only known result [4] in this direction states that it is at least as hard as the discrete logarithm in some cases where the information rate unfortunately goes to zero. In this paper, we remove the rate restriction and prove that the same complexity result holds for any positive information rate. In particular, this resolves an open problem left in [4], and rules out the possibility of a polynomial time algorithm for maximum likelihood decoding problem of Reed-Solomon codes of any rate under a well known cryptographical hardness assumption. As a side result, we give an explicit construction of Hamming balls of radius bounded away from the minimum distance, which contain exponentially many codewords for Reed-Solomon code of any positive rate less than one. The previous constructions in [2] [7] only apply to Reed-Solomon codes of diminishing rates. We also give an explicit construction of Hamming balls of relative radius less than 1 which contain subexponentially many codewords for Reed-Solomon code of rate approaching one.

1 Introduction

Let $\mathbf{F}_q$ be a finite field of q elements and of characteristic p. A linear error-correcting $[n,k]_q$ code is defined to be a linear subspace of dimension k in $\mathbf{F}_q^n$. Let $D = \{x_1, \cdots, x_n\} \subseteq \mathbf{F}_q$ be a subset of cardinality $|D| = n > 0$. For $1 \le k \le n$, let f run over all polynomials in $\mathbf{F}_q[x]$ of degree at most $k-1$, the vectors of the form

$$(f(x_1), \cdots, f(x_n)) \in \mathbf{F}_q^n$$

constitute a linear error-correcting $[n,k]_q$ code. If $D = \mathbf{F}_q^*$, it is famously known as the Reed-Solomon code. If $D = \mathbf{F}_q$, it is known as the extended Reed-Solomon code. We denote them by $RS_q[q-1,k]$ and $RS_q[q,k]$ respectively. We simply call it a generalized Reed-Solomon code if D is an arbitrary subset of $\mathbf{F}_q$.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 283–293, 2008.

Remark 1. In some code theory literature, $RS_q[q-1,k]$ is called primitive Reed-Solomon code, and a generalized Reed-Solomon code $[n,k]_q$ is defined to be

$$\{(y_1 f(x_1), \cdots, y_n f(x_n)) | f \in \mathbf{F}_q[x], deg(f) < k\},$$

where $y_1, y_2, \cdots, y_n$ are nonzero elements in $\mathbf{F}_q$.

The minimal distance of a generalized Reed-Solomon $[n,k]_q$ code is $n-k+1$ because a non-zero polynomial of degree at most $k-1$ has at most $k-1$ zeroes. The ultimate decoding problem for an error-correcting $[n,k]_q$ code is the maximum likelihood decoding: given a received word $u \in \mathbf{F}_q^n$, find a codeword v such that the Hamming distance $d(u,v)$ is minimal. When the number of errors is reasonably small, say, smaller than $n - \sqrt{nk}$, then the list decoding algorithms of Guruswami-Sudan [8] gives a polynomial time algorithm to find all the codewords for the generalized Reed-Solomon $[n,k]_q$ code.

When the number of errors increases beyond $n-\sqrt{nk}$, it is not known whether there exists a polynomial time decoding algorithm. The maximum likelihood decoding of a generalized Reed-Solomon $[n,k]_q$ code is known to be **NP**-complete [6]. The difficulty is caused by the combinatorial complication of the subset D with no structures. In fact, there is a straightforward way to reduce the subset sum problem in D to the deep hole problem of a generalized Reed-Solomon code, which can then be reduced to the maximum likelihood decoding problem [3]. Note that the subset sum problem for $D \subseteq \mathbf{F}_q$ is hard only if $|D|$ is much smaller than q.

In practical applications, one rarely uses the case of arbitrary subset D. The most widely used case is when $D = \mathbf{F}_q^*$ with rich algebraic structures. This case is essentially equivalent to the case $D = \mathbf{F}_q$. For simplicity, we focus on the extended Reed-Solomon code $RS_q[q,k]$ in this paper, all our results can be applied to the Reed-Solomon code $RS_q[q-1,k]$ with little modification. The maximum likelihood decoding problem of $RS_q[q,k]$ is considered to be hard, but the attempts to prove its **NP**-completeness have failed so far. The methods in [6] [3] can not be specialized to $RS_q[q,k]$ because we have lost the freedom to select D. The only known complexity result [4] in this direction says that the decoding of $RS_q[q,k]$ is at least as hard as the discrete logarithm in $\mathbf{F}_{q^h}^*$ for h satisfying

$$h \le \sqrt{q} - k, h \le q^{\frac{1}{2+\epsilon}} + 1 \quad \text{and} \quad h \le \frac{k - \frac{4}{\epsilon} - 2}{\frac{4}{\epsilon} + 1}$$

for any $\epsilon > 0$. The main weakness of this result is that $\sqrt{q}$ has to be greater than k, which implies that the information rate k/q goes to zero. But in the real world, we tend to use the Reed-Solomon codes of high rates.

1.1 Our Results

Our main result of this paper is to remove this restriction. Precisely, we show that

Theorem 1. *For any $c \in [0,1]$, there exists an infinite explicit family of Reed-Solomon codes*

$$\{RS_{q_1}[q_1,k_1], RS_{q_2}[q_2,k_2], \cdots, RS_{q_i}[q_i,k_i], \cdots\}$$

with $q_i = \Theta(i^2 \log^2 i)$ and $k_i = (c+o(1))q_i$ such that if there is a polynomial time randomized algorithm solving the maximum likelihood decoding problem for the above family of codes, then there is a polynomial time randomized algorithm solving the discrete logarithm problem over all the fields in $\{\mathbf{F}_{q_1^{h_1}}, \mathbf{F}_{q_2^{h_2}}, \cdots, \mathbf{F}_{q_i^{h_i}}, \cdots\}$, where h_i is any integer less than $q_i^{1/4+o(1)}$.

The discrete logarithm problem over finite fields is well studied in computational number theory. It is not believed to have a polynomial time algorithm. Many cryptographical protocols base their security on this assumption. The fastest general purpose algorithm [1] solves the discrete logarithm problem over finite field $\mathbf{F}^*_{q^h}$ in conjectured time

$$exp(O((\log q^h)^{1/3}(\log\log q^h)^{2/3})).$$

Thus, in the above theorem, it is best to take h_i as large as possible (close to $q_i^{1/4+o(1)}$) in order for the discrete logarithm to be hard. If $h = q^{1/4+o(1)}$, this complexity is subexponential on q. The above theorem rules out a polynomial time algorithm for the maximum likelihood decoding problem of Reed-Solomon code of any rate under a cryptographical hardness assumption.

By a direct counting argument, for any positive integer $r < q-k$, there exists a Hamming ball of radius r containing at least $\binom{q}{r}/q^{q-r-k}$ many codewords in Reed-Solomon code $RS_q[q,k]$. Thus, if $k = \lfloor cq \rfloor$ for a constant $0 < c < 1$, we set $r = \lfloor q - k - q^{1/4} \rfloor$ and the number of code words in the Hamming ball will be exponential in q. However, finding such a Hamming ball deterministically is a hard problem. There is some work done on this problem [7, 2], but all the results are for codes of diminishing rates. Our contribution to this problem is to remove the rate restriction.

Theorem 2. *For any $c \in (0,1)$, there exists a deterministic algorithm that given a positive integer i, outputs a prime power q, a positive integer k and a vector $v \in \mathbf{F}_q^q$ such that*

- *$q = \Theta(i^2 \log^2 i)$ and $k = (c+o(1))q$, and*
- *the Hamming ball centered at v and of radius $q - k - q^{1/4+o(1)}$ contains $exp(\Omega(q))$ many codewords in $RS_q[q,k]$, and*
- *the algorithm runs in time $i^{O(1)}$.*

Our construction allows the information rate to be positive. However, the ratio between the Hamming ball radius $q-k-q^{1/4+o(1)}$ and the minimum distance $q-k+1$, which is known as the relative radius of the Hamming ball, is approaching 1, as is in [7, 2]. The following result shows that we can decrease the relative radius to a constant less than 1 if we work with codes with information rates going to one.

Theorem 3. *For any real number $\rho \in (2/3, 1)$, there is a deterministic algorithm that, given a positive integer i, outputs a prime power $q = i^{O(1)}$, a positive integer $k = q - o(\sqrt{q})$ and a vector $v \in \mathbf{F}_q^q$ such that the Hamming ball centered at v and of radius $[\rho(q-k+1)]$ contains at least q^i many codewords in $RS_q[q,k]$. The algorithm has time complexity $i^{O(1)}$. Note that the information rate is $1 - o(1)$.*

It would be interesting for future research to extend the result to all $\rho \in (1/2, 1)$ and to prove a similar result with the information rate positive and the relative radius less than 1.

Given a real number $\rho \in (0, 1)$, the codes where some Hamming balls of relative radius ρ contain superpolynomially many codewords are called ρ-dense. It was known in [5] how to efficiently construct such codes for any $\rho \in (1/2, 1)$, but finding the center of such a Hamming ball in deterministic polynomial time was left open. In this paper, we solve this problem if the relative radius falls in the range $(2/3, 1)$ using Reed-Solomon codes of rate approaching one. This result derandomizes an important step in the inapproximability result for minimum distance problem of a linear code in [5]. However, to completely derandomize the reduction there, one needs to find a linear map from a dense Hamming ball into a linear subspace. This is again an interesting future research direction.

1.2 Techniques

Our earlier paper [4] proved Theorem 1 for $c = 0$ (in that case we have $h_i \leq q_i^{1/2+o(1)}$). The main result of our earlier paper was to show that the maximum likelihood decoding of $RS_q[q,k]$ is at least as hard as the discrete logarithm over $\mathbf{F}_{q^h}$ if every element in $\mathbf{F}_{q^h}$ can be represented as products of $k+h$ distinct elements from $\alpha + \mathbf{F}_q$ where α satisfies $\mathbf{F}_q[\alpha] = \mathbf{F}_{q^h}$. The number of representations corresponds to the number of codewords in certain Hamming ball of radius $q-k-h$.

In this paper, we shall be concentrating on $0 < c \leq 1$. We shall show that the case $c = 1$ follows from the case $c = 0$ by a dual argument. The main new idea for the case $0 < c < 1$ is to exploit the role of subfields contained in $\mathbf{F}_q$. Assume that $q = \tilde{q}^2$ and $h = q^{1/4+o(1)}$ is a positive integer. We have $\mathbf{F}_{\tilde{q}} \subseteq \mathbf{F}_q \subseteq \mathbf{F}_{q^h}$. Let α be an element in $\mathbf{F}_{q^h}$ such that $\mathbf{F}_{\tilde{q}}[\alpha] = \mathbf{F}_q[\alpha] = \mathbf{F}_{q^h}$. We observe that if every element in $\mathbf{F}_{q^h}$ can be written as a product of g_1 many distinct $\alpha + a$ with $a \in \mathbf{F}_{\tilde{q}}$, then for any nonnegative integer $g_2 \leq q - \tilde{q}$, every element in $\mathbf{F}_{q^h}$ can be written as a product of $g_1 + g_2$ many distinct $\alpha + a$ with $a \in \mathbf{F}_q$. This observation enables us to prove the main technical lemma that for any constant $0 < c < 1$, any element in $\mathbf{F}_{q^h}$ can be written as a product of $\lfloor cq \rfloor$ distinct factors in $\{\alpha + a | a \in \mathbf{F}_q\}$ for q large enough.

2 Previous Work for Rate $c = 0$

For readers' convenience, in this section, we sketch the main ideas in our earlier paper [4]. This will be the starting point of our new results in the present paper.

Let $h \geq 2$ be a positive integer. Let $h(x)$ be a monic irreducible polynomial in $\mathbf{F}_q[x]$ of degree h. Let α be a root of $h(x)$ in an extension field. Then, $\mathbf{F}_q[\alpha] = \mathbf{F}_{q^h}$ is a finite field of q^h element. We have

Theorem 4. *Let $h < g < q$ be positive integers. If every element of $\mathbf{F}_{q^h}^*$ can be written as a product of exactly g distinct linear factors of the form $\alpha + a$ with $a \in \mathbf{F}_q$, then the discrete logarithm in $\mathbf{F}_{q^h}^*$ can be efficiently reduced in random time $q^{O(1)}$ to the maximum likelihood decoding of the Reed-Solomon code $RS_q[q, g-h]$.*

Proof. In [4], the same result was stated for the weaker bounded distance decoding. Since the specific words used in [4] have exact distance $q-g$ to the code $RS_q[q, g-h]$, the bounded distance decoding and the maximum likelihood decoding are equivalent for those special words. Thus, we may replace bounded distance decoding by the maximum likelihood decoding in the above statement. We now sketch the main ideas.

Let $h(x)$ be a monic irreducible polynomial of degree h in $\mathbf{F}_q[x]$. We shall identify the extension field $\mathbf{F}_{q^h}$ with the residue field $\mathbf{F}_q[x]/(h(x))$. Let α be the class of x in $\mathbf{F}_q[x]/(h(x))$. Then, $\mathbf{F}_q[\alpha] = \mathbf{F}_{q^h}$. Consider the Reed-Solomon code $RS_q[q, g-h]$. For a polynomial $f(x) \in \mathbf{F}_q[x]$ of degree at most $h-1$, let u_f be the received word

$$u_f = (\frac{f(a)}{h(a)} + a^{g-h})_{a \in \mathbf{F}_q}.$$

By assumption, we can write

$$f(\alpha) = \prod_{i=1}^{g} (\alpha + a_i),$$

where $a_i \in \mathbf{F}_q$ are distinct. It follows that as polynomials, we have the identity

$$\prod_{i=1}^{g} (x + a_i) = f(x) + t(x)h(x),$$

where $t(x) \in \mathbf{F}_q[x]$ is some monic polynomial of degree $g-h$. Thus,

$$\frac{f(x)}{h(x)} + x^{g-h} + (t(x) - x^{g-h}) = \frac{\prod_{i=1}^{g}(x + a_i)}{h(x)},$$

where $t(x) - x^{g-h} \in \mathbf{F}_q[x]$ is a polynomial of degree at most $g-h-1$ and thus corresponds to a codeword. This equation implies that the distance of the received word u_f to the code $RS_q[q, g-h]$ is at most $q-g$. If the distance is smaller than $q-g$, then one gets a monic polynomial of degree g with more than g distinct roots. Thus, the distance of u_f to the code is exactly $q-g$.

Let C_f be the set of codewords in $RS_q[q, g-h]$ that has distance exactly $q-g$ to the received word u_f. The cardinality of C_f is then equal to $\frac{1}{g!}$ times the number of ordered ways that $f(\alpha)$ can be written as a product of exactly g

distinct linear factors of the form $\alpha + a$ with $a \in \mathbf{F}_q$. For error radius $q - g$, the maximum likelihood decoding of the received word u_f is the same as finding a solution to the equation

$$f(\alpha) = \prod_{i=1}^{g} (\alpha + a_i),$$

where $a_i \in \mathbf{F}_q$ being distinct.

To show that the discrete logarithm in $\mathbf{F}_{q^h}^*$ can be reduced to the decoding of the words of the type u_f, we apply the index calculus algorithm. Let $b(\alpha)$ be a primitive element of $\mathbf{F}_{q^h}^*$. Taking $f(\alpha) = b(\alpha)^i$ for a random $0 \le i \le q^h - 2$, the maximum likelihood decoding of the word u_f gives a relation

$$b(\alpha)^i = \prod_{j=1}^{g} (\alpha + a_j(i)),$$

where $a_j(i) \in \mathbf{F}_q$ are distinct for $1 \le j \le g$. This gives the congruence equation

$$i \equiv \sum_{j=1}^{g} \log_{b(\alpha)}(\alpha + a_j(i)) \pmod{q^h - 1}.$$

Repeating the decoding and let i vary, this would give enough linear equations in the q variables $\log_{b(\alpha)}(\alpha + a)$ $(a \in \mathbf{F}_q))$. Solving the linear system modulo $q^h - 1$, one finds the values of $\log_{b(\alpha)}(\alpha + a)$ for all $a \in \mathbf{F}_q$. To compute the discrete logarithm of an element $v(\alpha) \in \mathbf{F}_{q^h}^*$ with respect to the base $b(\alpha)$, one applies the decoding to the element $v(\alpha)$ and finds a relation

$$v(\alpha) = \prod_{j=1}^{g} (\alpha + b_j),$$

where the $b_j \in \mathbf{F}_q$ are distinct. Then,

$$\log_{b(\alpha)} v(\alpha) \equiv \sum_{j=1}^{g} \log_{b(\alpha)}(\alpha + b_j) \pmod{q^h - 1}.$$

In this way, the discrete logarithm of $v(\alpha)$ is computed. The detailed analysis can be found in [4]. □

The above theorem is the starting point of our method. In order to use it, one needs to get good information on the integer g satisfying the assumption of the theorem. This is a difficult theoretical problem in general. It can be done in some cases, with the help of Weil's character sum estimate together with a simple sieving. Precisely, the following result was proved for g in [4].

Theorem 5. *Let $h < g$ be positive integers. Let*

$$N(g,h) = \frac{1}{g!}\left(\frac{q^g - \binom{g}{2} q^{g-1}}{q^h - 1} - (1 + \binom{g}{2})(h-1)^g q^{g/2}\right).$$

Then every element in $\mathbf{F}_{q^h}^*$ *can be written in at least* $N(g,h)$ *ways as a product of exactly* g *distinct linear factors of the form* $\alpha + a$ *with* $a \in \mathbf{F}_q$.

If for some constant $\epsilon > 0$, *we have*

$$q \geq \max(g^2, (h-1)^{2+\epsilon}), \quad g \geq (\frac{4}{\epsilon} + 2)(h+1),$$

then

$$N(g,h) \geq q^{g/2}/g! > 0.$$

The main draw back of the above theorem is the condition $q \geq g^2$, which translates to the condition that the information rate $(g-h)/q$ goes to zero in applications.

3 The Result for Rate $c = 1$

Now we show that Theorem 1 holds when the information rate approaches one.

Proposition 6. *Let* g, h *be positive integers such that for some constant* $\epsilon > 0$, *we have*

$$q \geq \max(g^2, (h-1)^{2+\epsilon}), \quad g \geq (\frac{4}{\epsilon} + 2)(h+1).$$

Then, every element in $\mathbf{F}_{q^h}^*$ *can be written in at least* $N(g,h)$ *ways as a product of exactly* $q-g$ *distinct linear factors of the form* $\alpha + a$ *with* $a \in \mathbf{F}_q$.

To prove this proposition, we observe that the map that sends $\beta \in \mathbf{F}_{q^h}^*$ to $\prod_{a \in \mathbf{F}_q}(\alpha + a)/\beta$ is one-to-one from $\mathbf{F}_{q^h}^*$ to itself. **Proof:** Note that

$$\prod_{a \in \mathbf{F}_q} (\alpha + a) \neq 0.$$

Given an element $\beta \in \mathbf{F}_{q^h}^*$, from Theorem 5, we have that $\prod_{a \in \mathbf{F}_q}(\alpha + a)/\beta$ can be written in at least $N(g,h)$ ways as a product of exactly g distinct linear factors of the form $\alpha + a$ with $a \in \mathbf{F}_q$, hence β can be written in at least $N(g,h)$ ways as a product of exactly $q-g$ distinct linear factors of the form $\alpha + a$ with $a \in \mathbf{F}_q$. □

It follows from Theorem 4 that we have the following two results.

Proposition 7. *Suppose that*

$$q \geq \max(g^2, (h-1)^{2+\epsilon}), \quad g \geq (\frac{4}{\epsilon} + 2)(h+1).$$

Then the maximum likelihood decoding $RS_q[q, q-g-h]$ *is as hard as the discrete logarithm over the finite field* $\mathbf{F}_{q^h}^*$.

Note that the rate $(q-g-h)/q$ approaches 1 as q increases for $g = O(\sqrt{q})$ and $h = O(g) = O(\sqrt{q})$.

Proposition 8. *Suppose that*

$$q \geq \max(g^2, (h-1)^{2+\epsilon}), \quad g \geq (\frac{4}{\epsilon}+2)(h+1).$$

Let $h(x)$ be an irreducible polynomial of degree h over $\mathbf{F}_q$ and let $f(x)$ be a nonzero polynomial of degree less than h over $\mathbf{F}_q$. Then in Reed-Solomon code $RS_q[q, q-g-h]$, the Hamming ball centered at $(\frac{f(a)}{h(a)} + a^{q-g-h})_{a\in\mathbf{F}_q}$ of radius g contains at least $\frac{q^{g/2}}{g!}$ many codewords.

Note if we set $g = \lceil\sqrt{q}\rceil$, then the number of codewords is greater than $2^{\sqrt{q}}$, which is subexponential.

Proof of Theorem 3: The relative radius of the Hamming ball in the above proposition is $\frac{g}{g+h+1}$. If $g = \lceil(\frac{4}{\epsilon}+2)(h+1)\rceil$, then the relative radius is approaching to $\frac{\frac{4}{\epsilon}+2}{\frac{4}{\epsilon}+3} = \frac{2\epsilon+4}{3\epsilon+4}$. Select ϵ such that

$$\rho = \frac{2\epsilon+4}{3\epsilon+4}.$$

Note that ϵ can be large if ρ is close to $2/3$. If $g = \lceil q^{\frac{1}{2+\epsilon}}\rceil$, the number of codewords is at least

$$\frac{q^{g/2}}{g!} > (\sqrt{q}/g)^g = q^{\frac{\epsilon g}{2(2+\epsilon)}}.$$

To make sure that this number is greater than q^i, we need $g > \frac{2(2+\epsilon)i}{\epsilon}$. It is satisfied if we let q to be the least prime power that is greater than

$$(\frac{2(2+\epsilon)i}{\epsilon})^{2+\epsilon} = i^{O(1)}.$$

We then calculate $g = \lceil q^{\frac{1}{2+\epsilon}}\rceil$ and solve h from the equation $g = \lceil(\frac{2}{\epsilon}+2)(h+1)\rceil$. Finally we find an irreducible polynomial $h(x)$ of degree h over $\mathbf{F}_q$ using the algorithm in [9]. □

4 The Result for Rate $0 < c < 1$

We now consider the positive rate case with $0 < c < 1$. For this purpose, we take $q = q_1^m$ with $m \geq 2$. Let α be an element in $\mathbf{F}_{q^h}$ with $\mathbf{F}_{q_1}[\alpha] = \mathbf{F}_{q^h}$. Since

$$\mathbf{F}_{q_1}[\alpha] \subseteq \mathbf{F}_q[\alpha] \subseteq \mathbf{F}_{q^h},$$

we also have $\mathbf{F}_{q^h} = \mathbf{F}_q[\alpha]$.

Theorem 9. *Let $q = q_1^m$ with $m \geq 2$. Let g_1 and g_2 be non-negative integers with $g_2 \leq q - q_1$. Let*

$$N(g_1,g_2,h,m) = \frac{1}{g_1!}\left(\frac{q_1^{g_1} - \binom{g_1}{2}q_1^{g_1-1}}{q_1^{mh}-1} - (1+\binom{g_1}{2})(mh-1)^{g_1}q_1^{g_1/2}\right)\binom{q-q_1}{g_2}$$

Then, every element in $\mathbf{F}_{q^h}^*$ *can be written in at least* $N(g_1, g_2, h, m)$ *ways as a product of exactly* $g_1 + g_2$ *distinct linear factors of the form* $\alpha + a$ *with* $a \in \mathbf{F}_q$.

If for some constant $\epsilon > 0$, *we have*

$$q_1 \geq \max(g_1^2, (mh-1)^{2+\epsilon}), \quad g_1 \geq (\frac{4}{\epsilon} + 2)(mh+1)$$

then

$$N(g_1, g_2, h, m) \geq \frac{q_1^{g_1/2}}{g_1!} \binom{q - q_1}{g_2} > 0.$$

Proof. Since $g_2 \leq q - q_1$, we can choose g_2 distinct elements $b_1, \cdots, b_{g_2}$ from the set $\mathbf{F}_q - \mathbf{F}_{q_1}$. For any element $\beta \in \mathbf{F}_{q^h}^* = \mathbf{F}_{q_1^{mh}}^*$, since $\mathbf{F}_{q_1}[\alpha] = \mathbf{F}_{q_1^{mh}}$, we can apply Theorem 5 to deduce that

$$\frac{\beta}{(\alpha + b_1) \cdots (\alpha + b_{g_2})} = (\alpha + a_1) \cdots (\alpha + a_{g_1}),$$

where the $a_i \in \mathbf{F}_{q_1}$ are distinct. The number of such sets $\{a_1, a_2, a_3, \cdots, a_{g_1}\} \subseteq \mathbf{F}_{q_1}$ is greater than

$$\frac{1}{g_1!} \left(\frac{q_1^{g_1} - \binom{g_1}{2} q_1^{g_1 - 1}}{q_1^{mh} - 1} - (1 + \binom{g_1}{2})(mh-1)^{g_1} q_1^{g_1/2} \right).$$

Since $\mathbf{F}_{q_1}$ and its complement $\mathbf{F}_q - \mathbf{F}_{q_1}$ are disjoint, it follows that

$$\beta = (\alpha + b_1) \cdots (\alpha + b_{g_2})(\alpha + a_1) \cdots (\alpha + a_{g_1})$$

is a product of exactly $g_1 + g_2$ distinct linear factors of the form $\alpha + a$ with $a \in \mathbf{F}_q$. □

We now take $g_1 = \lfloor q^{1/2m} \rfloor = \lfloor \sqrt{q_1} \rfloor$ and $g_2 = \lfloor cq \rfloor - g_1$ in the above theorem. Thus, $g_1 + g_2 = \lfloor cq \rfloor$. We need g_2 satisfying the inequalities

$$0 \leq g_2 \leq q - q_1 = q - q^{1/m}.$$

That is,

$$0 \leq \lfloor cq \rfloor - \lfloor q^{1/2m} \rfloor \leq q - q^{1/m}.$$

The left side inequality is satisfied if $q_1 \geq c^{-2/(2m-1)}$. The right side inequality is satisfied if $q_1 \geq (1-c)^{-1/(m-1)}$. Thus, we obtain

Theorem 10. *Let* $m \geq 2$ *and* $h \geq 2$ *be two positive integers such that* $q = q_1^m$. *Let* $0 < c < 1$ *be a constant such that*

$$q_1 \geq \max((mh-1)^{2+\epsilon}, (\frac{4}{\epsilon} + 2)(mh+1)^2, c^{\frac{-2}{2m-1}}, (1-c)^{\frac{-1}{m-1}})$$

for some constant $\epsilon > 0$. *Then, every element in* $\mathbf{F}_{q^h}^*$ *can be written as a product of exactly* $\lfloor cq \rfloor$ *distinct linear factors of the form* $\alpha + a$ *with* $a \in \mathbf{F}_q$.

Combining this theorem together with Theorem 4, we deduce

Theorem 11. *Let $m \geq 2$ and $h \geq 2$ be two positive integers such that $q = q_1^m$. Let $0 < c < 1$ be a constant such that*

$$q_1 \geq \max((mh-1)^{2+\epsilon}, (\frac{4}{\epsilon}+2)(mh+1)^2, c^{\frac{-2}{2m-1}}, (1-c)^{\frac{-1}{m-1}})$$

for some constant $\epsilon > 0$. Then, the maximum likelihood decoding of the Reed-Solomon code $RS_q[q, \lfloor cq \rfloor - h]$ is at least as hard (in random time $q^{O(1)}$ reduction) as the discrete logarithm in $\mathbf{F}_{q^h}^*$.

Taking $m = 2$ in this theorem, we deduce Theorem 1.

Proposition 12. *Let h be a positive integer and $0 < c < 1$ be a constant. Let q_1 be a prime power such that*

$$q_1 \geq \max((2h-1)^{2+\epsilon}, (\frac{4}{\epsilon}+2)(2h+1)^2, c^{-2/3}, (1-c)^{-1}) \tag{1}$$

for some constant $\epsilon > 0$. Let $q = q_1^2$. Let $h(x)$ be an irreducible polynomial of degree h over $\mathbf{F}_q$ whose root α satisfies that $\mathbf{F}_{q_1}[\alpha] = \mathbf{F}_{q^h}$. Let $f(x)$ be a nonzero polynomial over $\mathbf{F}_q$ of degree less than h. Then in the Reed-Solomon code $RS_q[q, \lfloor cq \rfloor - h]$, the Hamming ball centered at $(\frac{f(a)}{h(a)} + a^{\lfloor cq \rfloor - h})_{a \in \mathbf{F}_q}$ of radius $q - \lfloor cq \rfloor$ contains at least $exp(\Theta(q))$ many codewords.

Proof: The number of codewords in the ball is greater than

$$\frac{q_1^{\lfloor \sqrt{q_1} \rfloor / 2}}{\lfloor \sqrt{q_1} \rfloor !} \binom{q - q_1}{\lfloor cq \rfloor - \sqrt{q_1}},$$

which is greater than $\binom{q-q_1}{\lfloor cq \rfloor - \sqrt{q_1}} = exp(\Theta(q))$. □

Proof of Theorem 2. Let q to be the square of the i-th prime power (listed in increasing order). Assume that i is large enough such that $\sqrt{q} \geq \max(c^{-2/3}, (1-c)^{-1})$. We then let ϵ to be $1/\log q$ and h to be the largest integer satisfying (1). It remains to find an irreducible polynomial of degree h over $\mathbf{F}_q$, whose root α satisfies that $\mathbf{F}_{q_1}[\alpha] = \mathbf{F}_{q^h}$. Let p be the characteristic of $\mathbf{F}_q$. We can use α such that $\mathbf{F}_p[\alpha] = \mathbf{F}_{q^h}$. We need to find an irreducible polynomial of degree $h \log_p q$ over $\mathbf{F}_p$. It can be done in time polynomial in p and the degree [9]. Then we factor the polynomial over $\mathbf{F}_q$ and take any factor to be $h(x)$. As for $f(x)$, we may simply let $f(x) = 1$. □

5 Conclusion and Future Research

In this paper, we show that the maximum likelihood decoding of the Reed-Solomon code is at least as hard as the discrete logarithm for any given information rate. In our result, we assumed that the cardinality of the finite field is composite. While this is not a problem in practical applications, e.g. $q = 256$ is

quite popular, it would be interesting to remove this restriction, that is, allowing prime finite fields as well.

Many important questions about decoding Reed-Solomon codes remain open. For example, little is known about the exact list decoding radius of Reed-Solomon codes. In particular, does there exist a Hamming ball of relative radius less than one that contains super-polynomial many codewords in Reed-Solomon codes of rate less than one?

References

1. Joux, N.S.A., Lercierand, R., Vercauteren, F.: The number field sieve in the medium prime case. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 326–344. Springer, Heidelberg (2006)
2. Ben-Sasson, E., Kopparty, S., Radhakrishnan, J.: Subspace polynomials and list decoding of Reed-Solomon codes. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 207–216 (2006)
3. Cheng, Q., Murray, E.: On deciding deep holes of Reed-Solomon codes. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 296–305. Springer, Heidelberg (2007)
4. Cheng, Q., Wan, D.: On the list and bounded distance decodability of Reed-Solomon codes. SIAM Journal on Computing 37(1), 195–209 (2007); Special Issue on FOCS 2004
5. Dumer, I., Micciancio, D., Sudan, M.: Hardness of approximating the minimum distance of a linear code. IEEE Transactions on Information Theory 49(1), 22–37 (2003)
6. Guruswami, V., Vardy, A.: Maximum-likelihood decoding of Reed-Solomon codes is NP-hard. IEEE Transactions on Information Theory 51(7), 2249–2256 (2005)
7. Guruswami, V., Rudra, A.: Limits to list decoding Reed-Solomon codes. IEEE Transactions on Information Theory 52(8), 3642–3649 (2006)
8. Guruswami, V., Sudan, M.: Improved decoding of Reed-Solomon and algebraic-geometry codes. IEEE Transactions on Information Theory 45(6), 1757–1767 (1999)
9. Shoup, V.: New algorithms for finding irreducible polynomials over finite fields. Mathematics of Computation 54, 435–447 (1990)

Computational Complexity of the Distance Constrained Labeling Problem for Trees (Extended Abstract)

Jiří Fiala[1], Petr A. Golovach[2,⋆], and Jan Kratochvíl[1]

[1] Institute for Theoretical Computer Science[⋆⋆]
and Department of Applied Mathematics,
Charles University, Prague, Czech Republic
{fiala,honza}@kam.mff.cuni.cz
[2] Institutt for informatikk,
Universitetet i Bergen, Norway
petr.golovach@ii.uib.no

Abstract. An $L(p,q)$-labeling of a graph is a labeling of its vertices by nonnegative integers such that the labels of adjacent vertices differ by at least p and the labels of vertices at distance 2 differ by at least q. The span of such a labeling is the maximum label used. Distance constrained labelings are an important graph theoretical approach to the Frequency Assignment Problem applied in mobile and wireless networks.

In this paper we show that determining the minimum span of an $L(p,q)$-labeling of a tree is NP-hard whenever q is not a divisor of p. This demonstrates significant difference in computational complexity of this problem for $q = 1$ and $q > 1$. In addition, we give a sufficient and necessary condition for the existence of an $H(p,q)$-labeling of a tree in the case when the metric on the label space is determined by a strongly vertex transitive graph H. This generalizes the problem of distance constrained labeling in cyclic metric, that was known to be solvable in polynomial time for trees.

1 Introduction

Distance constrained graph labelings stem from the highly practical problem of assigning frequencies to transmitters in order to avoid, or minimize, undesired interference. Suppose that the metric in the frequency space is expressible by a graph H. An $H(p,q)$-labeling of a graph G is defined as a mapping $f : V(G) \to V(H)$ such that $\mathrm{dist}_H(f(u), f(v)) \geq p$ for any two adjacent vertices $u, v \in V(G)$, and $\mathrm{dist}_H(f(u), f(v)) \geq q$ for any two nonadjacent vertices $u, v \in V(G)$ which have a common neighbor (i.e., are at distance 2 in G). Here the vertices of the graph G correspond to the transmitters in the network, and the edges of G express possible interference. This general approach was first studied in the

⋆ Supported by Norwegian Research Council.
⋆⋆ Supported by the Ministry of Education of the Czech Republic as project 1M0021620808.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 294–305, 2008.

connection to locally injective graph homomorphisms [7]. Two special cases have been introduced and intensively studied before — the case of linear metric, where $H = P_{\lambda+1}$ is the path of length λ, and the cyclic metric corresponding to the case $H = C_\lambda$.

These mappings are referred to as $L(p,q)$- and $C(p,q)$-labelings, respectively. In both cases λ is the span of the labeling. We define the linear span $L_{p,q}(G)$ as the minimum span of an $L(p,q)$-labeling of G. Analogously, the cyclic span $C_{p,q}(G)$ is the minimum span of a labeling with the cyclic metric.

The concept of $L(2,1)$-labeling was introduced by Roberts [18,11]. The exact values for special graphs and graph classes have been determined several works, cf. surveys [14,19,2] Griggs and Yeh [11] conjectured that $L_{2,1}(G) \leq \Delta^2(G)$, where $\Delta(G)$ denotes the maximum degree in G. This upper bound has been recently proven true for every sufficiently large $\Delta(G)$ by Havet et al. [13].

Distance constrained graph labelings provide a rather interesting graph invariant from the computational complexity point of view. Griggs and Yeh [11] proved that it is NP-hard to determine $L_{2,1}(G)$, while Fiala et al.[8] proved that deciding $L_{2,1}(G) \leq k$ is NP-complete for every fixed $k \geq 4$. Rather interesting is the complexity for restricted graph classes. Chang and Kuo [4] described a polynomial time algorithm for determining the $L_{2,1}(G)$ if G is a tree, but already for series-parallel graphs this problem becomes NP-complete [5]. The computational complexity of determining the $L_{p,q}(G)$ if G is a tree for $q > 1$ has been open since then. It was explicitly asked by D. Welsh [private communication during a graph coloring workshop at DIMACS in 1999] with a hope for a generalization of the method of Chang and Kuo. While this works easily for an arbitrary $p > 2$ and $q = 1$, the case $q > 1$ kept resisting all attempts. Intuitively, the difference between $q = 1$ and $q > 1$ relates to the difference between systems of *distinct* and *distant* representatives [10]. Resolving this question is the main result of this paper.

Formally, we consider the following decision problem:

$L(p,q)$-Labeling
Instance: A graph G and an integer λ.
Question: Does G allow an $L(p,q)$-labeling of span λ?

Note that p, q are fixed parameters while λ (and, of course, G) are part of the input. We prove the following result.

Theorem 1. *For positive integers p and q, the $L(p,q)$-Labeling problem restricted to trees is solvable in polynomial time only if q divides p, otherwise it is* NP*-complete.*

The polynomial part is now a folklore. It has been proved already in [11] that an optimum labeling using only labels of the form $ap + bq$ always exists, and hence $L(rp, rq)$-Labeling is equivalent to $L(p,q)$-Labeling. In particular, when q divides p we get the $L(\frac{p}{q}, 1)$-Labeling problem which is solvable in polynomial time by a slight modification of an algorithm by Chang and Kuo [4,3]. We note here that it is sufficient to prove the NP-hardness part of the theorem for the case of mutually prime p and q.

The NP-hardness part of the theorem is proved by a reduction from the problem of deciding the existence of a system of distant representatives in systems of symmetric sets. The construction extends the approach initiated in [9] where the NP-hardness of extending a prelabeling to a complete $L(p,q)$-labeling was proved. The main idea is a construction of trees that allow only specific labels on their roots. The main difficulty, that we successfully managed to overcome, was to keep the size of such trees polynomial. These constructions are presented in Section 3 and the proof of Theorem 1 is concluded in Section 5.

It is interesting to note that $L_{p,q}(T)$ can be approximated well for trees by $q\Delta(T)$. Griggs and Yeh proved that $\Delta(T)q+p-q \leq L_{p,q}(T) \leq \Delta(T)q+2p-q-1$ holds for positive integers $p \geq q$ and a tree T [11]. But perhaps a bit surprising is the fact that in the cyclic metric the span of a tree is uniquely determined by its maximum degree. Liu and Zhu [16] proved that $C_{p,q}(T) = q\Delta(T)+2p-q$ for every tree T and $p \geq q$. In particular, the $C_{p,q}$-span of a tree can be computed in linear time. We further explore this phenomenon and show that it is due to the fact that cycles are transitive graphs. We prove a necessary and sufficient condition for the existence of an $H(p,q)$-labeling of a tree T when H is a strongly transitive graph in Theorem 2. This result has several applications. For instance, the minimum n such that an input tree T has a $Q_n(p,q)$-labeling (where Q_n is the n-dimensional cube) can be determined in polynomial time for $q = 1$ and $q = 2$. These results are presented in Section 6.

2 Preliminaries and Notation

For integers i and j, we denote by $[i,j]$ the interval $\{i, i+1, i+2, \ldots, j\}$. By convention, $[i,j] = \emptyset$ if $j < i$. Analogously, if $i \equiv j \pmod q$, we denote by $[i,j]_{\equiv q}$ the q-stepped interval $\{i, i+q, i+2q, \ldots, j\}$. For a positive integer k, we write $[k] := [1,k]$. The binary operators div and mod stand for the integral division and the remainder of the division.

We consider undirected graphs without loops or multiple edges. In a graph G, the symbol $N_G(u)$ denotes the set of vertices adjacent to u, i.e., the (open) *neighborhood* of u. We also define the *closed neighborhood* as $N_G[u] := N_G(u) \cup \{u\}$. The subscripts G will be omitted if there is no danger of confusion which graph G is being considered. The symbol $\Delta(G)$ stands for the maximum degree of a vertex in the graph G.

A graph is connected if every pair of vertices can be connected by a path. For vertices $u, v \in V_G$, the *distance* $\mathrm{dist}_G(u,v)$ is the length of a shortest path between u and v.

We adopt standard notions from graph theory: the path P_n on n vertices; the cycle C_n; a star — a connected graph with at most one vertex of degree greater than one; a tree — a connected graph with no cycle; and a hypercube Q_n — a graph on binary words of length n where two such words are adjacent if they differ only at one position. For more details we refer to the classical monograph by Harary [12] or to a more recent textbook by Matoušek and Nešetřil [17].

When exploring $L(p,q)$-labelings in the first part of the paper, we assume that the label set is a set $[0,\lambda]$. Thus an *$L(p,q)$-labeling of G of span λ* is a mapping $l : V_G \to [0,\lambda]$ such that for any pair of adjacent vertices u and v, it holds that $|l(u) - l(v)| \geq p$, and for any pair of nonadjacent vertices u and v that share a common neighbor, it holds that $|l(u) - l(v)| \geq q$.

For a fixed λ we define the *reversed* mapping on $[0,\lambda]$ by $a \to \bar{a} := \lambda - a$. Observe that for any $L(p,q)$-labeling l of span λ, the reversed labeling $\bar{l}$ defined by $\bar{l}(u) := \overline{l(u)}$ is also an $L(p,q)$-labeling. We extend the reversing to sets by $\overline{S} = \{\bar{a} \,|\, a \in S\}$.

We extend any mapping f defined on vertices of a graph G into a mapping on sets of vertices by letting $f(W) := \bigcup_{u \in W} \{f(u)\}$ for each $W \subset V_G$.

In the context of fixed p, q and λ we say that a label $a \in [0,\lambda]$ is *feasible* for a vertex $u \in V_G$ if there exists an $L(p,q)$-labeling l of G of span λ such that $l(u) = a$. The set of feasible labels for u is called the *feasible set of u in G*. Observe that every feasible set S is reversable: $S = \overline{S}$. If a symmetric set S is expressed as the union of a set and its reverse $S = S' \cup \overline{S'}$, we abbreviate this expression by the notation $S = S' \cup \overline{\cdots}$. Finally, we say that a label a is *forced on a vertex $u \in V_G$* if $\{a, \bar{a}\}$ is the feasible set of u.

3 Auxiliary Constructions for the Case $p > q$

Throughout the coming three sections we assume that $p > q > 1$ and that p and q are relatively prime. In our construction we also use a third parameter, an integer k, whose value will be specified later in the polynomial reduction. We will use $\lambda := 2p + kq$ and $d := k + (p \text{ div } q) + 1$.

Construction 1. *For given $p > q > 1$ and $k \geq 2$, let T^1 be the only tree with a vertex u of degree two which is a common neighbor of vertices w and w' of degree d. The other neighbors of w and w' are of degree $k + 2$. All remaining vertices are leaves.*

Construction 2. *For given $p > q > 1$ and $k \geq 2$, let T^2 be the tree obtained from T^1 by adding k leaves to u. Denote by v some leaf adjacent to u. (See Fig. 1.)*

Lemma 1. *If p, q and k satisfy the assumptions of Constructions 1 and 2, then $[p, \bar{p}]_{\equiv q}$ is the feasible set for u in T_1 and $[q, \overline{2p}]_{\equiv q} \cup \overline{\cdots}$ is the feasible set for v in T_2.*

Note that for each choice of $a \in [p, \bar{p}]_{\equiv q}$ and $b \in [q, a - p]_{\equiv q} \cup [a + p, \bar{q}]_{\equiv q}$, a labeling l of T^2 exists where $l(u) = a$ and $l(v) = b$.

To simplify the next construction we define $d' := d - 2 = k - 1 + (p \text{ div } q)$.

Construction 3. *For given $p > q > 1$, $k \geq \frac{2p}{q} + 1$ and $i \in [k-1]$, let T^3_i be the tree constructed as follows: Take the disjoint union of i copies of the tree T^1 and*

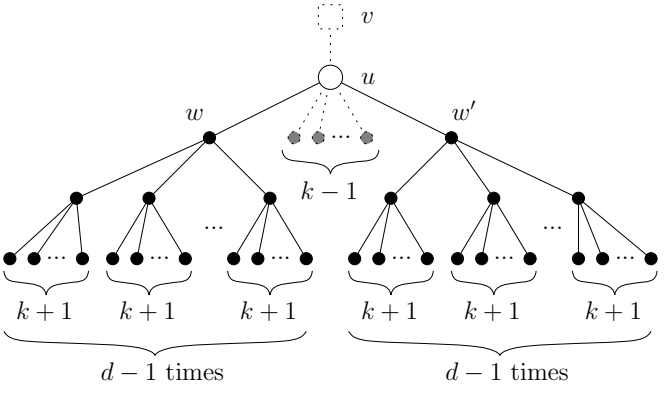


Fig. 1. The tree T^1 (solid lines) and tree T^2 (solid and dotted lines)

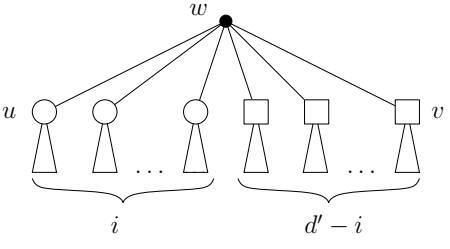


Fig. 2. The tree T_i^3

$d' - i$ copies of T^2. Then insert a new vertex w and connect it to the i vertices u of trees T^1 as well as to the $d' - i$ vertices v of trees T^2.

Rename the vertices such that only u of the first copy of T^1 and v of the last copy of T^2 keep their names. (See Fig. 2.)

Lemma 2. *Assume that p, q, k, and i satisfy the assumptions of Construction 3. Then*

- *the set $[p+q, p+iq]_{\equiv q} \cup \overline{\cdots}$ is the feasible set for u in T_i^3,*
- *$[(d'-i)q, \bar{q}]_{\equiv q} \cup \overline{\cdots}$ is the feasible set for v in T_i^3, and*
- *q is forced on w.*

Proof. Let l be a hypothetical labeling. We show that this labeling is unique upto an isomorphism of T_i^3 and upto reversion of the labeling.

Observe that for the closed neighborhood of w it holds that $l(N[w]) \subseteq [q, \bar{q}]$, since for every vertex in this set we identify vertices labeled by 0 and λ at distance at most two. (This follows from the labeling described in Lemma 1.)

As the degree of w is at least k we exclude the case $l(w) \in [p+q, \overline{p+q}]$ by the same argument as in Lemma 1. Assume without loss of generality that $l(w) \in [q, p+q]$. Consequently, for the open neighborhood of w we get that $l(N(w)) \subseteq [p+q, \bar{q}]$.

We also assume without loss of generality that $l(u) < l(v)$: if the maximal label on $N(w)$ was on some vertex u of the first copies, then $l(N(w)) \subset [p, \bar{p}]$. This interval is not long enough to accommodate $k+1$-many q-distant labels.

If we choose v such that it receives the maximal label on $N(w)$, we see that $l(v) \equiv 2p \pmod q$. Hence, at least once the distance between consecutive labels on $N(w)$ is at least $q + (p \bmod q)$.

The only way how labels of $N(w)$ can be arranged into the interval $[p+q, \bar{q}]$ is to use the arithmetic progression $[p+q, p+iq]_{\equiv q}$ on the first i copies of u, and then after the gap $q + (p \bmod q)$ to use the set $[\overline{(d'-i)q}, \bar{q}]_{\equiv q}$ on the copies of v. In all other ways a gap greater than q would be used at least twice, and the above arrangement is already tight: the smallest possible label on v is $p + iq + q + (p \bmod q) = 2p + kq - (k - 1 + (p \operatorname{div} q) - i)q$.

As the smallest label used on $N(w)$ is $p+q$, the label q is forced on w.

The above described labeling of $N(w)$ can be extended to the rest of the tree T_i^3. In the copies of T_1, the labels of u and w comply with the labelings mentioned after Lemma 1. In the remaining $d' - i$ copies of T^2, we choose $l(u) = p + q$ if $l(v) > 2p + q$ and $l(u) = \bar{p}$ otherwise. The choice of k in the Definition 3 assures that this partial labeling can be extended on each copy of T^2.

In the following two lemmas we show constructions of trees that force exact labels on some vertices.

Construction 4. *For $p > q > 1$ and an even $k \geq \frac{2p}{q} + 1$, take the disjoint union of $k-2$ trees: one copy of T_1^3, one copy of $T_{\frac{k}{2}}^3$, and two copies of each tree T_i^3 for $i \in [2, \frac{k}{2} - 1]$. Insert an extra new vertex w and make it adjacent to each of the $k-2$ vertices u. The resulting graph is the tree T^4. Rename the vertices such that u_i is the vertex u of T_i^3, i.e., of one of the two isomorphic copies when $i \in [2, \frac{k}{2} - 1]$. (See Fig. 3.)*

Lemma 3. *If p, q and k satisfy the assumptions of Construction 4, then in T^4, $p + iq$ is forced on u_i for each $i \in [\frac{k}{2}]$, and $2q$ is forced on w.*

Construction 5. *For $p > q > 1$ and an even $k \geq \frac{2p}{q} + 1$, construct the tree T^5 from the disjoint union of pairs of trees T_i^3 for $i \in [\frac{k}{2}]$ by adding an extra new vertex w and making it adjacent to all vertices v. Rename the vertices such that for each i, v_i is the vertex v of one of the two copies of T_i^3.*

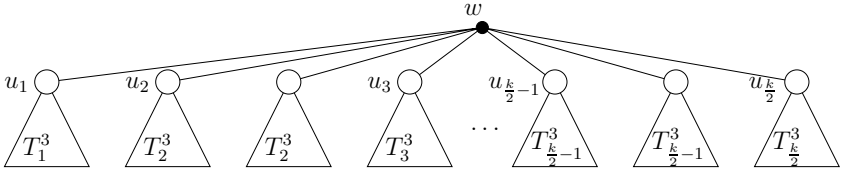


Fig. 3. The Tree T^4

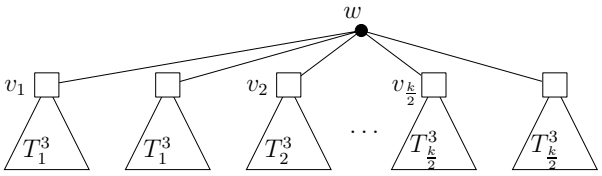


Fig. 4. The Tree T^5

Lemma 4. *Suppose p, q, and k satisfy the assumptions of Construction 5. Then in T^5, $p + \frac{k}{2}q$ is forced on w, and for every $i \in [\frac{k}{2}]$, iq is forced on v_i.*

Let $\Lambda(k, p, q)$ (called the *set of applicable labels*) be the set of labels that are forced on some vertex u_i of T^4 or on some v_i of T^5. By Lemmas 4 and 5,

$$\Lambda(k,p,q) := \left(\left[p+q, p+\frac{kq}{2}\right]_{\equiv q} \cup \left[q, \frac{kq}{2}\right]_{\equiv q} \right) \cup \overline{\cdots}$$

4 Symmetric Systems of q-Distant Representatives

As a technical tool for proving NP-hardness results we use the following problem of finding distant representatives:

System of q-distant representatives	Sq-DR
Parameter: A positive integer q.	
Instance: A collection of sets $R_i, i \in [m]$ of integers.	
Question: Is there a collection of elements $r_i \in R_i$, $i \in [m]$ that pairwise differ by at least q	

It is known that the S1-DR problem allows a polynomial time algorithm (by finding a maximum matching in a bipartite graph), while for all $q \geq 2$ the Sq-DR problem is NP-complete, even if each set R_i has at most three elements [10].

We adjust the Sq-DR problem so that it can be used to prove NP-hardness for the $L(p, q)$-Labeling problem for trees. Since every $L(p, q)$-labeling l comes together with the reversed labeling $\bar{l}$, we need a stronger concept of systems of q-distant representatives where the sets are λ-symmetric, i.e., $R_i = \overline{R_i}$ for every $i \in [m]$. Moreover, we say that a set R is $2q$-sparse if the distance between any two elements in R is at least $2q$.

Lemma 5. *For any $p > q > 1$, the* Sq-DR *problem remains* NP*-complete even when restricted to instances where each R_i*

- *is of size at most 6,*
- *is $2q$-sparse*
- *is λ-symmetric for $\lambda = 2p + (6n + 2(p \text{ div } q) + 4)q$, and*
- *is a subset of $\Lambda(6n + 2(p \text{ div } q) + 4, p, q) \cap \left([p + 2q, p + 3nq] \cup \overline{\cdots}\right)$*

where n is a suitable integer.

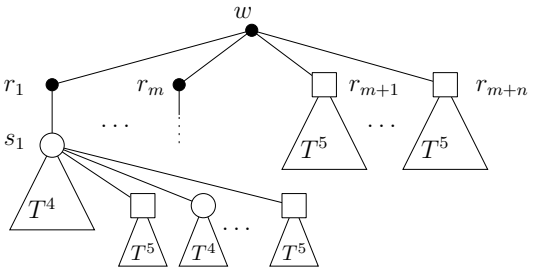


Fig. 5. The final tree T^6

5 Proof of Theorem 1 for $p > q$

Proof. Assume an instance $(R_i)_{i\in[m+n]}$ of the Sq-DR problem with properties described in Lemma 5, and let $|R_i| = 4$ or 6 for $i \in [m]$ and $|R_{m+i}| = 2$ for $i \in [n]$. We construct a tree T^6 with special vertices r_i such that these vertices share a common neighbor and force for every $i \in [m+n]$ that r_i can not be labeled by any label outside the set R_i under any $L(p,q)$-labeling of span λ of T^6.

Assume that the set $R_i, i \in [m]$ consists of elements $\{a, b, \bar{b}, \bar{a}\}$ with $a < b$.

We choose an auxiliary set $S_i \subset [q, p+(3n+1)q]$ of applicable labels such that $\{q, a-q, a+q, b-q, b+q, p+(3n+1)q\} \subset S_i$. The set S_i contains also sufficiently many other labels such that the distance between consecutive elements of S_i is at least q but strictly less than $2q$ with only two exceptions: $a-q, a+q$ and $b-q, b+q$. (A construction of such set can be given explicitly, but we would like to avoid excessive formalism.) As the set R_i is $2q$-sparse, each S_i is nonempty.

Analogously, we construct the sets S_i for all R_i with six elements.

We construct tree T^6 as follows:

1. For each R_i with four or six elements
   (a) Insert into T^6 a new copy of T^4 and rename its vertex $u_{\frac{k}{2}}$ by s_i
   (b) For each element $p+jq$ of S_i add two copies of the tree T^4 and make s_i adjacent to both vertices u_j.
   (c) Analogously, for each $jq \in S_i$ add two copies of the tree T^5 and make s_i adjacent to both vertices v_j.
   (d) Add an extra new leaf r_i adjacent to s_i
2. For each $R_i = \{jq, \overline{jq}\}$ add a copy of T^5 and rename its v_j by r_i.
3. Finally, connect these $m+n$ partial trees by a new common neighbor w of all vertices $r_i, i \in [m+n]$. (See Fig. 5.)

We argue that for every $i \in [n]$ only the set R_i is feasible for the vertex r_i in each partial tree constructed in the first step of the construction. As $\frac{\lambda}{2}$ is forced on s_i and both 0 and λ appear on $l(N(s_i))$ inside the copy of T^4 we get that $l(r_i) \in [q, \frac{kq}{2}] \cup \overline{\cdots}$.

The elements of S_i are forced on $N(s_i)$, so only $l(r_i) \in (0 \cup R_i \cup [p + (3n + 2)q, p + \frac{kq}{2}]) \cup \overline{\cdots}$. By the choice of $k = 6n + 2(p \text{ div } q) + 4$ we have $\frac{kq}{2} = (3n + p \text{ div } q + 2)q < p + (3n + 2)q$ and hence $l(r_i) \in R_i$.

Observe that the labelings giving pairs of u_j of step 1b) λ-symmetric labels can be simply combined altogether with any label of r_i from the set R_i to get an $L(p, q)$-labeling of the partial tree.

We conclude the proof by showing that the entire T^6 allows an $L(p, q)$-labeling of span λ if and only if the set system $(R_i)_{i\in[m+n]}$ allows a system of q-distant representatives.

Given a labeling l, the labels of vertices r_i provide valid representatives for R_i. This is since vertices r_i are mutually at distance two and we have shown that their labels can only belong to sets R_i (for $i > m$ this follows directly from Lemma 4).

In the opposite direction, we label each r_i by the representative of R_i and use the corresponding labelings of the partial trees described above. Then w can be labeled by 0 as $l(r_i) > q$ for every $i \in [m + n]$ as well as it holds that $0 \notin N(l(v_j))$ for every feasible labeling of T^5 which was added in the second step (consult Lemma 4).

Though the practical motivation for $L(p, q)$-labelings implies that $p \geq q$, the notion itself makes sense also for $p < q$. The NP-hardness result prevails as well.

6 $H(p, q)$-Labelings for Transitive Target Graphs

Consider the following graph labeling problem with a condition at distance two:

(p, q)-Distance Labeling	(p, q)-DL
Instance: Graphs G and H.	
Question: Does G allow an $H(p, q)$-labeling?	

We emphasize that the target graph H is a part of the input of the problem. The problem of determining $L_{p,q}(G)$ and $C_{p,q}(G)$ is equivalent to the (p, q)-DL problem restricted to graphs H being paths and cycles. In contrary to our former result on NP-completeness of the (p, q)-DL problem for $q > 1$, G being a star and an arbitrary target graph H [6], the (p, q)-DL problem becomes easy when G is a tree and the target graph H is a cycle:

Proposition 1 (Leese and Noble [15], Liu and Zhu [16]). *Let T be a tree, and $p \geq q$ be nonnegative integers. Then a $C_\lambda(p, q)$-labeling of T exists if and only if $\lambda \geq q\Delta(T) + 2p - q$.*

We show now that the change of the complexity of the labeling problem for linear and circular metrics follows from the fact that cycles are vertex-transitive. Recall that a graph H is (strongly) *vertex transitive* if for every two vertices $x, y \in V_H$, the graph H allows an automorphism f swapping vertices x and y, i.e., $f(x) = y$ and $f(y) = x$.

Theorem 2. *Let H be a vertex transitive graph, and p, q be positive integers. An $H(p,q)$-labeling of a tree T exists if and only if the graph H contains vertices $x, y_1, y_2, \ldots, y_{\Delta(T)}$ such that $\mathrm{dist}_H(x, y_i) \geq p$ and $\mathrm{dist}_H(y_i, y_j) \geq q$ hold for every distinct $i, j \in [\Delta(T)]$.*

Proof. Observe that the existence of an $H(p,q)$-labeling of T immediately gives the existence of vertices $x, y_1, \ldots, y_{\Delta(T)}$.

For the opposite implication we construct the labeling by induction. The $H(p,q)$-labeling will satisfy the property that for every vertex u of T and its neighbors $v_1, \ldots v_k$, the graph H allows an automorphism g such that the labels satisfy $l(u) = g(x)$ and for every $i \in [k]$ it holds that $l(v_i) = g(y_j)$ for some $j \in \Delta(T)$.

Firstly, select an arbitrary vertex r of T, and make the tree rooted in r. Also define $l(r) = x$ and extend the labeling l on $N(r)$ such that distinct neighbors of r are mapped onto distinct y_i's. Clearly, the required automorphism g is the identity.

Assume that the labeling is already defined on a vertex u and its parent v, but not yet at the children of u. Let g be the automorphism of H used to distribute labels on $N[v]$ and f be the automorphism swapping vertices $l(u)$ and $l(v)$. We now compose both automorphisms $h := f \circ g$ and extend the labeling on the whole neighborhood of u by using distinct vertices of $h(y_1), \ldots, h(y_{\Delta(T)})$ as labels.

Note that Proposition 1 is a corollary of Theorem 2 since every cycle is vertex transitive. The (p,q)-DL problem for trees is solvable in polynomial time for those classes of target graphs for which the condition of the existence of vertices x and $y_1, \ldots, y_{\Delta(T)}$ can be answered in polynomial time. In particular, this applies for vertex transitive graphs of restricted treewidth as follows from well known results by Arnborg et al. [1].

Corollary 1. *Let $\mathcal{G}$ be a class of vertex transitive graphs with bounded treewidth. Then the (p,q)-DL problem, restricted to input graphs G that are trees and graph H from the class $\mathcal{G}$, can be solved in polynomial time.*

We now consider the case when H is an n-dimensional hypercube Q_n and $q = 1, 2$.

Corollary 2. *Let T be a tree, and H be an n-dimensional hypercube Q_n. Then a tree T allows an $H(p,1)$-labeling if and only if*

$$\Delta(T) \leq \binom{n}{p} + \binom{n}{p+1} + \cdots + \binom{n}{n}.$$

Also, T has an $H(p,2)$-labeling if and only if

$$\Delta(T) \leq \binom{n}{p} + \binom{n}{p+2} + \binom{n}{p+4} + \cdots.$$

Proof. Every hypercube is vertex transitive. Choose $x \in V_{Q_n}$ arbitrarily.

The first claim follows directly from the fact that the number of vertices in Q_n that are at distance at least p from x is exactly $\binom{n}{p} + \binom{n}{p+1} + \cdots + \binom{n}{n}$.

Let U_i be the set of vertices at distance i from x. It is well known that $|U_i| = \binom{n}{i}$. Define $U := U_p \cup U_{p+2} \cup U_{p+4} \cup \ldots$. This U is an independent set and every vertex of U is at distance at least p from x. Let $W := U_{p-2} \cup U_{p-4} \cup U_{p-6} \cup \ldots$. As the union $U \cup W$ is a maximum independent set in Q_n, the set U is a maximum independent set among vertices that are at distance at least p from x. By Theorem 2 the tree T allows an $H(p,2)$-labeling if and only if $\Delta(T) \leq |U|$ and the other claim follows.

Note that for $q \geq 3$ the problem of finding $x, y_1, \ldots, y_{\Delta(T)}$ in Q_n becomes harder, since it requires to compute the set of vertices that are pairwise at distance at least three — in particular none of the layers $U_i, i < n$ can be taken completely.

Finally, observe that for $p = q = 2$ the search for $x, y_1, \ldots, y_{\Delta(T)}$ in a general graph H is equivalent to the test whether H allows an independent set of size at least $\Delta(T) + 1$.

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. J. Algorithms 12(2), 308–340 (1991)
2. Calamoneri, T.: The $L(h,k)$-labeling problem: A survey and annotated bibliography. Computer Journal 49(5), 585–608 (2006)
3. Chang, G.J., Ke, W.-T., Liu, D.D.-F., Yeh, R.K.: On $L(d,1)$-labelings of graphs. Discrete Mathematics 220(1–3), 57–66 (2000)
4. Chang, G.J., Kuo, D.: The $L(2,1)$-labeling problem on graphs. SIAM Journal on Discrete Mathematics 9(2), 309–316 (1996)
5. Fiala, J., Golovach, P.A., Kratochvíl, J.: Distance constrained labelings of graphs of bounded treewidth. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 360–372. Springer, Heidelberg (2005)
6. Fiala, J., Golovach, P. A., and Kratochvíl, J.: Distance constrained labelings of trees. Tech. Rep. ITI Series 2008–369, Charles University (2007)
7. Fiala, J., Kratochvíl, J.: Partial covers of graphs. Discussiones Mathematicae Graph Theory 22, 89–99 (2002)
8. Fiala, J., Kratochvíl, J., Kloks, T.: Fixed-parameter complexity of λ-labelings. Discrete Applied Mathematics 113(1), 59–72 (2001)
9. Fiala, J., Kratochvíl, J., Proskurowski, A.: Distance constrained labeling of precolored trees. In: Restivo, A., Ronchi Della Rocca, S., Roversi, L. (eds.) ICTCS 2001. LNCS, vol. 2202, pp. 285–292. Springer, Heidelberg (2001)
10. Fiala, J., Kratochvíl, J., Proskurowski, A.: Systems of distant representatives. Discrete Applied Mathematics 145(2), 306–316 (2005)
11. Griggs, J.R., Yeh, R.K.: Labelling graphs with a condition at distance 2. SIAM Journal on Discrete Mathematics 5(4), 586–595 (1992)
12. Harary, F.: Graph theory. Addison-Wesley Series in Mathematics IX. Addison-Wesley, Reading (1969)

13. Havet, F., Reed, B., Sereni, J.-S.: L(2,1)-labelling of graphs. In: Huang, S.-T. (ed.) Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008, pp. 621–630. SIAM, Philadelphia (2008)
14. Král, D.: Mixed hypergraphs and other coloring problems. Discrete Mathematics 307(7-8), 923–938 (2007)
15. Leese, R.A., Noble, S.D.: Cyclic labellings with constraints at two distances. Electronic Journal of Combinatorics 11(1) (2004)
16. Liu, D.D.-F., Zhu, X.: Circulant distant two labeling and circular chromatic number. Ars Combinatoria 69, 177–183 (2003)
17. Matoušek, J., Nešetřil, J.: Invitation to Discrete Mathematics. Oxford University Press, Oxford (1998)
18. Roberts, F.S.: T-colorings of graphs: recent results and open problems. Discrete Mathematics 93(2–3), 229–245 (1991)
19. Yeh, R.K.: A survey on labeling graphs with a condition at distance two. Discrete Mathematics 306(12), 1217–1231 (2006)

The Randomized Coloring Procedure with Symmetry-Breaking

Sriram Pemmaraju[1] and Aravind Srinivasan[2,⋆]

[1] Dept. of Computer Science,
The University of Iowa, Iowa City, IA 52242-1419, USA
sriram@cs.uiowa.edu
[2] Dept. of Computer Science and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742, USA
srin@cs.umd.edu

Abstract. A basic *randomized coloring procedure* has been used in probabilistic proofs to obtain remarkably strong results on graph coloring. These results include the asymptotic version of the List Coloring Conjecture due to Kahn, the extensions of Brooks' Theorem to sparse graphs due to Kim and Johansson, and Luby's fast parallel and distributed algorithms for graph coloring. The most challenging aspect of a typical probabilistic proof is showing adequate concentration bounds for key random variables. In this paper, we present a simple symmetry-breaking augmentation to the randomized coloring procedure that works well in conjunction with *Azuma's Martingale Inequality* to easily yield the requisite concentration bounds. We use this approach to obtain a number of results in two areas: *frugal coloring* and *weighted equitable coloring*. A β-*frugal coloring* of a graph G is a proper vertex-coloring of G in which no color appears more than β times in any neighborhood. Let $G = (V, E)$ be a vertex-weighted graph with weight function $w : V \rightarrow [0, 1]$ and let $W = \sum_{v \in V} w(v)$. A *weighted equitable coloring* of G is a proper k-coloring such that the total weight of every color class is "large", i.e., "not much smaller" than W/k; this notion is useful in obtaining tail bounds for sums of dependent random variables.

1 Introduction and Summary of Results

The *randomized coloring procedure* refers to a simple randomized algorithm for coloring graphs that has been used in probabilistic proofs over the past two decades, to obtain remarkably strong results on graph coloring. Some of these results are existential, whereas some lead to polynomial-time algorithms. These results include the asymptotic version of the List Coloring Conjecture due to Kahn [16], and Kim [18] and Johansson's [14] extensions of Brooks' Theorem for

⋆ Supported in part by NSF Award CCR-0208005, NSF ITR Award CNS-0426683, and NSF Award CNS-0626964. Part of this work was done while on sabbatical at the Network Dynamics and Simulation Science Laboratory of the Virginia Bioinformatics Institute, Virginia Tech.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 306–319, 2008.

graphs with lower-bounded girth. The randomized coloring procedure has also been used by Luby [22] to obtain fast parallel and distributed algorithms for graph coloring. In its simplest form, the procedure is:

> Each vertex v picks a *tentative color* uniformly at random from its color palette. With high probability the tentative coloring may not be proper; it is repaired by uncoloring each vertex v that receives the same color as a neighbor.

The randomized coloring procedure allows us to claim the existence of a proper, partial coloring of the graph, having certain desired properties. This coloring can be extended to a complete proper coloring in a variety of ways (for example, greedily). Variants of this procedure involve vertices making a *non-uniform* color choice or each vertex having an *activation probability*. For example, in Johansson's extension of Brooks' Theorem for triangle-free graphs [14], it is important that some colors are used more than others and to ensure this, a non-uniform probability distribution is used for the choice of colors. In Luby's parallel algorithm for graph coloring [22] each node has an activation probability of 1/2; the algorithm starts by flipping an unbiased coin for each vertex and only those vertices that are activated in this step take further part in the coloring procedure.

In the *iterative* or *incremental* version of this procedure, the partial coloring obtained after one application of the procedure is not completed deterministically; instead the coloring is incrementally extended by repeating the procedure. Specifically, after one application of the procedure we argue that (i) sufficient progress has been made and (ii) the partial coloring has the desired properties with positive (though, typically very small) probability. A "good" partial coloring is then fixed and we extend this coloring by performing the next iteration of the randomized coloring procedure. This incremental version of the procedure is extremely powerful and the results [14,16,18,22] mentioned above, are obtained thus. The incremental randomized coloring procedure is a special case of the general technique variously referred to as the "semi-random method," the "pseudo-random method," or the "Rödl Nibble."

In this paper, we present a simple symmetry-breaking augmentation to the randomized coloring procedure. This symmetry-breaking approach, when used in conjunction with *Azuma's Martingale Inequality*, allows us to use the incremental randomized coloring procedure to prove a number of new results (mentioned below). In its simplest form, our approach starts by picking a permutation π of the vertex set V of the given graph G. Depending on the application, the permutation π can be arbitrary or random or dictated by the structure of G. As in the standard randomized coloring procedure, in the first step, vertices pick tentative colors. Then, in the uncoloring step, a vertex v is uncolored *only if there is a neighbor of lower rank in* π *that has received the same tentative color*. Thus the difference between the symmetry-breaking approach and the standard randomized procedure is only in which neighbors are examined in the uncoloring step.

The most challenging aspect of a typical probabilistic-method proof is showing the adequate concentration of key random variables. For example, for the first

result of this paper, we need to show that after an iteration of the randomized coloring, a constant fraction of the neighbors of each vertex get colored. Let the random variable P_v denote the number of neighbors of vertex v that get permanently colored in one iteration of the coloring procedure. It can be shown that $E[P_v] = \alpha \cdot degree(v)$ for some constant $0 < \alpha < 1$; we need to show that P_v is sharply concentrated around $E[P_v]$. In establishing the concentration of P_v, we need to take into account the fact that even though vertices independently chose tentative colors, the permanent acquisition of colors by vertices may be highly correlated. Recall that a vertex u permanently acquires a color x if u has tentatively chosen x and no neighbor of u has. To get around such "dependence" problems, concentration inequalities such as Azuma's Inequality that do not require independence, have been used widely. However, obtaining sharp enough concentration bounds using Azuma's Inequality is not always easy. Consider the following version of the inequality [23]:

Lemma 1. ***[Azuma's Inequality]*** *Let X be a random variable determined by n trials $T_1, T_2, \ldots, T_n$, such that for each i, and any two possible sequences of outcomes $t_1, t_2, \ldots, t_{i-1}, t_i$ and $t_1, t_2, \ldots, t_{i-1}, t_i'$:*

$$\left| E[X|T_1 = t_1, \ldots, T_i = t_i] - E[X|T_1 = t_1, \ldots, T_i = t_i'] \right| \leq c_i \tag{1}$$

then

$$\Pr\left[|X - E[X]| > t\right] \leq 2e^{-t^2/(2\sum c_i^2)}. \tag{2}$$

The difficulty in using Azuma's Inequality arises from the need to show that $\sum_i c_i^2$ is small. For example, if each c_i is shown to be bounded above by a small constant, then the resulting bound in (2) is $e^{-\varepsilon t^2/n}$ for a positive constant ε. The "n" here is problematic, as it may be considerably larger than $E[X]$. The more desirable concentration bound is $e^{-\Omega(\varepsilon t^2/E[X])}$. For our random variable P_v, n may be as large as Δ^2 (Δ being the maximum degree of the graph), whereas $E[P_v]$ is linear in Δ. This is because the permanent acquisition of colors by vertices in the neighborhood of v depends on the tentative choices of colors by vertices in the distance-2 neighborhood of v. The symmetry-breaking approach provides critical help in showing that $\sum_i c_i^2$ is small, usually $O(E[X])$. For example, to show that P_v is concentrated around $E[P_v]$, we consider $U_v = \langle u_1, u_2, \ldots, u_\ell \rangle$, the sequence of vertices at distance at most two from v, arranged in *increasing rank according to* π. Letting T_i denote the tentative choice of a color by u_i, we observe that changing T_i *does not affect any of the lower ranked vertices; only the higher ranked vertices.* However, (1) only considers the *expected* effect of changing T_i on higher ranked vertices, rather than the worst-case effect. These observations play a critical role in proving a better bound on $\sum_i c_i^2$; see, e.g., the proofs of Lemmas 4 and 6.

Results and Notation. We apply randomized coloring with symmetry-breaking to obtain a number of results in two areas: *frugal coloring* and *weighted equitable coloring*. Our results include existential bounds, polynomial-time algorithms, and

polylog-time distributed algorithms. We state our specific results next. All logarithms here are natural logarithms, unless specified otherwise. We use e to denote the base of the natural logarithm, and for any positive integer h, $[h]$ to denote the set $\{1, 2, \ldots, h\}$. For a vertex v, $N(v)$ denotes the set of neighbors of v in some graph G that is clear from the context; $deg(v)$ denotes $|N(v)|$. We make use of the following version of the Lovász Local Lemma.

Lemma 2. ***[Lovász Local Lemma]*** *Consider a set $\mathcal{E}$ of events such that for each $A \in \mathcal{E}$, $\Pr[A] \leq p < 1$, and A is mutually independent of a set of all but at most d other events from $\mathcal{E}$. If $4pd \leq 1$, then with positive probability, none of the events in $\mathcal{E}$ occur.*

Frugal coloring. A *β-frugal coloring* of a graph G is a proper vertex-coloring of G in which no color appears more than β times in any neighborhood. Frugal coloring is a useful subroutine in *total coloring* [10] and can also be used as a subroutine for efficient channel-allocation schemes in multi-channel, multi-radio wireless networks [11]. We obtain four results (F1)-(F4) for frugal coloring; let Δ denote the maximum degree and n denote the number of vertices of the graph.

(F1) Every graph has a $(\Delta + 1)$-coloring that is $O(\frac{\log^2 \Delta}{\log \log \Delta})$-frugal. Such a coloring can be computed in polynomial time. Furthermore, there is a distributed algorithm that can compute an $O(\log \Delta \cdot \frac{\log n}{\log \log n})$-frugal, $(\Delta + 1)$-coloring in $O(\log n)$ communication rounds.

Hind, Molloy, and Reed [10] show that any graph has a $O(\log^5 \Delta)$-frugal, $(\Delta+1)$ coloring, and also show a lower bound of $\Omega(\log \Delta / \log \log \Delta)$ on the frugality of any $(\Delta + 1)$-coloring. Thus our result improves on the Hind-Molloy-Reed upper bound and is "$\log \Delta$" away from being optimal[1]. Result (F1) is obtained via $O(\log \Delta)$ iterations of the randomized coloring procedure with symmetry-breaking and a proof that shows that in each iteration, each color is used $O(\log \Delta / \log \log \Delta)$ times in each neighborhood. In order to tightly control the coloring procedure, we also assign activation probabilities to vertices before each iteration; those vertices which are not activated, sleep through the iteration.

(F2) Let g be the maximum number of nodes at distance within $C_0 \log \Delta$ from any vertex in the given graph G, where C_0 is an absolute constant. Then G has a $(\Delta + 1)$-coloring that is t-frugal, where $t = O\left(\frac{\log g}{\log(\log g / \log \Delta)}\right)$.

This result subsumes the bound of (F1), since $g \leq \Delta^{O(\log \Delta)}$. Note that if $g \leq \Delta^{O(1)}$, then t is just $O(\log \Delta)$. This result is particularly applicable to "growth-bounded" graphs [8,21] which are graphs for which the number of nodes within distance r from any node grows much smaller than Δ^r (typically, as a polynomial in r). Thus, even for graphs with *exponential* growth (the number of nodes at a distance r can be poly$(\Delta, 2^r)$), we still obtain an $O(\log \Delta)$ bound on the frugality.

[1] In recent personal communication, Molloy and Reed have mentioned that they have proved the existence of an $O(\log \Delta / \log \log \Delta)$-frugal $(\Delta + 1)$-coloring; their paper is currently under preparation.

For this result, we again use iterative randomized coloring, but carefully exploit the fact that each iteration is *local* in the graph, and that there are only $O(\log \Delta)$ iterations: this is the reason for our definition of g going only up to distance $O(\log \Delta)$.

(F3) Let G be a *d-inductive* graph. Then G has a $(d+1)$-coloring that is t-frugal, where $t = O(\frac{\Delta}{d} \cdot \log^{1+\epsilon} \Delta)$ for any constant $\epsilon > 0$ whenever $d < 2\Delta/\log \Delta$, and $t = O(\frac{\Delta}{d} \cdot \frac{\log^2 \Delta}{\log(d \log \Delta/\Delta)})$ when $d \geq 2\Delta/\log \Delta$.

Recall that a graph is said to be *d-inductive* if there is an ordering $v_1, v_2, \ldots, v_n$ of its vertex set such that every vertex v_i has at most d neighbors in $\{v_1, v_2, \ldots, v_{i-1}\}$. For instance, planar graphs are 5-inductive. Any such ordering $v_1, v_2, \ldots, v_n$ is called a *d-inductive ordering*. A greedy algorithm that considers vertices of a d-inductive graph G in d-inductive order and assigns to each vertex the smallest available color, succeeds in producing a proper $(d+1)$-vertex coloring of G. Note that no $(d+1)$-coloring can have frugality better than $\Delta/(d+1)$ and therefore our result provides upper bounds that are within $O(\text{polylog}(\Delta))$ of the $\Delta/(d+1)$ lower bound. Also note that since any graph with maximum vertex degree Δ is Δ-inductive, our result (F1) on arbitrary graphs can be viewed as a special case of this result.

(F4) Let G be an n-vertex graph with maximum degree Δ and girth at least 5. Then G has an $O(\Delta/\log \Delta)$-coloring that is $O(\frac{\log^2 \Delta}{\log \log \Delta})$-frugal. Such a coloring can be computed in polynomial time. Furthermore, an $O(\Delta/\log \Delta)$-coloring that is $O(\log \Delta \cdot \frac{\log n}{\log \log n})$-frugal can be constructed by a distributed algorithm in $O(\log n)$ rounds of communication.

This result generalizes the result of Kim [18] and the result of Grable and Panconesi [7]. In 1948 Brooks showed that any connected graph with maximum degree Δ, with the exception of odd cycles and the $(\Delta+1)$-clique, has a Δ-coloring. In 1968 Vizing asked if this bound could be improved for "sparse graphs" of certain kinds — graphs with large girth, for example. This question remained unanswered until the work of [14] showed that any triangle-free graph has chromatic number $O(\Delta/\log \Delta)$. Building on this and a result due to [18] for graphs with girth at least 5, the paper [7] presents a simple randomized distributed algorithm that produces an $O(\Delta/\log \Delta)$-coloring of a triangle-free graph with high probability, in $O(\log n)$ communication rounds. This algorithm requires that $\Delta \geq \log^{1+\delta} n$ for any constant $\delta > 0$, whereas the existential result of [14] holds for any Δ. We extend the analysis of [7,18] to obtain (F4). We do not require our symmetry-breaking approach for the polynomial-time algorithm guaranteed by (F4).

Weighted Equitable Coloring. An *equitable coloring* of a graph is a proper vertex coloring such that the sizes of any two color classes differ by at most 1. If a k-coloring of an n-vertex graph G is equitable, then the size of every color class is in $\{\lceil n/k \rceil, \lfloor n/k \rfloor\}$. In a pivotal result, Hajnal and Szemerédi [9] showed that every graph G with maximum degree at most Δ has an equitable k-coloring for

every $k \geq \Delta + 1$. Recently, Kierstead and Kostochka [17] have presented a short proof of this result, along with a polynomial-time algorithm for computing such a coloring. Equitable colorings naturally arise in scheduling, partitioning, and load balancing [1,3,12,20,25,26]. Pemmaraju [24] and Janson and Ruciński [13] have used equitable colorings to derive large-deviation bounds for sums of random variables that exhibit limited dependence. Let $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ be a collection of bounded random variables with $X_i \in [0,1]$ for all i, let $S = \sum_i X_i$, and let $\mu = E[S]$. It is well-known that if the X_i's are independent then the following bounds on the tail probabilities of S, due to Chernoff [5], hold:

$$\Pr[S \geq (1+\varepsilon)\mu] \leq F^+(\mu,\varepsilon) \doteq \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{(1+\varepsilon)}}\right)^{\mu}; \quad \Pr[S \leq (1-\varepsilon)\mu] \leq F^-(\mu,\varepsilon) \doteq e^{-\mu\varepsilon^2/2}. \tag{3}$$

Researchers have attempted to extend the Chernoff bounds to situations where the X_i's exhibit limited dependence. We will model the "limited dependence" as usual by a *dependency graph* G of $\mathcal{X}$: G is an undirected graph with vertices $\{1,2,\ldots,n\}$ such that if $\{i_1, i_2, \ldots, i_\ell\}$ *is any independent set in* G*, then the random variables* $X_{i_1}, X_{i_2}, \ldots, X_{i_\ell}$ *are mutually independent.* If the X_i's have the *same* mean and G can be equitably k-colored, then the following tail-bounds hold [24]: $\Pr[S \geq (1+\varepsilon)\mu] \leq \frac{4k}{e} F^+(\mu,\varepsilon)^{1/k}$, and $\Pr[S \leq (1-\varepsilon)\mu] \leq \frac{4k}{e} F^-(\mu,\varepsilon)^{1/k}$, where $F^+(\mu,\varepsilon)$ and $F^-(\mu,\varepsilon)$ are from (3). Observe that if k, the palette size of the equitable coloring, is small then these bounds are quite good relative to the Chernoff bounds.

The above bounds apply only in the case where the X_i's have the same mean. To deal with the general case of arbitrary $X_i \in [0,1]$ using a similar approach, one needs to consider *weighted equitable colorings.* Let $G = (V = [n], E)$ be a vertex-weighted graph with weight function $w : V \to [0,1]$ such that $w(i) = E[X_i]$, and let $W = \mu = E[S] = \sum_{v \in V} w(v)$. Informally, a weighted equitable coloring of G is a proper k-coloring such that the total weight of **every** color class is "large", i.e., "not much smaller" than W/k. (A classical equitable coloring is the special case with unit weight for all vertices.) Call a k-coloring of G with λ being the minimum weight of all color classes, a (k,λ)*-coloring.* Using the approach of [13,24], such a coloring can be used to show that

$$\Pr[S \geq (1+\varepsilon)\mu] \leq k \cdot F^+(\lambda,\varepsilon); \quad \Pr[S \leq (1-\varepsilon)\mu] \leq k \cdot F^-(\lambda,\varepsilon). \tag{4}$$

Hence, given k, we aim for as large a λ as possible. We prove two results (E1), (E2) for $k = \Delta + 1$ colors, where Δ denotes the maximum degree of G. We prove result (E3) for d-inductive graphs with $k = d + 1$.

(E1) A $(\Delta+1,\lambda)$-coloring exists, with $\lambda \geq \left(1 - \frac{1}{e}\right)\frac{W}{\Delta+1} - O(\sqrt{W\log(\Delta+1)})$.

(E2) There is a constant $c > 0$ such that a $(\Delta+1,\lambda)$-coloring exists, with $\lambda \geq \left\lfloor \frac{cW}{\Delta\log(\Delta+1)} \right\rfloor$.

When $W > \Delta^{2+\Omega(1)}$, the lower bound in (E1) simplifies to $(1 - 1/e - o(1))\frac{W}{\Delta+1}$, which is to within a constant of the best possible. Result (E2) holds for all values of W relative to Δ and in particular holds for "small" W as well. As

shown by (4), these results yield bounds on the tail probabilities of sums of *arbitrary* bounded random variables. (E1) is obtained via a single iteration of randomized coloring with symmetry-breaking; (E2) is obtained by combining [9] and (E1) with a partitioning approach. We note that for a slightly smaller choice of c, we can also obtain (E2) by combining [9] and a second-moment analysis with partitioning; however, we are not aware of any other approach that yields our result (E1).

(E3) Every d-inductive graph has a $(d+1,\lambda)$-coloring with $\lambda \geq \frac{1}{e} \cdot \frac{W}{d+1} - O\left(\sqrt{\frac{W}{d+1}\Delta \log(d+1)}\right)$.

When $W > \Delta \cdot d^{1+\Omega(1)}$, the lower bound in (E3) simplifies to $(\frac{1}{e} - o(1))\frac{W}{d+1}$. Note that since a graph with maximum degree Δ is a Δ-inductive graph, result (E3) can be viewed as a generalization of result (E1), with the small change that the leading constant is $1/e$ instead of $1 - 1/e$. This result extends known results on classical equitable colorings of d-inductive graphs [4,19]. Bollobas and Guy [4] consider the equitable coloring of 1-inductive graphs (i.e., forests) whereas Kostochka et al. [19] consider d-inductive graphs for arbitrary d. Specifically, Kostochka et al. [19] show that every d-inductive graph has an equitable coloring with at most $16d$ colors (provided $\Delta < n/15$). Here, in result (E3), we relax the notion of "equitability" while requiring that exactly $d+1$ colors be used. Like (E1), result (E3) is also obtained via one iteration of the randomized coloring procedure, but with π being an arbitrary d-inductive vertex ordering.

Organization. Due to space constraints, we only present proofs of results (F1) and (E1) in this paper; these appear in the next two sections. For result (F1) we only present the existential proof, postponing the algorithmic results to the full version of the paper. We highlight the use of symmetry-breaking and Azuma's Inequality in our proofs.

2 Frugal Coloring for Arbitrary Graphs

In this section we prove result (F1). This follows by repeated application of the following result, that describes what happens in one iteration of randomized coloring procedure.

Theorem 1. *Let $G = (V, E)$ be a graph with maximum vertex degree Δ. Suppose that associated with each vertex $v \in V$, there is a palette $P(v)$ of colors, where $|P(v)| \geq deg(v) + 1$. Furthermore, suppose $|P(v)| \geq \Delta/4$ for all vertices v in G. Then, for some subset $C \subseteq V$, there is a list coloring of the vertices in C such that:*

(a) $G[C]$ is properly colored.
(b) For every vertex $v \in V$ and for every color x, there are at most $9 \cdot \frac{\log \Delta}{\log \log \Delta}$ neighbors of v colored x.
(c) For every vertex $v \in V$, the number of neighbors of v not in C is at most $\Delta(1 - \frac{1}{e^3}) + 27\sqrt{\Delta \log \Delta}$.

(d) *For every vertex $v \in V$, the number of neighbors of v in C is at most* $\frac{\Delta}{e^5} + 27\sqrt{\Delta \log \Delta}$.

Before we prove this theorem, we show how repeated applications of it yield result (F1), proving the existence of an $O\left(\frac{\log^2 \Delta}{\log\log \Delta}\right)$-frugal, $(\Delta+1)$-coloring of a graph G with maximum degree Δ. Start by associating the palette of colors $[\Delta+1]$ to each vertex. Letting $P_0(v)$ denote the initial palette of a vertex v, we have $P_0(v) = [\Delta+1]$ for all $v \in V$. Let $G_0 = G$. For each $i \geq 0$, we apply Theorem 1 to obtain a partial coloring of G_i. Let G_{i+1} denote the subgraph of G_i induced by vertices that are not colored in this partial coloring of G_i. The palette of colors $P_{i+1}(v)$ associated with a vertex v in G_{i+1} is obtained by deleting from $P_i(v)$ all colors used by neighbors of v in the partial coloring of G_i. Let $deg_i(v)$ denote the degree of vertex v in G_i. Let Δ_i denote the maximum vertex degree of G_i and let p_i denote the minimum palette size in G_i. Thus, $\Delta_0 = \Delta$ and $p_0 = \Delta + 1$. Note that initially the requirements of Theorem 1 are satisfied. Suppose that for some $i \geq 0$, the requirements of the theorem are satisfied. That is, (i) $|P_i(v)| \geq deg_i(v) + 1$ for all vertices v and (ii) $p_i \geq \Delta_i/4$. Since the palette of a vertex loses at most as many colors as neighbors that are colored, it is still true that $|P_{i+1}(v)| \geq deg_{i+1}(v) + 1$ for all v in G_{i+1}. Theorem 1(c) implies that $\Delta_{i+1} \leq \Delta_i(1 - \frac{1}{e^5}) + 27\sqrt{\Delta_i \log \Delta_i}$ and Theorem 1(d) implies that $p_{i+1} \geq p_i - \frac{\Delta_i}{e^5} - 27\sqrt{\Delta_i \log \Delta_i}$. Thus the worst case behavior of Δ_i and p_i is captured by the recurrences:

$$\Delta_{i+1} = \Delta_i\left(1 - \frac{1}{e^5}\right) + 27\sqrt{\Delta_i \log \Delta_i}; \qquad p_{i+1} = p_i - \frac{\Delta_i}{e^5} - 27\sqrt{\Delta_i \log \Delta_i}. \quad (5)$$

The above recurrences can be solved to obtain the following bounds on Δ_i and p_i.

Lemma 3. *Let $\alpha = (1 - 1/e^5)$. For all i for which $\Delta_i \geq 10^9$, $\Delta_i \leq 2\Delta_0 \cdot \alpha^i$ and $p_i \geq \frac{\Delta_0}{2}\alpha^i$.*

This implies that, provided p_{i+1} and Δ_{i+1} are large enough, it is the case that $p_{i+1} \geq \Delta_{i+1}/4$, thereby permitting the next application of the above theorem. To get a $(\Delta+1)$-coloring of the desired frugality, we repeatedly obtain partial colorings by applying the Theorem 1 until $\Delta_i < 10^9$. Given the rate of decay of Δ_i, letting $\alpha = (1 - 1/e^5)$, we see that at most $\log_{1/\alpha}\left(\frac{2\Delta_0}{10^9}\right) = O(\log \Delta)$ applications of the theorem are needed. Since the palette at every vertex has at least one more color than the number of neighbors of the vertex, a greedy list coloring algorithm will succeed in completing the coloring of the graph. Clearly, what we have constructed is a $(\Delta+1)$-proper vertex coloring of G. In each round i, each color appears in a neighborhood at most $9\log \Delta_i / \log\log \Delta_i = O(\log \Delta / \log\log \Delta)$ times. The final round adds only $O(1)$ copies of any color to any neighborhood, yielding a $(\Delta+1)$-coloring that is $O(\log^2 \Delta / \log\log \Delta)$-frugal. We have thus proved result (F1).

Proof of Theorem 1. We start by describing a randomized coloring procedure that will produce, with positive probability, a partial coloring of G with the four

desired properties. Let π be an arbitrary permutation of V. This establishes a ranking of the vertices. For any vertex v and color $x \in P(v)$, let $L(v,x)$ be the set of neighbors u of v such that u has a lower rank than v in π and u contains x in its palette $P(u)$. Each vertex v computes the quantity

$$q_v = \frac{1}{|P(v)|} \sum_{x \in P(v)} \prod_{u \in L(v,x)} \left(1 - \frac{1}{|P(u)|}\right).$$

This is the probability that no lower ranked neighbor of v will tentatively pick the color picked by v. Vertex v will use the value of q_v to determine its "sleep probability." For each vertex $v \in V$, independently pick a color $x \in P_v$ uniformly at random. We say that x is the *tentative color* of v. After picking a tentative color, v either goes off to sleep for the rest of this round or stays awake and attempts to make its tentative color permanent. Specifically, vertex v stays awake with probability $a_v = \frac{1}{q_v \cdot e^5}$ and it dozes off with probability $(1 - a_v)$. Later we will show that q_v is never smaller than $1/e^5$ and therefore $a_v \leq 1$. If v dozes off, then it remains uncolored at the end of the procedure. Note that v dozes off only after picking a tentative color and even though it may fall asleep, this choice of tentative color by v may have an influence on whether a neighbor gets permanently colored or not. If v stays awake, then it checks if there is a neighbor $u \in N(v)$ of smaller rank in π, that has the same tentative color as v. If no such u exists, then v is *permanently* colored x. The vertex subset C consists of all vertices that are permanently colored at the end of the procedure. The rest of the vertices are said to be *uncolored*.

For any vertex $v \in V$ and color $x \in C$, let $T_{v,x}$ be the indicator random variable that equals 1 if x is picked as v's tentative color. Note that $\Pr[T_{v,x} = 1] = 1/|P(v)|$ and therefore $E[T_{v,x}] = 1/|P(v)|$. For every $v \in V$ and $x \in C$, let $N_{v,x}$ be the random variable that equals the number of neighbors of v permanently colored x. Since a vertex has to be tentatively colored x before it can be permanently colored x, $N_{v,x} \leq \sum_{u \in N(v)} T_{u,x}$. By linearity of expectation, $E[\sum_{u \in N(v)} T_{u,x}] = \sum_{u \in N(v)} 1/|P(u)| \leq \frac{\Delta}{\Delta/4} = 4$. Note that the lower bound on the palette sizes plays a critical role here. Since the $T_{u,x}$ are mutually independent, we can use the Chernoff bound to show that $\sum_{u \in N(v)} T_{u,x}$ exceeds $9 \cdot \frac{\log \Delta}{\log \log \Delta}$ with probability less than $\frac{1}{\Delta^6}$. Therefore, for any vertex v and any color x,

$$\Pr\left[\text{Number of neighbors of } v \text{ colored } x \text{ exceeds } 9 \cdot \frac{\log \Delta}{\log \log \Delta}\right] < \frac{1}{\Delta^6}. \quad (6)$$

For any vertex $v \in V$, let R_v be the indicator random variable that equals 1 if v is permanently colored at the end of the coloring procedure. Note that v is permanently colored if v stayed awake and if no lower-ranked neighbor picked the same tentative color as vertex v did. Therefore, the probability that R_v equals 1, is $\Pr[R_v = 1] = a_v \cdot \frac{1}{|P(v)|} \cdot \sum_{x \in P(v)} \prod_{u \in L(v,x)} \left(1 - \frac{1}{|P(u)|}\right) = a_v \cdot q_v$. Recall that a_v was chosen to be $\frac{1}{q_v \cdot e}$ and therefore $\Pr[R_v = 1] = 1/e^5$. However, for the

"go to sleep" step of the coloring procedure to be well-defined, we need to show that $a_v \leq 1$. We do this by showing a lower bound of $1/e^5$ on q_v. Recall that q_v is the probability that no neighbor will choose the same tentative color as v. This probability is minimized when vertex v's palette is as small as possible, v has as many neighbors as possible, each of these neighbors have palettes that are as small as possible, and finally each of these palettes is identical to v's palette. Therefore,

$$q_v = \frac{1}{|P(v)|} \cdot \sum_{x \in P(v)} \prod_{u \in L(v,x)} \left(1 - \frac{1}{|P(u)|}\right) \geq \frac{1}{|P(v)|} \cdot \sum_{x \in P(v)} \left(1 - \frac{1}{\Delta/4}\right)^{\Delta} = \left(1 - \frac{4}{\Delta}\right)^{\Delta} \geq \frac{1}{e^5}.$$

Since $q_v \geq 1/e^5$, it follows that $a_v = 1/(q_v e^5) \leq 1$.

Let P_v denote the number of neighbors of v that are permanently colored by the procedure. Note that $P_v = \sum_{u \in N(v)} R_u$. Then by linearity of expectation, $E[P_v] = \sum_{u \in N(v)} E[R_u] = deg(v)/e^5$. Since the random variables R_u are not mutually independent for $u \in N(v)$, the Chernoff bound cannot be applied to show the concentration of P_v about its expectation. Instead, we apply Azuma's inequality in Lemma 4 to show the following concentration bound for P_v. The vertex-ordering imposed by the permutation π, will play a crucial role in this lemma.

Lemma 4

$$\Pr\left[\left|P_v - \frac{deg(v)}{e^5}\right| > 27\sqrt{\Delta \log \Delta}\right] < \frac{2}{\Delta^{4.5}}. \tag{7}$$

Proof. Let $U_v = \langle u_1, u_2, \ldots, u_m \rangle$ be the sequence of vertices at distance at most two from v, arranged in increasing rank according to π. Note that $m \leq \Delta^2$. Let S_i indicate whether vertex u_i has decided to go to sleep or not and let C_i denote the tentative color choice of vertex u_i. Let $T_i = (S_i, C_i)$. Clearly, P_v is completely determined by the trials $T_1, T_2, \ldots, T_m$. Referring to Azuma's Inequality, let D denote the absolute difference in conditional expectation $\Big|E[P_v|T_1 = t_1, \ldots, T_i = t_i] - E[P_v|T_1 = t_1, \ldots, T_i = t_i']\Big|$. Let $t_i = (s_i, c_i)$ and $t_i' = (s_i', c_i')$. Provided u_i is a neighbor of v, the difference between t_i and t_i' may contribute at most 1 to the above difference. Any other contribution to this difference is due to vertices u_j such that (i) $j > i$, (ii) u_j is a neighbor of both u_i and v, and (iii) u_j picks as a tentative color either c_i or c_i'. To make this more precise, let S_i be the set $\{u_j \mid j > i$ and u_j is adjacent to both u_i and $v\}$. Thus, S_i is the set of vertices satisfying conditions (i) and (ii) above. For any $u_j \in S_i$, the probability that u_j picks c_i or c_i' as its tentative color is $2/|P(u_j)|$. Note that u_j has to pick one or the other color to many any contribution to D. The quantity $2/|P(u_j)|$ is bounded above by $8/\Delta$, since $|P(u_j)| \geq \Delta/4$. Therefore, the expected contribution of u_j to the quantity D is at most $8/\Delta$. So, the expected contribution of all of S_i to D is bounded above by $8|S_i|/\Delta$. Referring to Azuma's Inequality, we can therefore take $c_i = 1 + 8|S_i|/\Delta$ for each $i : u_i \in N(v)$ and $c_i = 8|S_i|/\Delta$ for all other i. Now note that since S_i is a subset of the set of neighbors of v, $|S_i| \leq \Delta$ for every i. Also, $\sum_{i=1}^{m} |S_i|$ is bounded above by Δ^2. This is because for every vertex

$u_j \in S_i$, there is a unique corresponding edge $\{u_i, u_j\}$ in the graph G. Noting that there are at most Δ^2 edges incident on neighbors of v, we get the upper bound on $\sum_{i=1}^{m} |S_i|$. The two inequalities, $|S_i| \leq \Delta$ and $\sum_{i=1}^{m} |S_i| \leq \Delta^2$ together imply that $\sum_{i=1}^{m} |S_i|^2 \leq \Delta^3$. Hence,

$$\sum_{i=1}^{m} c_i^2 = \sum_{i:u_i \in N(v)} \left(1 + \frac{8|S_i|}{\Delta}\right)^2 + \sum_{i:u_i \notin N(v)} \left(\frac{8|S_i|}{\Delta}\right)^2$$
$$\leq \frac{64}{\Delta^2} \cdot \sum_{i=1}^{m} |S_i|^2 + \frac{16}{\Delta} \cdot \sum_{i=1}^{m} |S_i| + \Delta \leq 81\Delta$$

Finally, plugging the value $t = 27 \cdot \sqrt{\Delta \log \Delta}$ and $\sum_{i=1}^{m} c_i^2 \leq 81\Delta$ into Azuma's Inequality (see (2)), we get the desired result.

Let B_v denote the "bad event" that for some color x, vertex v has more than $9\frac{\log \Delta}{\log \log \Delta}$ neighbors colored x *or* $|P_v - \frac{deg(v)}{e^5}| > 27\sqrt{\Delta \log \Delta}$. Using Inequalities (6), (7), and the union bound we get that $\Pr[B_v] < (\Delta{+}1)/\Delta^6 + 2/\Delta^{4.5} \leq 4/\Delta^{4.5}$. If vertices u and v are more than four hops away from each other in G, then B_u and B_v are mutually independent. Therefore, B_v is independent of all except at most Δ^4 other bad events. Applying the Lovász Local Lemma (see Lemma 2) yields the theorem.

Remark on constants. In this section we have explicitly specified constants so that a careful reader may verify our calculations completely. However, we have not attempted to optimize these constants. For example, it is possible to show that Δ_i and p_i decay at a rate of $(1 - \frac{1}{e^{1+\epsilon}})$ (rather than $(1 - \frac{1}{e^5})$) for any $\epsilon > 0$. The choice of ϵ here affects various constants in the proof including the constant in the asymptotic notation used to specify the frugality of the coloring.

3 Weighted Almost-Equitable Colorings

In this section we prove result (E1). Let $G = (V = [n], E)$ be a vertex-weighted graph with weight function $w : V \to [0, 1]$. Let π be a permutation of $(1, 2, \ldots, n)$ picked uniformly at random from the set of all permutations of $(1, 2, \ldots, n)$. Run one round of our random coloring procedure described in Section 2, but without any nodes falling asleep. For any color x and for any vertex v let $P(v, x)$ denote the event that v is (permanently) colored x, at the end of one round of the random coloring procedure.

Lemma 5. $\Pr[P(v, x)] \geq \left(1 - \frac{1}{e}\right) \frac{1}{\Delta+1}$.

Proof. Let $L(v)$ be the subset of neighbors of vertex v that are ranked before v by π. Then, $\Pr[P(v, x)] = \frac{1}{\Delta+1} \sum_{j=0}^{deg(v)} \frac{1}{deg(v)+1} \cdot \left(1 - \frac{1}{(\Delta+1)}\right)^j$. In this expression, the term "$\frac{1}{(\Delta+1)}$" is the probability that color x is tentatively picked by v, the summation is over all possible sizes of $L(v)$, the term "$\frac{1}{deg(v)+1}$" is the probability

that exactly j of the neighbors of v are ranked below v in π, and the term "$(1-\frac{1}{(\Delta+1)})^j$" is the probability that none of these j neighbors that are ranked below v, pick x as their tentative color. This expression can be simplified to yield

$$\Pr[P(v,x)] = \frac{1}{deg(v)+1}\left(1-\left(1-\frac{1}{(\Delta+1)}\right)^{deg(v)+1}\right).$$

Noting that the r.h.s. above achieves the minimum value at $deg(v) = \Delta$, we obtain

$$\Pr[P(v,x)] \geq \frac{1}{\Delta+1}\left(1-\left(1-\frac{1}{(\Delta+1)}\right)^{\Delta+1}\right) \geq \frac{1}{\Delta+1}\cdot\left(1-\frac{1}{e}\right).$$

The above lemma implies that the expected weight of any color class is at least $(1-1/e)\cdot W/(\Delta+1)$. We now show that the weight of any color class is concentrated around its expectation and in particular the weight of a color class is much smaller than $(1-1/e)\cdot W/(\Delta+1)$ only with small probability. As usual, we employ Azuma's Martingale inequality to prove this result; again the symmetry-breaking approach plays a critical role.

Lemma 6. *Let $x \in [\Delta+1]$ and let W_x denote the total weight of vertices colored x. Then for any fixed $c > 0$,*

$$\Pr\left[W_x < \left(1-\frac{1}{e}\right)\cdot\frac{W}{(\Delta+1)} - c\sqrt{W\log(\Delta+1)}\right] \leq \frac{1}{(\Delta+1)^{c^2/18}}.$$

Proof. Let T_i be the tentative choice of a color by the vertex of rank i in π. W_x is completely determined by the outcomes of the trials $T_1, T_2, \ldots, T_n$. Now consider the difference in conditional expectations from Azuma's Martingale Inequality:

$$c_i = \Big|E[W_x \mid T_1 = t_1, T_2 = t_2, \ldots, T_i = t_i] - E[W_x \mid T_1 = t_1, T_2 = t_2, \ldots, T_i = t_i']\Big|.$$

Let v be the vertex of rank i in π. The above difference is at most

$$c_i \leq w(v) + \frac{2}{\Delta+1}\sum_{u\in U(v)} w(u),$$

where $U(v)$ is the set of neighbors of v ranked higher than v in π. The first term "$w(v)$" in the above bound is due to the change in the tentative color of v from t_i to t_i'. The second term is the *expected* change in W_x; note that this occurs only at vertices in $U(v)$. Using the fact that every vertex weight is in $[0,1]$, we get that $c_i \leq 1 + 2\Delta/(\Delta+1) < 3$. Also,

$$\sum_{i=1}^{n} c_i \leq \sum_{v\in V} w(v) + \frac{2}{\Delta+1}\sum_{v\in V}\sum_{u\in U(v)} w(u) \leq W + \frac{2\Delta}{\Delta+1}\cdot\sum_{v\in V} w(v) < 3W.$$

Finally, $\sum_{i=1}^{n} c_i^2 < 3\sum_{i=1}^{n} c_i < 9W$. Therefore, the bound in Azuma's Inequality simplifies to

$$\exp\left(\frac{-t^2}{2\sum_{i=1}^{n} c_i^2}\right) \leq \exp\left(\frac{-c^2 W \log(\Delta+1)}{18W}\right) = \frac{1}{(\Delta+1)^{c^2/18}}.$$

From this we get, using Azuma's Inequality,

$$\Pr\left[W_x < \left(1-\frac{1}{e}\right)\cdot\frac{W}{\Delta+1} - c\cdot\sqrt{W\log(\Delta+1)}\right] < \frac{1}{(\Delta+1)^{c^2/18}}.$$

Since we have $\Delta+1$ color classes, using the union bound we get that

$$\Pr\left[\exists x \in [\Delta+1] : W_x < \left(1-\frac{1}{e}\right)\cdot\frac{W}{\Delta+1} - c\cdot\sqrt{W\log(\Delta+1)}\right] \leq \frac{(\Delta+1)}{(\Delta+1)^{c^2/18}}.$$

Choosing $c \geq 5$ guarantees that with positive probability, for every color $x \in [\Delta+1]$, the weight W_x of the vertices colored x is at least $\left(1-\frac{1}{e}\right)\cdot\frac{W}{\Delta+1} - c\cdot\sqrt{W\log(\Delta+1)}$. This proves result (E1).

Acknowledgment. We thank Mike Molloy for his helpful comments and encouragement, and the referees for their comments that helped improve the paper.

References

1. Baker, B., Coffman, E.: Mutual exclusion scheduling. Theor. Comput. Sci. 162, 225–243 (1996)
2. Beck, J.: An algorithmic approach to the Lovász Local Lemma. Random Structures and Algorithms 2, 343–365 (1991)
3. Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J.: Scheduling computer and manufacturing processes, 2nd edn., p. 485. Springer, Berlin (2001)
4. Bollobás, B., Guy, R.K.: Equitable and proportional coloring of trees. Journal of Combinatorial Theory, Series B 34, 177–186 (1983)
5. Chernoff, H.: A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. American Statistical Association Journal 58, 13–30 (1963)
6. Erdős, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. In: Hajnal, A., et al. (eds.) Infinite and Finite. Colloq. Math. Soc. J. Bolyai, vol. 11, pp. 609–627. North Holland, Amsterdam (1975)
7. Grable, D.A., Panconesi, A.: Fast distributed algorithms for Brooks-Vizing colorings. Journal of Algorithms 37, 85–120 (2000)
8. Gupta, A., Krauthgamer, R., Lee, J.R.: Bounded geometries, fractals, and low-distortion embeddings. In: Proc. IEEE Symposium on Foundations of Computer Science (2003)
9. Hajnal, A., Szemerédi, E.: Proof of a conjecture of Erdös. In: Erdös, P., Rényi, A., Sós, V.T. (eds.) Combinatorial Theory and its Applications, vol. II, pp. 601–603. North-Holland, Amsterdam (1970)
10. Hind, H., Molloy, M., Reed, B.: Total colouring with Δ+polylog(Δ) colours. SIAM Journal on Computing 28, 816–821 (1998)

11. Hou, Y.T., Kumar, V.S.A., Marathe, M.V., Srinivasan, A.: Personal communication (2006)
12. Irani, S., Leung, V.: Scheduling with conflicts, and applications to traffic signal control. In: Proceedings of the 7th Annual ACM-SIAM symposium on discrete algorithms, held in Atlanta, GA, pp. 85–94. SIAM, Philadelphia (1996)
13. Janson, S., Ruciński, A.: The infamous upper tail. Random Structures and Algorithms 20, 317–342 (2002)
14. Johansson, A.: Asymptotic choice number for triangle free graphs. In DIMACS Technical Report, 91-5 (1996)
15. Johansson, Ö.: Simple distributed $\Delta+1$-coloring of graphs. Information Processing Letters 70, 229–232 (1999)
16. Kahn, J.: Asymptotically good list-colorings. J. Combinatorial Theory, Series A 73, 1–59 (1996)
17. Kierstead, H.A., Kostochka, A.V.: A Short Proof of the Hajnal-Szemeredi Theorem on Equitable Coloring. Combinatorics, Probability, and Computing (to appear)
18. Kim, J.H.: On Brooks' Theorem for Sparse Graphs. Combinatorics, Probability, and Computing 4, 97–132 (1995)
19. Kostochka, A.V., Nakprasit, K., Pemmaraju, S.V.: On Equitable Coloring of d-Degenerate Graphs. SIAM J. Discret. Math. 19(1), 83–95 (2005)
20. Krarup, J., de Werra, D.: Chromatic optimisation: Limitations, objectives, uses, references. Eur. J. Oper. Res. 11, 1–19 (1982)
21. Krauthgamer, R., Lee, J.R.: Navigating nets: simple algorithms for proximity search. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (2004)
22. Luby, M.: Removing randomness in parallel computation without a processor penalty. Journal of Computer and System Sciences 47(2), 250–286 (1993)
23. Molloy, M., Reed, B.: Graph Colouring and the Probabilistic Method. Springer, Heidelberg (2000)
24. Pemmaraju, S.V.: Equitable colorings extend Chernoff-Hoeffding bounds. In: Proceedings of the 5th International Workshop on Randomization and Approximation Techniques in Computer Science (APPROX-RANDOM 2001), pp. 285–296 (2001)
25. Smith, B.F., Bjorstad, P.E., Gropp, W.D.: Domain decomposition. Parallel multilevel methods for elliptic partial differential equations, p. 224. Cambridge University Press, Cambridge (1996)
26. Tucker, A.: Perfect graphs and an application to optimizing municipal services. SIAM Review 15, 585–590 (1973)

The Local Nature of List Colorings for Graphs of High Girth*

Flavio Chierichetti[1] and Andrea Vattani[2]

[1] CS Department, Sapienza University of Rome
chierichetti@di.uniroma1.it
[2] CSE Department, University of California San Diego
avattani@ucsd.edu

Abstract. We consider list coloring problems for graphs $\mathcal{G}$ of girth larger than $c \log_{\Delta-1} n$, where n and $\Delta \geq 3$ are, respectively, the order and the maximum degree of $\mathcal{G}$, and c is a suitable constant. First, we determine that the edge and total list chromatic numbers of these graphs are $\chi'_l(\mathcal{G}) = \Delta$ and $\chi''_l(\mathcal{G}) = \Delta + 1$. This proves that the general conjectures of Bollobás and Harris (1985), Behzad and Vizing (1969) and Juvan, Mohar and Škrekovski (1998) hold for this particular class of graphs.

Moreover, our proofs exhibit a certain degree of "locality", which we exploit to obtain an efficient distributed algorithm able to compute both kinds of optimal list colorings.

Also, using an argument similar to one of Erdös, we show that our algorithm can compute k-list vertex colorings of graphs having girth larger than $c \log_{k-1} n$.

1 Introduction

Graph coloring is a fundamental problem in computer science and combinatorics. Applications arise in many different areas, such as networks, resource allocations and VLSI design. For many coloring problems, though, no efficient algorithms are known to exist. This led to considering restrictions of coloring problems to special classes of graphs.

In this paper we consider (list) edge, total and vertex colorings for graphs of high girth (that is, for graphs having no small cycles), with special emphasis on distributed algorithms. In the list edge coloring problem, we have a graph $\mathcal{G}$ and, for each of its edges e, a list of colors available for e. The goal is to compute a proper coloring using colors from the lists, where a coloring is proper if adjacent edges have different colors. The list edge chromatic number (or list chromatic index) of $\mathcal{G}$, denoted $\chi'_l(\mathcal{G})$, equals the minimum integer t such that, for each possible assignment of lists of t colors to the edges of $\mathcal{G}$, a proper coloring of $\mathcal{G}$ exists. The edge chromatic number (or chromatic index) $\chi'(\mathcal{G})$ of $\mathcal{G}$ is the minimum integer t such that, if all the edges of $\mathcal{G}$ are assigned the same list of t colors, a proper coloring of $\mathcal{G}$ exists.

* This work was partially supported by a grant of Yahoo! Research and by the MIUR PRIN Project "Web Ram: web retrieval and mining". The main part of this work was carried out at the Computer Science Department of Sapienza University of Rome.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 320–332, 2008.

The list vertex coloring problem, and the list and non-list vertex chromatic numbers $\chi_l(\mathcal{G}), \chi(\mathcal{G})$, are analogous, except that in this case the vertices are the elements of the graph to be properly colored. In the list total coloring problem both vertices and edges have lists, and the problem is to color the graph in such a way that any two adjacent or incident objects, whether edges or vertices, have different colors. We use $\chi''(\mathcal{G})$ and $\chi''_l(\mathcal{G})$ to denote, respectively, the total chromatic number and the list total chromatic number of $\mathcal{G}$.

In this paper we show that, if $\mathcal{G}$ is a graph with n vertices, maximum degree $\Delta \notin \{1, 2\}$, and girth at least $c \log_{\Delta-1} n$ (for a suitable constant c), then $\chi'(\mathcal{G}) = \chi'_l(\mathcal{G}) = \Delta$ and $\chi''(\mathcal{G}) = \chi''_l(\mathcal{G}) = \Delta + 1$.

Our constructive proofs highlight a local property of the coloring operations, that we use to obtain such optimal colorings efficiently in a distributed setting.

From an existential point of view, our results settle, for the case of high girth graphs, a conjecture of Bollobás and Harris [3] (that states $\chi'(G) = \chi'_l(G)$), a conjecture of Behzad and Vizing [2] ($\chi''(G) \leq \Delta + 2$) and a related conjecture of Juvan, Mohar, Škrekovski [11] ($\chi''(G) = \chi''_l(G)$), all of which were formulated for general graphs. The first two conjectures date back to more than two decades ago, while the third one is more recent.

Also, by extending an argument of Erdös [7], we show that list vertex colorings can be obtained efficiently by a distributed algorithm. In this case, though, the number of colors used may not be optimal.

1.1 Main Results

We now state more precisely our results. Recall that no chromatic number is greater than its list counterpart.

Theorem 1. *If $\mathcal{G}$ is a graph with n nodes, $\Delta(\mathcal{G}) \neq 2$, having girth $g(\mathcal{G}) > 4\lceil \log_{\Delta-1} n \rceil + 1$, then $\chi'_l(\mathcal{G}) = \Delta(\mathcal{G})$.*

Theorem 2. *If $\mathcal{G}$ is a graph with n nodes, $\Delta(\mathcal{G}) \notin \{1, 2\}$, having girth $g(\mathcal{G}) > 4\lceil \log_{\lceil \frac{\Delta}{2} \rceil} n \rceil + 3$, then $\chi''_l(\mathcal{G}) = \Delta + 1$.*

These theorems imply that, if the girth of a graph is high enough and the graph is not a collection of paths and cycles, then that graph is both Class-1 and Type-1 (a graph $\mathcal{G}$ is Class-1 if $\chi'(\mathcal{G}) = \Delta$ and it is Type-1 if $\chi''(\mathcal{G}) = \Delta + 1$).

The constraint $\Delta \neq 2$ is necessary for trivial reasons, since any cycle C_n has $g(C_n) = n$, $\Delta(C_n) = 2$, and (a) if $n \bmod 2 \neq 0$ then $\chi'(C_n) = \Delta + 1$, while (b) if $n \bmod 3 \neq 0$ then $\chi''(C_n) = \Delta + 2$. For the total chromatic number even the requirement $\Delta \neq 1$ is needed as any matching is Type-2 (a graph $\mathcal{G}$ is Type-2 if $\chi''(\mathcal{G}) \geq \Delta + 2$).

As for the girth requirement, we show the existence of an infinite family of Class-2 graphs (a graph $\mathcal{G}$ is Class-2 if $\chi'(\mathcal{G}) = \Delta + 1$) having a girth three times smaller than the one required in Thm. 1.

Proposition 1. *For each element of an infinite sequence of increasing degrees $\{\Delta_i\}_{i=1}^{\infty}$, there exists an infinite family of graphs $\{\mathcal{G}_j(\Delta_i)\}_{j=1}^{\infty}$ of maximum degree Δ_i and increasing order n_j, such that $g(\mathcal{G}_j(\Delta_i)) \geq \frac{4}{3} \log_{\Delta_i - 1} n_j - O(1)$ and $\chi'(\mathcal{G}_j(\Delta_i)) = \Delta_i + 1$.*

On the other hand, we were not able to obtain an infinite family of Type-2 graphs having reasonably large girth.

Erdös [7] gave an upper bound on the vertex chromatic number of graphs of high girth. His argument can be extended to show the following:

Theorem 3. *Let $k \geq 3$. If $\mathcal{G}$ is a graph with n nodes and $g(\mathcal{G}) > 2\lceil \log_{k-1} n \rceil$, then $\chi_l(\mathcal{G}) \leq k$.*

The proofs of theorems 2 and 3 are omitted from this extended abstract for lack of space.

1.2 Algorithmic Consequences

An interesting feature of our method is that it illustrates a certain local nature of list colorings for high girth graphs.For instance, for list edge colorings we show the following. Assume that the whole graph $\mathcal{G}$ is colored except for one last edge e. Then, to color e it is enough to re-color a neighborhood of e of radius $O(\log n)$. Analogous properties hold for list total and list vertex colorings. These properties lead to efficient distributed implementations of our algorithms.

Theorem 4. *The list colorings of Theorems 1, 2, 3 can be computed in $O(\log^3 n)$-many communication rounds, in the synchronous, message passing model of computation.*

All the three kinds of colorings will be computed by the same algorithm.

We remark that this algorithm can be simulated sequentially in polynomial-time, as every node in the network only performs polynomially many operations in every round of the protocol.

2 Related work

A well-known result of Vizing [19] shows that the chromatic index $\chi'(\mathcal{G})$ of any graph $\mathcal{G}$ of maximum degree Δ is either Δ or $\Delta + 1$. A conjecture of Bollobás and Harris [3] states that the list chromatic index $\chi'_l(\mathcal{G})$ is equal to $\chi'(\mathcal{G})$. For total coloring, a conjecture independently suggested by Behzad [2] and Vizing states that $\chi''(\mathcal{G}) \leq \Delta + 2$. A more recent conjecture by Juvan, Mohar, Škrekovski [11] states that the list total chromatic number $\chi''_l(\mathcal{G})$ equals $\chi''(\mathcal{G})$.

On the other hand, it is known [8] that the gap between the chromatic number $\chi(\mathcal{G})$ and the list chromatic number $\chi_l(\mathcal{G})$ of a graph can be logarithmic in its order.

There has been a lot of work in trying to prove the first two conjectures. For arbitrary graphs $\mathcal{G}$, Kahn [12] proved that the list chromatic index is $\chi'_l(\mathcal{G}) \leq (1 + o(1))\Delta$, while Molloy and Reed [17] proved that the the total chromatic number is $\chi''(\mathcal{G}) \leq \Delta + c$, for some (rather large) constant c. Exact values are known only for special classes of graphs; we now comment on these kinds of results as they are more directly related to ours.

The relationship between girth and list edge chromatic number has been studied in [13], where it is shown that, if $g(\mathcal{G}) \geq 8\Delta(\ln \Delta + 1.1)$, then $\chi'_l(\mathcal{G}) \leq \Delta + 1$ (and thus $\chi''_l(\mathcal{G}) \leq \Delta + 3$). Our requirement on the girth is less stringent for large enough

Δ (e.g. already for Δ logarithmic in n). Note that, even for smaller Δ, we establish a better bound of Δ (resp. $\Delta + 1$) for the list edge (resp., total) chromatic number.

Another approach uses the concept of *degeneracy* of a graph, i.e. the maximum smallest degree of its subgraphs. Vizing showed (see for instance [10]) that any graph $\mathcal{G}$ with degeneracy $\leq \Delta/2$ belongs to Class-1. One can show that for high enough $\Delta(\mathcal{G})$, the degeneracy of our graphs (that is, graphs with girth as high as we need) is small enough for Vizing's result to hold. On the other hand, for small maximum degrees, there are graphs that satisfy our requirement and not Vizing's (for instance, take two cycles intersecting on a single edge).

In [9,20], the authors give parallel algorithms for computing Δ and $\Delta + 1$ edge and total colorings of graphs of small degeneracy. Their results are not directly comparable to ours. While their requirement is weaker than ours for large enough Δ, there exist graphs of small degree that satisfy our requirement and not theirs. The main difference, however, is that our results hold for the more general case of *list* colorings, as opposed to non-list ones. Also, our method leads to efficient distributed algorithms, while their parallel algorithms do not seem to be efficiently distributable.

Borodin et al. attacked the problem from another point of view. The maximum average degree (MAD) of a graph is the maximum of the average degrees of its subgraphs. In [4], they show that, if $\Delta(\mathcal{G}) \geq 4$ and the MAD is "small enough", then $\chi_l'(\mathcal{G}) = \Delta(\mathcal{G})$ and $\chi_l''(\mathcal{G}) = \Delta(\mathcal{G}) + 1$. The result extends to list chromatic index for $\Delta(\mathcal{G}) = 3$. The relationship between this result and that in the present paper is unclear and intriguing. Let $m(g, n)$ be the maximum number of edges of graphs having girth g and order n, and let $M(g, n) := n^{1+1/\lfloor (g-1)/2 \rfloor}$. A well-known bound states that $m(g, n) \leq M(g, n)$, and improving this is known to be a challenging open problem (see for instance [16]). It can be shown that if our result is subsumed by that of [4] then a sharper bound holds for $m(g, n)$ at least for the girths we require. More precisely, for these girths, the bound would have to be improved non trivially, by at least a $\frac{1}{\sqrt{2}}$ factor. Be as it may, our proof is conceptually different and it highlights an interesting local property of list colorings that leads directly to efficient distributed algorithms. Our result holds even for $\Delta(\mathcal{G}) = 3$ in the case of list total coloring.

As for vertex colorings, it was shown by Erdös [7] that graphs of high enough girth have small chromatic number — his proof can be modified to give an upper bound on their *list* chromatic number; our distributed algorithm can color the vertices of these graphs using a number of colors equal to that upper bound.

The distributed (non list) edge coloring problem has been the object of a lot of study (see [5,6,18] and references therein). All the previous works we are aware of considered the edge coloring problem for general graphs, obtaining suboptimal colorings.

3 List Edge Coloring

In this section we prove Theorem 1. Here we are interested in the existential result deferring the algorithmic discussion to a later section. The idea of the proof is as follows. Suppose by induction that we have list-colored the entire graph except for one last edge $e = uv$. The following local property holds. No matter how $\mathcal{G} - e$ is colored, it is always possible to reassign the colors inside a neighborhood of e of radius $O(\log n)$ in such a

way that there will be a free color for e, drawn from e's list. More precisely, the neighborhood to be re-colored consists of two disjoint BFS trees, each of which is rooted at one of the two endpoints of e. The basis of the induction is trivial, since we can start with any edge and assign it any color from its list.

The local nature of the re-coloring operation will be later exploited to show that such list-colorings can be obtained by means of efficient distributed algorithms.

The above discussion motivates the following definitions. From now on, let $\mathcal{G}$ be the graph we are list coloring and let $\Delta = \Delta(\mathcal{G})$ denote its maximum degree. Henceforth, we will use the term coloring to mean list coloring.

Definition 1. *A Δ-tree is a rooted tree of maximum degree at most Δ whose leaves are all at the same distance from the root. Furthermore, the degree of the root is $< \Delta$.*

The intuition that drives this and the following definitions is that, after removing $e = uv$ from $\mathcal{G}$, we want to consider two BFS trees $T(u)$ and $T(v)$ starting from e's endpoints and show that they can always be re-colored in such a way that there will be a free color for e, regardless of how $\mathcal{G} - e$ is colored initially. The degree of the roots is $< \Delta$ by the removal of e. Intuitively, we do not consider leaves above the lowest level as they are not affected by the rest of the graph's coloring.

The next definition captures the idea of a tree T whose set of possible colorings is constrained by the coloring of $\mathcal{G} - e$. Henceforth, we will denote by $T(u)$ a tree that is rooted at u.

Definition 2. *Let $T(r)$ be a Δ-tree. G is an* augmentation *of $T(r)$ if it is obtained from $T(r)$ by adding edges and paths of length two connecting only leaves of $T(r)$. Furthermore, it must be $\Delta(G) = \Delta$ and $\deg_G(r) < \Delta$.*

The constraints on a Δ-tree T given by the list coloring of $\mathcal{G} - e$ can be succinctly expressed by coloring the edges of an augmentation G of T.

Definition 3. *Let T be a Δ-tree. Given a Δ-list assignment $\mathcal{L}$ to $E(T)$, an augmentation G of T and a coloring γ of G, we say that the triple $(\mathcal{L}, G, \gamma)$ is* legal *if γ is a proper coloring of G that agrees with $\mathcal{L}$.*

Note that every Δ-tree has at least a legal triple, say, the identical list-assignment, the trivial augmentation $G = T$, and any of its proper colorings. We now define a notion of "freedom" of Δ-trees. Intuitively, a tree T is t-free if, regardless of how $\mathcal{G} - e$ is colored, we can always re-color it in such a way that t colors become available at the root.

Definition 4. *Let $T(r)$ be a Δ-tree. $T(r)$ is* at least t-free *if, for each list L of Δ colors and each legal triple $(\mathcal{L}, G, \gamma)$, there exists some set $C \subseteq L$ of t colors such that for all $c \in C$, there exists a proper coloring γ_c of G such that*

- *$\gamma_c(e) = \gamma(e)$, for all $e \in E(G) - E(T(r))$,*
- *$\gamma_c(e) \in \mathcal{L}(e)$, for all $e \in E(T(r))$, and*
- *γ_c do not assign the color c to any edge incident to r.*

We now show some basic properties of Δ-trees that will be used later in the proofs.

- Each Δ-tree is at least 1-free, as for each of its legal triples, using the coloring of the triple, at most $\Delta - 1$ colors will be unavailable at the root, thus at least a color will remain in any list L of cardinality Δ.
- No Δ-tree is at least $(\Delta + 1)$-free (again by $|L| = \Delta$).

A tree is exactly t-free (or, simply, t-free) if it is at least t-free, but not at least $(t + 1)$-free. If a tree is t-free we say that it has t degrees of freedom.

Lemma 1. *Let T be a Δ-tree T that is exactly t-free, let L be a set of Δ colors, and $C \in \binom{L}{t}$. Then, there exists a legal triple $(\mathcal{L}, G, \gamma)$ such that, for all and only $c \in C$, there exists a proper coloring γ_c of G that satisfies*

- $\gamma_c(e) = \gamma(e)$, *for all* $e \in E(G) - E(T)$,
- $\gamma_c(e) \in \mathcal{L}(e)$, *for all* $e \in E(T)$, *and*
- γ_c *do not assign the color* c *to any edge incident to* r.

Proof. Since T is not at least $(t + 1)$-free, there exists a legal triple R which does not leave $t+1$ colors available at the root. But since T is at least t-free, all legal triples allow the choice of t colors at the root. So R allows exactly t colors. We can obtain all possible sets $C \subseteq L$ of t colors from R just by renaming the colors of the coloring of R. □

Observation 1. *If the root of a Δ-tree T has exactly k children and one of them is at least $(k + 1)$-free, the degree of freedom of T does not change if that child is deleted.*

Proof. Let u be that child and let $T(u)$ denote its subtree. By deleting u the degree of freedom does not decrease. To see that it does not increase either, let T' be the tree obtained by removing $T(u)$ from T. Take any color c available at the root of T'. We show that c is also available at the root of T. Color T' as to have c available at the root; also color all the edges of T, excluding those of $T(u)$, in the same manner. Now the only uncolored edge incident to the root has at least 2 colors available, for $T(u)$ is at least $(k + 1)$-free and we used at most $k - 1$ colors for the other edges incident to the root. Thus, we can choose a color other than c to color the edge connecting u to the root to complete the coloring, that is c is available for T. □

Observation 2. *Each minimum t-free tree $T(r)$ has $\Delta - t$ children.*

Proof. If $T = T(r)$ has less than $\Delta - t$ children, then it is necessarily more than t-free, for less than $\Delta - t$ colors are blocked at its root.

To show that $\Delta - t$ is also an upper bound, let r (the root of T) have $k \geq \Delta - t$ children. Let $T_1, \ldots, T_k$ be their corresponding subtrees.

By the minimality of T, observation 1 cannot be applied to it. That is, each tree T_i $(1 \leq i \leq k)$ is at most k-free.

By lemma 1, given any set C of k colors, for all $1 \leq i \leq k$, there exists an augmentation of T_i such that in every proper coloring of T_i the colors available at its root are a subset of C.

These augmentations of the T_i's, taken together, constitute an augmentation for T that forces every edge incident to r to take a color from C. Since there are k such edges and $|C| = k$, the set of colors of those edges must be C in any proper coloring. So exactly $\Delta - k$ colors are available at the root of T. That is, T is at most $(\Delta - k)$-free. Since $k \geq \Delta - t$, this can only be true for $k = \Delta - t$. □

The next definition is pivotal.

Definition 5. *Let $\mathcal{T}_t^h$ be the set of t-free Δ-trees of height h. Also, let n_t^h be the order of any smallest t-free tree in $\mathcal{T}_t^h$ (or ∞ if that set is empty).*

Recall our goal: we start with a list coloring of $\mathcal{G} - e$, $e = uv$, and grow two BFS trees $T(u)$ and $T(v)$ with the aim of showing that they can be re-colored in such a way that (a) the coloring in $\mathcal{G} - (T(u) \cup T(v))$ remains unchanged and (b) there is an available color for e. We will do this by showing that if the height of $T(u)$ and $T(v)$ is large enough then they both are at least $\left(\left\lceil \frac{\Delta}{2} \right\rceil + 1\right)$-free and hence there is at least one spare color for e to complete the coloring. In what follows we will characterize precisely the minimum order of a tree of height h that is t-free, i.e. n_t^h. We will then show that if we grow $T(u)$ and $T(v)$ at sufficient depth $\hat{h}$, their size will be less than the minimal size $n_t^{\hat{h}}$, for $t = 1, \ldots, \left\lceil \frac{\Delta}{2} \right\rceil$, and therefore their degree of freedom must be at least $\left\lceil \frac{\Delta}{2} \right\rceil + 1$, and this ensures the existence of an available color for e. The orders n_t^h are pinned down in the next couple of lemmas by a double induction.

Lemma 2. *The following holds:*

(i) $n_1^0 = 1$ *and* $n_t^0 = \infty$ *for* $t \geq 2$*;*
(ii) $n_t^1 = \Delta - t + 1$*, for* $t \geq 1$*;*
(iii) $n_t^h = 1 + (\Delta - t) \min_{1 \leq i \leq \Delta - t} n_i^{h-1}$*, for* $t \geq 1, h \geq 2$.

Proof. For (i) it is sufficient to note that $\mathcal{T}_1^0$ contains only the tree composed of a single node. We can augment it with $\Delta - 1$ edges properly colored with $1, \ldots, \Delta - 1$; with $L = \{1, 2, \ldots, \Delta\}$ we obtain the 1-freedom of the tree. Also (ii) is trivial, if we observe that, for $t \geq 1$, $\mathcal{T}_t^1$ contains only one tree, the star with $\Delta - t$ edges. The endpoints of the star are "roots" of tree of height 0. At least t colors are available at the root, regardless of how a legal triple for the star is chosen. Also, in the worst case, no more than t colors can be available at the root because its list contains just Δ colors and, by lemma 1, the set of colors of the edges can be forced to be any set of $\Delta - t$ colors.

Observations 1-2 imply that every smallest tree in $\mathcal{T}_t^h$ (that is, one having order n_t^h) has to have a root with $\Delta - t$ children, each of which is at most $(\Delta - t)$-free. Thus, this tree consists of a root connected to $\Delta - t$ smallest trees in $\bigcup_{i=1}^{\Delta - t} \mathcal{T}_i^{h-1}$. Now (iii) follows. □

Lemma 3. *The following properties hold:*

- $\mathcal{O}$*: For odd* $h \geq 1$, $n_1^h = \frac{\Delta}{2} n_{\Delta-1}^h$ *and* $n_t^h = \left(n_{\Delta-1}^h - 1\right)(\Delta - t) + 1$*, for* $2 \leq t \leq \Delta - 1$*;*
- $\mathcal{O}'$*: For odd* $h \geq 3$, $n_2^h \geq n_3^h \geq \cdots \geq n_{\lceil \frac{\Delta}{2} \rceil - 1}^h \geq n_1^h \geq n_{\lceil \frac{\Delta}{2} \rceil}^h \geq \cdots \geq n_{\Delta-1}^h$*;*
- $\mathcal{E}$*: For even* $h \geq 2$, $n_{\Delta-1}^h \leq n_1^h \leq n_t^h$*, for* $2 \leq t \leq \Delta - 2$.

Furthermore, for $h \geq 1$, $n_{\Delta-1}^h = \min_{1 \leq t \leq \Delta - 1} n_t^h$.

Proof. The three properties imply the minimality of $n_{\Delta-1}^h$, for $h \geq 1$. We show them by induction on h, starting with the base cases. For $h = 1$, the value of $n_{\Delta-1}^h$ and $\mathcal{O}$

follow from (ii) of lemma 2. For $h = 2$, (iii) and $\mathcal{O}$ imply that $n_t^2 = (\Delta - t)n_{\Delta-t}^1 + 1 = (\Delta - t)(t+1) + 1$. So the sequence $\{n_t^2\}_{t=1}^{\Delta-1}$ is bitonic: it starts by increasing and then decreases until the end. Thus to obtain $\mathcal{E}$ (which in turn implies the lemma for $h = 2$) it is sufficient to verify that $n_{\Delta-2}^2 \geq n_1^2 \geq n_{\Delta-1}^2$.

Now, assuming that for even $h - 1 \geq 2$ property $\mathcal{E}$ holds, we prove that property $\mathcal{O}$ holds for h. By (iii) and $\mathcal{E}$ we have that

$$n_t^h = 1 + (\Delta - t) \min_{1 \leq i \leq \Delta - t} n_i^{h-1} = \begin{cases} 1 + (\Delta - 1)n_{\Delta-1}^{h-1} & t = 1 \\ 1 + (\Delta - t)n_1^{h-1} & 2 \leq t \leq \Delta - 1 \end{cases}$$

This proves property $\mathcal{O}$ for $2 \leq t \leq \Delta - 1$. We consider $t = 1$ separately. By the equation above for $t = 1$ and (iii), we have that $n_1^h = 1 + (\Delta - 1)n_{\Delta-1}^{h-1} = 1 + (\Delta - 1)(n_1^{h-2} + 1)$.
Also, respectively by (iii), $\mathcal{E}$, $\mathcal{O}$ (that hold inductively), we get

$$n_{\Delta-1}^h = 1 + n_1^{h-1} = 2 + (\Delta - 1) \min_{1 \leq i \leq \Delta - 1} n_i^{h-2} = 2 + (\Delta - 1)\frac{2}{\Delta} n_1^{h-2}$$

which is equivalent to $n_1^{h-2} = (n_{\Delta-1}^h - 2)\frac{\Delta}{2(\Delta-1)}$. By substituting this term in the previous equation we obtain $n_1^h = \frac{\Delta}{2} n_{\Delta-1}^h$. Thus property $\mathcal{O}$ is proved.

To prove $\mathcal{O}'$, we first note that the sequence $\{n_t^h\}_{t=2}^{\Delta-1}$ is decreasing by property $\mathcal{O}$. So it is sufficient to prove that $n_{\lceil \frac{\Delta}{2} \rceil}^h \leq n_1^h \leq n_{\lceil \frac{\Delta}{2} \rceil - 1}^h$.

To prove $n_1^h \geq n_{\lceil \frac{\Delta}{2} \rceil}^h$ we apply $\mathcal{O}$ equations on both terms, to get the following equivalent inequality:

$$\frac{\Delta}{2} n_{\Delta-1}^h \geq (n_{\Delta-1}^h - 1)\left\lfloor \frac{\Delta}{2} \right\rfloor + 1 \iff n_{\Delta-1}^h \left\{ \frac{\Delta}{2} \right\} \geq 1 - \left\lfloor \frac{\Delta}{2} \right\rfloor$$

where $\{x\} = x - \lfloor x \rfloor$ is the fractional part of x. The LHS of the last inequality is always non-negative, while the RHS is non-positive for $\Delta \geq 3$. This proves $n_{\lceil \frac{\Delta}{2} \rceil}^h \leq n_1^h$.
For the other inequality, $n_1^h \leq n_{\lceil \frac{\Delta}{2} \rceil - 1}^h$, we proceed in an analogous way

$$(n_{\Delta-1}^h - 1)\left(\left\lfloor \frac{\Delta}{2} \right\rfloor + 1\right) + 1 \geq \frac{\Delta}{2} n_{\Delta-1}^h \Leftrightarrow n_{\Delta-1}^h \left(1 - \left\{ \frac{\Delta}{2} \right\}\right) \geq \left\lfloor \frac{\Delta}{2} \right\rfloor$$

The last inequality is implied by $\frac{1}{2} n_{\Delta-1}^h \geq \lfloor \frac{\Delta}{2} \rfloor$ which is in turn implied by $n_{\Delta-1}^h \geq \Delta$, which is true for any $h \geq 2$ just because $n_1^1 = \Delta$ by (ii) and $n_t^h > n_{t'}^{h'}$ for each t, t' as long as $h > h'$, as it can be inferred from (iii).

It remains to prove that for all even $h \geq 4$ property $\mathcal{E}$ holds. Again, we assume that properties $\mathcal{O}$ and $\mathcal{O}'$ hold for $h - 1$. For $1 \leq t \leq \lceil \frac{\Delta}{2} \rceil$, using respectively (iii), $\mathcal{O}'$ and $\mathcal{O}$ we obtain

$$n_t^h = 1 + (\Delta - t) \min_{1 \leq i \leq \Delta - t} n_i^{h-1} = 1 + (\Delta - t)\left((n_{\Delta-1}^{h-1} - 1)t + 1\right)$$

The RHS is non-decreasing for $2t \leq \Delta - (n_{\Delta-1}^{h-1} - 1)^{-1}$. This is implied by $t < \Delta/2$ assuming that $n_{\Delta-1}^{h-1} \geq 2$ (which holds for $h \geq 3$ as proven previously). Analogously, for $\lceil \frac{\Delta}{2} \rceil + 1 \leq t \leq \Delta - 1$, we obtain

$$n_t^h = 1 + (\Delta - t) \min_{1 \le i \le \Delta - t} n_i^{h-1} = 1 + (\Delta - t) n_1^{h-1} = 1 + (\Delta - t) \frac{\Delta}{2} n_{\Delta-1}^{h-1}$$

which decreases in its range of t. To complete the proof of property $\mathcal{E}$, it remains to verify that $n^h_{\lceil \frac{\Delta}{2} \rceil} \geq n^h_{\lceil \frac{\Delta}{2} \rceil + 1}$ and $n_1^h \geq n_{\Delta-1}^h$. Using the previous expressions, the former inequality is equivalent to

$$\left(\Delta - \left\lceil \frac{\Delta}{2} \right\rceil\right) \left((n_{\Delta-1}^{h-1} - 1) \left\lceil \frac{\Delta}{2} \right\rceil + 1 \right) \geq \left(\Delta - \left(\left\lceil \frac{\Delta}{2} \right\rceil + 1 \right) \right) \frac{\Delta}{2} n_{\Delta-1}^{h-1}$$

which is implied by $n_{\Delta-1}^{h-1} \geq \Delta$ (already proven for $h \geq 3$). Again by the previous expressions, it is straightforward to check that $n_1^h \geq n_{\Delta-1}^h$ holds when $\Delta \geq 3$. □

Lemma 4. *A Δ-tree T of order n, $\Delta \geq 3$ and height $h \geq 2 \log_{\Delta-1} n$ is at least $(\lceil \frac{\Delta}{2} \rceil + 1)$-free.*

Proof. The number n of nodes of T is at most $(\Delta - 1)^{\frac{h}{2}}$. Lemma 2 and 3 imply that

$$\min_{1 \le i \le \Delta - 1} n_i^h = n_{\Delta-1}^h = \begin{cases} 2\frac{(\Delta-1)^{\lceil \frac{h}{2} \rceil} - 1}{\Delta - 2} + (\Delta - 1)^{\lceil \frac{h}{2} \rceil} & h \text{ even} \\ 2\frac{(\Delta-1)^{\lceil \frac{h}{2} \rceil} - 1}{\Delta - 2} & h \text{ odd} \end{cases}$$

If h is even then $n_{\Delta-1}^h > (\Delta - 1)^{\frac{h}{2}}$. For h odd we have $n_{\Delta-1}^h > (\Delta - 1)^{\frac{h-1}{2}}$; thus, by $\mathcal{O}$,

$$n^h_{\lceil \frac{\Delta}{2} \rceil} > (\Delta - 1)^{\frac{h-1}{2}} \left\lfloor \frac{\Delta}{2} \right\rfloor + 1 > (\Delta - 1)^{\frac{h}{2}},$$

where the last inequality holds for $\Delta \geq 3$. Therefore, since $n \leq (\Delta - 1)^{\frac{h}{2}}$, if h is even we have $n < n_{\Delta-1}^h = \min_{1 \le i \le \Delta - 1} n_i^h$. It follows that T has to be Δ-free.

Analogously, if $h \geq 3$ is odd, $n < n^h_{\lceil \frac{\Delta}{2} \rceil} \leq n_1^h \leq n^h_{\lceil \frac{\Delta}{2} \rceil - 1} \leq \cdots \leq n_3^h \leq n_2^h$ by $\mathcal{O}'$, so T is at least $(\lceil \frac{\Delta}{2} \rceil + 1)$-free. For the case $h = 1$, a similar argument holds by (ii). □

Proof (of Thm. 1). We are given a graph $\mathcal{G}$ of maximum degree $\Delta \geq 3$ with girth $> 1 + 2R$ where $R = \lceil 2 \log_{\Delta-1} n \rceil$ is the lower bound on the height of a tree stated in lemma 4. We are also given a Δ-list assignment $\mathcal{L}$ for the edges of $\mathcal{G}$. We color the edges one at a time, starting with any edge and assigning it a color from its list. Assume by induction that $\mathcal{G} - e$ is colored, where $e = \{u, v\}$. Grow two BFS trees $T(u)$ and $T(v)$, respectively rooted at nodes u and v, up to distance R from their roots. From the girth assumption, these BFS' will be composed of disjoint trees. Consider the subtrees $T'(u)$ and $T'(v)$, respectively of $T(u)$ and $T(v)$, induced by the root-leaf paths of length exactly R. Both $T'(u)$ and $T'(v)$ are Δ-trees of two possible heights: either R or 0. Any tree having height R is $(\lceil \frac{\Delta}{2} \rceil + 1)$-free by lemma 4, and any tree of height 0 has no edge to color. That is, there must exist a color $c \in \mathcal{L}(e)$, unused at both endpoints of e that can be used to color e. After having colored the edges in $T'(u) \cup T'(v) \cup \{e\}$, re-coloring the rest of the edges in $T(u) \cup T(v) - (T'(u) \cup T'(v))$ is an easy matter (as they induce a forest of rooted trees having no constraints on their leaves). □

We conclude by explicitly stating that the re-coloring operations of the previous theorem can be performed in a local manner.

Lemma 5. *Let $\mathcal{G}$ be a graph of order n, maximum degree $\Delta \geq 3$ and girth greater than $2R+1$ where $R = \lceil 2\log_{\Delta-1} n \rceil$. Given any Δ-list assignment for the edges of $\mathcal{G}$, any proper partial coloring of $\mathcal{G}$, and any uncolored edge $e = \{u, v\}$ of $\mathcal{G}$, it is possible to properly color e, by changing the colors of the already colored edges of the trees rooted at u and v having height $\leq R$.*

3.1 Class-2 Graphs of High Girth

We now sketch the proof of Prop. 1 which basically states that the girth requirement of Thm. 1 is within a factor of 3 of the optimal one. To this aim we give an infinite family of Class-2 graphs having girth $\geq \frac{4}{3}\log_{\Delta-1} n - O(1)$.

These graphs can be obtained by manipulating the graphs of high girth of [15]. We will give a function that maps regular graphs of order n and girth g to Class-2 graphs of odd order $2n-1$ having girth $\geq g$. This will prove the proposition.

Let $\mathcal{G}$ be a Δ-regular graph of order n and let $\mathcal{G}'$ be the graph that $\mathcal{G}$ is mapped into; $\mathcal{G}'$ will either be a Δ-regular graph, or a graph having $2n-2$ nodes of degree Δ and a single node of degree $\Delta - 1$ — this implies that $\mathcal{G}'$ is "overfull" and, thus, belongs to Class-2 (a graph having m edges, n nodes and maximum degree Δ belongs to Class-2 if it is "overfull" — that is, if $m > \lfloor \frac{n}{2} \rfloor \Delta$).

To obtain $\mathcal{G}'$, delete any node v from $\mathcal{G}$. The graph $\mathcal{G}'$ will consist of two disjoint copies of $\mathcal{G} - v$ and of a new node v'. The new edges are as follows. Take any subset of $\lceil \Delta/2 \rceil$ nodes of degree $\Delta - 1$ in a copy of $\mathcal{G} - v$ and connect each of them to its respective node in the other copy. Finally, connect node v' to each of the remaining $2\lfloor \Delta/2 \rfloor$ nodes of degree $\Delta - 1$.

4 Algorithms

In this section we sketch some algorithmic consequences of our theorems for list colorings, with special emphasis on distributed algorithms that are our main focus. In particular, we will sketch the proof of theorem 4.

We consider the classic model suggested by Linial [14] of a synchronous, message-passing distributed network. The running time of an algorithm is given by the number communication rounds. In each round a processor can broadcast a message to all of its neighbors, receive messages from all of them, and perform any amount of local computation. An algorithm is *efficient* if its running time is at most poly-logarithmic in the network size.

All list colorings can be computed efficiently in a distributed setting by means of some sort of meta-algorithm. We will see that this algorithm can be implemented in such a way that each processor actually performs only a polynomial amount of computation. Thus it could also be simulated sequentially in polynomial time. The meta-algorithm operates on a conflict graph C of the input graph $\mathcal{G}$. For vertex coloring $C = \mathcal{G}$; for edge coloring, C is the line graph of $\mathcal{G}$; for total coloring, C is the so-called total graph of $\mathcal{G}$, i.e. there is a node in C for every edge or vertex in $\mathcal{G}$ and two nodes are adjacent in C if

their respective elements are adjacent, or incident, in $\mathcal{G}$. The meta-algorithm produces a vertex coloring of C, regardless of the requested type of coloring.

The main idea is that the local nature of list colorings illustrated in the previous sections allows to color several nodes of C in parallel. For these operations not to interfere, it is sufficient to ensure that the corresponding re-coloring trees do not overlap. This can be achieved by selecting, in each phase of the algorithm, the nodes to re-color according to a network decomposition of a power of C. Recall that an (α, β)-decomposition of a graph $\mathcal{G}$ is a partition of the nodes of $\mathcal{G}$ into β-weakly-connected components, each labeled with an integer in $\{1, \ldots, \alpha\}$, such that two adjacent components have different labels (two components are adjacent if at least one edge in $\mathcal{G}$ hits both of them). A β-weakly-connected component of $\mathcal{G}$ is a subset of the nodes of $\mathcal{G}$ such that any two nodes of the subset are at distance at most β in $\mathcal{G}$. For the sake of brevity, we use the word *cluster* to refer to a β-weakly-connected component. In [1] the authors give a randomized distributed algorithm that obtains $(O(\log n), O(\log n))$-decompositions in $O(\log n)$ many rounds.

Lemma 5 gives a logarithmic bound on the depth of the trees to be considered for the edge re-coloring operations. The following lemmas, that we state without proof for lack of space, give analogous bounds for the vertex and total re-coloring operations.

Lemma 6. *Let $\mathcal{G}$ be a graph of order n and girth greater than $2R + 1$, where $R = \lceil \log_{k-1} n \rceil$ with $k \geq 3$. Given any k-list assignment for the nodes of $\mathcal{G}$, any proper partial coloring of $\mathcal{G}$, and any uncolored node v of $\mathcal{G}$, it is possible to properly color v by changing the colors of the already colored nodes of the tree rooted at v of height $\leq R$.*

Lemma 7. *Let $\mathcal{G}$ be a graph of order n, maximum degree $\Delta \geq 3$ and girth greater than $2R+1$ where $R = \lceil 2 \log_{\lceil \frac{\Delta}{2} \rceil} n \rceil + 1$. Given any $(\Delta + 1)$-list assignment for edges and nodes of $\mathcal{G}$, any proper partial coloring of $\mathcal{G}$, and any uncolored edge $e = \{u, v\}$ of $\mathcal{G}$, it is possible to properly color e, u and v, by changing the colors of the already colored nodes and edges of the trees rooted at u and v having height $\leq R$.*

Note that for each kind of coloring, the depth of the trees to be re-colored is bounded by some $d \in O(\log n)$. To ensure non-interference between different re-coloring operations, we compute a decomposition of C^t, for $t = 2d + O(1)$, where C^t, the t-th power of C, is the graph such that $V(C^t) = V(C)$ and where two nodes are connected in C^t whenever they are at distance at most t in C.

In each cluster, the re-coloring operations will all be handled by just one of its nodes. This node (or *leader*) can be chosen as the one having the smallest ID of the cluster. The re-coloring operations of equally-labeled clusters can be performed in parallel, as their distance in C is greater than or equal t. The meta-algorithm is as follows. For each label k, in parallel, each leader of a k-colored cluster,

1. gathers all the information about its cluster K and its neighbors;
2. colors, locally, the nodes of K with respect to the edges of C;
3. broadcasts the new colors to the nodes in K.

This algorithm runs for $O(T + \alpha\beta t)$ communication rounds, where T is the time needed to compute the (α, β)-decomposition of C^t. Using the randomized algorithm in [1], we have $\alpha = \beta = O(\log n)$ and $T = O(t \log n) = O(\log^2 n)$. This proves thm. 4.

So far, we have neglected to discuss how a leader can (locally) compute the re-coloring of a tree. In Linial's model, each node of the network is allowed to perform an unlimited amount of computation, thus the tree re-colorings (the most complex operations performed by the leaders) could just be obtained by exhaustive search. Nonetheless, the re-colorings can be computed in polynomial time. First, note that to re-color a tree it is sufficient to compute the list of available colors of a generic object in the tree (i.e., edge and/or node depending on the type of coloring), given the lists of its children in the tree. This can be solved efficiently by means of a reduction to the maximum bipartite matching problem.

The proof of the following proposition is omitted for lack of space.

Proposition 2. *The recoloring of a tree can be computed in time $O(n \cdot \Delta^{7/2})$ for list edge coloring, in time $O(n \cdot \Delta^{9/2})$ for list total coloring and in time $O(n \cdot k)$ for list vertex coloring.*

Acknowledgments

We greatly thank Alessandro Panconesi for his guidance and for his constant encouragement, advice and support. We also thank Zoltán Füredi, Alexandr V. Kostochka, Ravi Kumar, Mike Molloy, Romeo Rizzi and Aravind Srinivasan for useful comments and tips.

References

1. Anil Kumar, V.S., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: End-to-end packet-scheduling in wireless ad-hoc networks. In: Proc. 15th ACM-SIAM Symp. on Discrete Alg (SODA 2004), pp. 1021–1030 (2004)
2. Behzad, M.: The total chromatic number. Comb. Math. and its Appl. (Proc. Conf., Oxford 1969). Academic Press, London (1971)
3. Bollobás, B., Harris, A.J.: List colorings of graphs. Graphs and Combinatorics 1, 115–127 (1985)
4. Borodin, O.V., Kostochka, A.V., Woodall, D.R.: List edge and list total colourings of multigraphs. J. Comb. Theory, Series B 71 (1997)
5. Czygrinow, A., Handckowiak, M., Karonski, M.: Distributed $O(\Delta \log n)$-Edge-Coloring Algorithm. In: Proc. 9th Europ. Symp. on Alg. (2001)
6. Dubhashi, D., Grable, D., Panconesi, A.: Nearly-optimal, distributed edge-colouring via the nibble method. Theoretical Computer Science 203 (1998)
7. Erdös, P.: On circuits and subgraphs of chromatic graphs. Mathematika 9, 170–175 (1962)
8. Erdös, P., Rubin, A.L., Taylor, H.: Choosability in graphs. In: Proc. West Coast Conf. on Combinatorics, Graph Theory and Computing, Congressus Numerantium XXVI, pp. 125–157 (1979)
9. Isobe, S., Zhou, X., Nishizeki, T.: Total colorings of degenerate graphs. Combinatorica 27, 167–182 (2007)
10. Jensen, T.R., Toft, B.: Graph Coloring Problems. John Wiley & Sons, New York (1995)
11. Juvan, M., Mohar, B., Škrekovski, R.: List Total Colourings of Graphs. Comb., Prob. and Comp. 7(2) (1998)

12. Kahn, J.: Asymptotics of the List Chromatic Index for Multigraph. Random Struct. & Alg. 17, 117–156 (2000)
13. Kostochka, A.V.: List edge chromatic number of graphs with large girth. Discrete Math. 101 (1992)
14. Linial, N.: Locality in distributed graph algorithms. SIAM J. on Comp. 21(1), 193–201 (1992)
15. Lubotzky, A., Phillips, R., Sarnak, P.: Ramanujan graphs. Combinatorica 8, 261–277 (1988)
16. Matoušek, J.: Lecture Notes in Discrete Geometry. Springer, New York (2002)
17. Molloy, M., Reed, B.: A bound on the Total Chromatic Number. Combinatorica 18, 241–280 (1998)
18. Panconesi, A., Srinivasan, A.: Fast randomized algorithms for distributed edge coloring. SIAM J. on Comp. 26(2) (1997)
19. Vizing, V.G.: On an estimate of the chromatic class of a p-graph. Diskret. Analiz. 3, 25–30 (1964)
20. Zhou, X., Nishizeki, T.: Edge-coloring and f-coloring for various classes of graphs. J. Graph Algorithms and Applications 3(1) (1999)

Approximating List-Coloring on a Fixed Surface

Ken-ichi Kawarabayashi*

National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
k_keniti@nii.ac.jp

Abstract. It is well-known that approximating the chromatic number within a factor of $n^{1-\varepsilon}$ cannot be done in polynomial time for any $\varepsilon > 0$, unless $coRP = NP$. Also, it is known that computing the list-chromatic number is much harder than the chromatic number (assuming that the complexity classes NP and $coNP$ are different). In fact, the problem of deciding if a given graph is f-list-colorable for a function $f : V \to \{k-1, k\}$ for $k \geq 3$ is Π_2^p-complete.

In this paper, we are concerned with the following questions:

1. Given a graph embedded on a surface of bounded genus, what is its list-chromatic number ?
2. Given a graph embedded on a surface of bounded genus with list-chromatic number k, what is the least l ($l \geq k$) such that the graph can be efficiently and legally colored given a list (coloring scheme) of order l ?

The seminal result of Thomassen [19] gives rise to answers for these problems when a given graph is planar. In fact, he gave a polynomial time algorithm to 5-list-color a planar graph. Thomassen's result together with the hardness result (distinguishing between 3, 4 and 5 list-colorability is NP-complete for planar graphs and bounded genus graphs) gives an additive approximation algorithm for list-coloring planar graphs within 2 of the list-chromatic number.

Our main result is to extend this result to bounded genus graphs. In fact, our algorithm gives a list-coloring when each vertex has a list with at least $\chi_l(G) + 2$ colors available. The time complexity is $O(n)$.

It also generalizes the other deep result of Thomassen [20] who gave an additive approximation algorithm for graph-coloring bounded genus graphs within 2 of the chromatic number.

This theorem can be compared to the result by Kawarabayashi and Mohar(STOC'06) who gave an $O(k)$-approximation algorithm for list-coloring graphs with no K_k-minors. For minor-closed graphs, there is a 2-approximation algorithm for graph-coloring by Demaine, Hajiaghayi and Kawarabayashi (FOCS'05), but it seems that there is a huge gap between list-coloring and graph-coloring in minor-closed family of graphs. On the other hand, this is not the case for bounded genus graphs, as we pointed out above.

* Research partly supported by JSPS Postdoctoral Fellowship for Research Abroad.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 333–344, 2008.

1 Introduction

1.1 Coloring and List-Coloring

Graph coloring is arguably the most popular subject in graph theory. Also, it is one of the central problems in combinatorial optimization, since it is one of the hardest problems to approximate. In general, the chromatic number is inapproximable in polynomial time within factor $n^{1-\epsilon}$ for any $\epsilon > 0$, unless $coRP = NP$, cf. Feige and Kilian [8]. An interesting variant of the classical problem of properly coloring the vertices of a graph with the minimum possible number of colors arises when one imposes some restrictions on the colors or the number of colors available to particular vertices. This variant received a considerable amount of attention by many researchers, and that led to several beautiful conjectures and results. This subject, known as *list-coloring*, was first introduced in the second half of the 1970s, in two papers by Vizing [25] and independently by Erdős, Rubin and Taylor [7].

If $G = (V, E)$ is a graph, and f is a function that assigns to each vertex of v in G a positive integer $f(v)$, we say that G is *f-choosable* (or *f-list-colorable*) if for every assignment of sets of integers $S(v) \subseteq \mathcal{Z}$, where $|S(v)| = f(v)$ for all $v \in V(G)$, there is a proper vertex coloring $c : V \to \mathcal{Z}$ so that $c(v) \in S(v)$ for all $v \in V(G)$. We sometimes call $f(v)$ *assignment of colors*. The smallest integer k such that G is f-choosable for $f(v) = k$ $(v \in V(G))$ is the *list-chromatic number* $\chi_l(G)$. If G is f-choosable for $f(v) = k$ $(v \in V(G))$, we sometimes call G has a list-coloring using at most k colors. Clearly, $\chi(G) \leq \chi_l(G)$, and there are many graphs for which $\chi(G) < \chi_l(G)$. A simple example is the complete bipartite graph $K_{2,4}$, which is not 2-choosable. Another well-known example is the complete bipartite graph $K_{3,3}$. In fact, it is easy to show that for every k, there exist bipartite graphs whose list-chromatic number is bigger than k.

The problem of computing the list-chromatic number of a given graph is thus difficult, even for small graphs with a simple structure. It is shown in [10] that the problem of deciding if a given graph is f-list-colorable for a function $f : V \to \{k-1, k\}$ for $k \geq 3$ is Π_2^p-complete. Hence if the complexity classes NP and $coNP$ are different, as is commonly believed, the problem is strictly harder than the NP-complete problem of deciding if the chromatic number is k (if $k \geq 3$).

1.2 List-Coloring Planar Graphs

Although there are many negative results as stated above, there are some positive results, which are mainly connected to the Four Color Theorem. One celebrated example is Thomassen's result on planar graphs [19]. It says that every planar graph is 5-choosable, and its proof is within 20 lines and gives rise to a linear time algorithm to 5-list-color planar graphs. In contrast with the Four Color Theorem, there are planar graphs that are not 4-choosable [23]. These were conjectured by Erdős, Rubin and Taylor [7].

1.3 List-Coloring Bounded Genus Graphs and Our Main Result

In this paper, we are interested in bounded genus graphs. Our main target is to extend the result of Thomassen [19] to the bounded genus graphs. We develop an additive approximation algorithm for list-coloring bounded genus graphs within 2 of the list-chromatic number. Our main result is the following.

Theorem 1. *Suppose G is embedded on a fixed surface. Then there is a linear time additive approximation algorithm to list-color G within two of the list-chromatic number of G.*

Theorem 1 is actually more specific in the following sense.

1. The list-chromatic number of a bounded genus graph can be computed exactly when it is not 3, 4, or 5 (Similarly, such graphs can be efficiently colored given a list of order k with $k \geq 6$).
2. For $k = 3, 4, 5$, the list-chromatic number (say, k) of a bounded genus graph can be calculated up to an additive approximation of $\min(2,\ 6 - k)$ (And moreover, one can efficiently and legally list-color the graph given a list of order $\min(k + 2,\ 6)$).

Let us make some remarks at this moment. The bound $\chi_l(G)+2$ is essentially best possible, even for planar graphs, since it is Π_2^P-complete to decide whether or not they are 4-list-colorable [24], and they are 3-list-colorable. This implies that distinguishing between 3, 4 and 5 list-colorability is NP-complete for graphs on any fixed surface. This also means that unless $coRP = NP$, one cannot approximate the list-chromatic number of planar graphs and bounded genus graphs within 1 of the list-chromatic number $\chi_l(G)$.

Let us now give some remarks concerning graph-coloring. Improving to an $\chi(G)+1$ approximation algorithm for graph-coloring of graphs on a fixed surface would be probably very hard. In fact, distinguishing between 3 and 4 colorability is NP-complete for graphs on any fixed surface, and distinguishing between 4 and 5 colorability for graphs on a fixed surface would require a significant generalization of the Four Color Theorem, which would characterize the 4-colorability of graphs in fixed surfaces. Apart from planar graphs (which is exactly the Four Color Theorem), it is not known how to test the 4-colorability of graphs on a fixed surface in polynomial time.

Leaving the plane to consider graphs on surfaces of higher genus, the chromatic and list chromatic number can increase. However, for graphs which obey certain local planarity conditions, one can deduce similar properties as for planar ones. We say that a graph G embedded in a surface S is *locally planar* if it does not contain short noncontractible cycles. Quantitatively, we introduce the *edge-width* of G as the length of a shortest cycle which is noncontractible in S. Thomassen proved in [20] that graphs in S with sufficiently large edge-width are 5-colorable. He asked if they are also 5-choosable. This was answered affirmatively in [4].

Theorem 2. *For every surface S there exists a constant w such that every graph that can be embedded in S with edge-width at least w is 5-choosable.*

The method used in this paper for the proof of Theorem 1 is quite different from that in [4].

1.4 Overview of the Algorithm

Roughly, the algorithm proceeds as follows. If $\chi_l(G) = 2$, then we can easily recognize this and 4-list-color the whole graph by using the result by Erdős, Rubin and Taylor [7]. So, we may assume that $\chi_l(G) \geq 3$.

The idea is to delete disjoint disks (each of which bounds a planar graph on a fixed surface) as many as possible. Then the resulting graph has bounded tree-width. To see this, if the tree-width is huge, then by the result of Thomassen [21], there must be a big grid-minor, whose inside induces a planar graph. Conversely, if there is no big grid minor, whose inside induces a planar graph, then the tree-width of the whole graph is bounded. Since we have already deleted the disks, so there would not be a big grid-minor, whose inside induces a planar graph. Hence after deleting finitely many disks, the resulting graph has bounded tree-width. The methods of dynamic programming can be applied successfully for list-colorings once we have a desired tree-decomposition, see [1,24,9]. Also, by the algorithm of Bodlaender [2], once we know the tree-width of small order, we can construct a tree-decomposition in linear time. Hence we know the list-chromatic number of the resulting graph G'. Actually, we can list-color G' too.

It remains to extend the list-coloring of G' to each of the deleted disks using at most $\max\{\chi_l(G')+2, 5\}$ colors. As we mentioned, we may assume that $\chi_l(G) \geq 3$. The key is that Thomassen's result tells us that if a given graph is planar, and each of the vertices on the boundary has a list with three colors available, and the rest of vertices have lists with five colors available, then we can list-color the whole graph. Since we are allowed to list-color each of the deleted disks using at most $\max\{\chi_l(G') + 2, 5\}$ colors, so we have extra two colors for the vertices on the boundary of each deleted disk. If we can save another one color in the list of each vertex on the boundary of these disks, we will be able to list-color the graph in each of the disks. This is the main issue here, and to overcome this difficulty, we will need Thomassen's recent extension [22] of his result in [19].

More specifically, the algorithm facilitates a 3 step solution:

1. Delete at most 2 colors from every vertex list in the boundary of each disk (as well as a small buffer zone inside each disk).
2. After deleting all the flat disks, color the rest of the graph together with the disk boundary and the buffer using the new list. This can be done by tree-width bounded method and dynamic programming. In fact, we will get an optimal coloring.
3. Using the original list, color the interior of each disk, possibly re-coloring some vertices in the buffer zone.

All of these steps can be done in linear time. In Step 2, we will get an optimal coloring. So, in Step 3, we would get a list-coloring using at most $\chi_l(G) + 2$ colors, since for each vertex in the boundary of each disk, we are allowed to use another 2 colors that are deleted in Step 1.

2 Definitions and Preliminaries

2.1 Definitions

For basic graph theoretical notation, we refer the reader to the book of Diestel [5]. So we assume basic notations in graph theory.

Before we give an algorithm, we need the following lemma. We say that the disk is *flat* if it is embedded into a sphere.

Lemma 1. *For any graph G on a fixed surface, there is a polynomial time algorithm to find flat disjoint disks $D_1, \ldots, D_l$ such that deletion of them results in a graph on a fixed surface without a flat disk containing* 10 *nested cycles.*

This can be certainly done in polynomial time, in fact, in linear time. Most of the techniques are introduced in [17]. In [14,15], Reed, Robertson, Schrijver and Seymour used this technique to give a linear time algorithm for the k disjoint paths problem for planar graphs for fixed k.

2.2 List-Coloring Planar Graphs

Let G be a graph. A *list-assignment* is a function L which assigns to every vertex $v \in V(G)$ a set $L(v)$ of natural numbers, which are called *admissible colors* for that vertex. An *L-coloring* of the graph G is an assignment of admissible colors to all vertices of G, i.e., a function $c : V(G) \to \mathcal{N}$ such that $c(v) \in L(v)$ for every $v \in V(G)$, and for every edge uv we have $c(u) \neq c(v)$. If k is an integer and $|L(v)| \geq k$ for every $v \in V(G)$, then L is a *k-list-assignment*. The graph is *k-choosable* if it admits an L-coloring for every k-list-assignment L. Sometimes, we call it *k-list-colorable*.

In order to get our algorithm, we need to know which kind of graphs are 2-list-colorable. The following result gives the answer.

Lemma 2 ([7]). *A graph is 2-choosable if and only if it is bipartite graph plus some structures, which can be easily recognized.*

The following is the well-known result due to Thomassen [19].

Theorem 3 (Thomassen [19]). *Let G be a plane graph with outer facial walk C, and let a, b be adjacent vertices on C. Let L be a list-assignment for G such that $L(a) = \{\alpha\}$, $L(b) = \{\beta\}$, where $\beta \neq \alpha$, every vertex on $C \setminus \{a, b\}$ has at least three admissible colors, and every vertex that is not on C has at least five admissible colors. Then G can be L-colored.*

Finally, we need the following recent result, which is a generalization of Theorem 3. To state this result, we need some definition. Let G be a near-triangulation, i.e., each face, except for the outer face boundary, is a triangle. Suppose that the outer face boundary of C consists of a vertex set $v_1, \ldots, v_{l-1}$ in this clockwise order. If the interior of G consists of the edges $v_1v_2, v_1v_4, \ldots, v_1v_{l-1}$, then we call G a *broken wheel*. We also call it a *generalized wheel*. We call v_1 its *major vertex*

and $v_l v_1 v_2$ its *principal path*. We also say that $v_l v_1, v_1 v_2$ are the *principal edges* and that v_l, v_2 are the *principal neighbors* of v_1. If $k \geq 4$, then this generalized wheel is clearly not 3-extendable with respect to its principal path. If the interior of G consists of a vertex u and all edges from u to the outer cycle, then G is a wheel. We also call that a *generalized wheel*, and again, we call v_1 its *major vertex* and $v_l v_1 v_2$ its *principal path*. It is easy to see that this generalized wheel is not 3-extendable with respect to its principal path when k is odd and $k \geq 5$. Finally, if G_1, G_2 are generalized wheels and we identify a principal edge in one of them with a principal edge in the other in such a way that their major vertices are identified, then the resulting graph is also called a generalized wheel. Its two principal edges are those which are principal edges in one of the graphs, but not part of the identification above. Again, it is easy to see that this generalized wheel is not 3-extendable with respect to its principal path unless it contains a vertex of even degree or degree 3 inside the outer cycles. We can now state the theorem.

Theorem 4 (Thomassen [22]). *Let G be a plane graph with outer facial walk C, and let a, b, c be a path of length 2 on C. Let L be a list-assignment for G such that $L(a) = \{\alpha\}$, $L(b) = \{\beta\}$, $L(c) = \{\gamma\}$, where $\beta \neq \alpha$ and $\beta \neq \gamma$, every vertex on $C \setminus \{a, b, c\}$ has at least three admissible colors, and every vertex that is not on C has at least five admissible colors. Then G can be L-colored, unless G contains the generalized wheel W with the principal path abc such that each vertex of $C \setminus \{a, b, c\}$ has exactly three colors in its list.*

Let us remark that the proofs of Theorems 3 and 4 can be translated into linear time algorithms to list-color a planar graph, if such a coloring exists.

3 Main Lemma

In this section, we prove the following lemma, which is a key for our algorithm. This is concerning Steps 1 and 3 in the overview. This lemma deals with a flat disk, and shows how we save three colors in the list of each vertex on the boundary of each deleted disks. Then we shall 5-list-color all the vertices in the disk by Theorems 3 and 4. As we pointed out before, we may assume that the list-chromatic number of G is at least 3, otherwise, we are able to list-color the whole graph by Lemma 2. So the main challenge is to list-color the graphs inside disks using at most $\chi_l(G) + 2 \geq 5$ colors.

Before that, we need some definition. By a *block*, we mean a maximal connected subgraph B of G such that no vertex of B is a cutvertex of B. Let us observe that any two blocks have at most one vertex in common, and clearly a vertex of G is a cutvertex if and only if it is contained in more than one block of G. Every connected graph G has a *block decomposition* $(T, \mathcal{B})$ where T is a tree and $\mathcal{B} = \{B_v | v \in V(T)\}$ is a collection of subsets of vertices of G indexed by the vertices of T such that the following hold:

i. for every $v \in V(T)$, $G[B_v]$ is either an edge or a block of G,
ii. for every edge uv of T, $|B_v \cap B_u| = 1$, and
iii. every edge of G is contained in B_v for some $v \in V(T)$.

Observe that for any edge $uv \in E(T)$, the vertex in $B_u \cap B_v$ is a cut vertex of the graph. See [5] for more details.

Lemma 3. *Suppose D is a flat disk, and C is the outer face boundary of this disk. Let $V(D)$ be the vertices in the disk D. Let W be a vertex set in $V(D)-V(C)$ such that every vertex in W has at least 3 neighbors in C. Suppose furthermore that every vertex in $V(D)$ has a list with at least $k \geq 5$ colors available. Then we can delete two colors from the list of each vertex in $C \cup W$ such that the following is possible:*

Suppose a precoloring of $C \cup W$, which only uses the colors in the new (truncated) list of each vertex in $C \cup W$, is given. Then this precoloring of C (not $C \cup W$!) can be extended to a coloring of $V(D)$.

Proof. Note that, in the statement of Lemma 3, we shall possibly re-color the vertices of W (but we shall not re-color the vertices in C). The vertices in W are called "buffer zone" in the overview section, section 1.4.

Our first goal is to delete two colors from the list of each vertex in $C \cup W$ so that, for any precoloring of $C \cup W$, which only uses the colors in the new (truncated) list of each vertex in $C \cup W$, each vertex v in W (except for at most three vertices in each block of $V(D) - C$, which we call *exceptional vertices*, and we shall specify later) still has three colors $\{c_1, c_2, c_3\}$ in its list in such a way that any color of c_1, c_2, c_3 are not used in the precoloring of $N(v) \cap V(C)$. More specifically, one color of c_1, c_2, c_3, say c_1, comes from the precoloring of $C \cup W$. The other two colors, c_2 and c_3, are deleted. Let us prove our first goal. We are not interested in the vertices in $V(C) - N(W)$. To achieve our purpose, we just delete arbitrary two colors from the list of each vertex in $V(C) - N(W)$. Hence, hereafter, we are only interested in the vertices in $W \cup N(W)$.

It is clear that $V(D) - V(C)$ consists of block decompositions. Take one block decomposition $(T, \mathcal{B})$. Note that W appears on the outer face boundary of this block decomposition. We fix a root r of T. Take all the vertices of $B_r \cap W$. If $B_r \cap W = \emptyset$, then our purpose for B_r can be achieved. So assume $B_r \cap W \neq \emptyset$. We also specify one vertex c in the outer face boundary of $B_r \cap W$, and delete two colors a, b from the list of c (We assume that both colors a and b are in the list of c.). Let $v_1, \ldots, v_l$ be the vertices of W on the face boundary of B_r that appear in the clockwise order, where $c = v_1$. We also put $d = v_l$. Later, d will play a role. Note that d may not exist if $|B_r \cap W| = 1$.

Let us now remark that we will appoint d as *exceptional* if c, u, d consists of a 2-path, where $u \in V(C)$ is a neighbor of both c and d. In this case, our argument below may be problematic. We just delete two colors from the list of u, as we shall see below, and this may result in a conflict with our purpose. The reader may wonder why this does not harm. The idea is to use Theorem 3. It allows us to precolor at most two adjacent vertices in the outer face boundary. So, we are allowed to precolor the edge ud when we apply Theorem 3. Therefore, our goal

here is to delete two colors from the list of each vertex in $(B_r \cap W) \cup (N(B_r \cap W) \cap V(C))$.

Let $v_{i,1}$ be the vertex in $N(v_i) \cap C$ that appears first, and $v_{i,2}$ be the vertex in $N(v_i) \cap C$ that appears last in the clockwise order, for $i = 1, \ldots, l$. We first delete a, b from the list of each vertex in $N(v_1) \cap V(C)$. If a, b is in the list of $v_{1,2}$, then we delete a, b from the list of $v_{1,2}$. Else, we delete a, a' or b, a' or a', b' from the list, where both a' and b' are in the list of $v_{1,2}$. Suppose, say, we delete x, y from the list of $v_{1,2}$. If $v_{2,1} \neq v_{1,2}$, then we select arbitrary two colors in the list of $v_{2,1}$, and delete two colors from the list. Note that $v_{2,1} = v_{1,2}$ could happen. In this case, of course, we delete x, y from the list of $v_{2,1}$. Suppose we delete x', y' from the list of $v_{2,1}$. Delete x', y' from the list of v_2. If v_2 does not have x' or y' in the list, then we select arbitrary color(s) in the list of v_2, and delete two colors from the list. So whenever v_2 has x' or y' in the list, we shall delete. We keep doing this procedure until v_l. Note that we are following the outer face boundary of B_r in the clockwise order, therefore, there are no overlaps (besides $v_{2,1} = v_{1,2}$, $v_{l,2} = v_{1,1}$, etc.). The problem could be the list of $v_{l,2}$ since $v_{l,2}$ could have already lost two colors from the list because $v_{l,2}$ could be $v_{1,1}$. But in this case, $c, v_{1,1} = v_{l,2}, d$ consist of a path of length 2, and d is now an exceptional vertex. We jsut delete a, b from $v_{1,1} = v_{l,2}$. Then we are done with $B_r \cap W$.

Let $B_{r'}$ be a child of B_r with a cutvertex v. Note that v may not be in W. If v is not in W, then we perform the same procedure as we described in the previous paragraph, starting with an arbitrary vertex in $B_{r'} \cap W$ (If $B_{r'} \cap W = \emptyset$, then our purpose for $B_{r'}$ can be achieved.). We shall delete two colors in each list of the vertices in $((W \cap B_{r'}) \cup (N(B_{r'} \cap W) \cap V(C))$. Let u in W be a vertex, which is closest to v in the outer face boundary of $B_{r'}$ (Note that u may not exist if $|B_{r'} \cap W| = 0$ or 1. But in these cases, it is easy to achieve our goal.). Since we have already performed our procedure in B_r, so we may already delete two colors from the list of v. We appoint v as c, and appoint u as d.

Let us now remark that we will appoint d as *excpetional* if c, u, d consists of a 2-path, where $u \in V(C)$ is a neighbor of both c and d. In this case, this argument may be problematic. We just delete two colors from the list of u, as above, and this may result in a conflict with our purpose. (In this case, we shall call d an exceptional vertex. We also observe that since two colors in the list of c may be already deleted from the previous procedure for B_r, so we call it an exceptional vertex too). The reader may wonder why this does not harm. The idea is to use Theorem 3 or Theorem 4. It allows us to precolor at most two or three adjacent vertices in the outer face boundary. So we are allowed to precolor the edge ud or the 2-path cud when we apply Theorem 3 or Theorem 4. Therefore, our goal here is to delete two colors from the list of each vertex in $(B_r \cap W) \cup (N(B_r \cap W) \cap V(C))$.

We delete two colors in each list of the vertices in $((W \cap B_{r'}) \cup (N(B_{r'} \cap W) \cap V(C))$, as we did in the previous paragraph, starting with c, and ending with d. Note that two colors in the list of c may be already deleted from the previous procedure for B_r. For $u = d$, we do exactly as we did in the previous paragraph. $(N(v) \cap V(C)) \cap (N(u) \cap V(C))$ may not be empty. In this case, $u, N(v) \cap V(C), v$

consist of a path of length 2, in which case, both c and d are exceptional vertices. Note that we just delete a, b from the list of $N(v) \cap V(C)$, as we did above.

In this way, we can keep doing this procedure from the root of T to the leaves, for all the block decompositions in $V(D) - C$. So, we are now able to delete two colors from the list of each vertex of $C \cup W$. In fact, we just delete two colors from the list of vertices in $C \cap W$, and if there are exceptional vertices, then they form a path of length 1 or 2 in a block of the block decomposition $(T, \mathcal{B})$.

Having had this situation, we shall prove our main assertion in Lemma 3.

Suppose that $W \cup C$ is precolored (from the new truncated lists). So, each vertex v in W has a list with one color, which is not used in the coloring of $N(v) \cap C$ at this moment. Let us keep in mind that, as we proved before, for any vertex v in W (except for exceptional vertices), the precoloring of $N(v) \cap C$ does not use two deleted colors of v.

It suffices to list-color each component of $V(D) - V(C)$. Again, fix one of a block tree, say, a block decomposition $(T, \mathcal{B})$, in $V(D) - V(C)$. Fix a root r. Let c, d be the vertices as defined above. Then for each list of each vertex v of $(B_r \cap W) - \{c, d\}$, we put the two deleted colors back to the list of v. If $v_{1,1} \neq v_{l,2}$, then we can put the two colors of d back to its list. Otherwise, since d is exceptional, we may need to deal with the case when $c, v_{1,1}, d$ is precolored, and they consist of a 2-path. For each vertex v in the outer face boundary of B_r, which is not in W, we just delete at most two colors from its list, but make sure that any of colors in the new list of v does not contain the colors of the coloring in $N(v) \cap V(C)$. This is possible since v has at most two neighbors in C.

Then it is easy to see that, currently, each vertex v in the outer face boundary of $B_r - \{c, d\}$ has a list with at least three colors, and these colors do not appear in the precoloring of $N(v) \cap C$. d may be precolored, but in this case, $c, v_{1,1}, d$ is a 2-path and we may assume that they are all precolored (if we give a color to c). Otherwise, d has a list with at least three colors.

Then we can list-color all the vertices in B_r or $B_r \cup \{v_{1,1}\}$ by Theorem 3 or Theorem 4 from the current lists. Note that, if the block B_r is the generalized wheel as in Theorem 4, then the vertices between c and d on the outer face boundary has a list with at least four colors available, since there are no vertices of W between c and d on the outer face boundary. So the obstruction in Theorem 4 cannot happen.

Let $B_{r'}$ be a child of B_r with a cutvertex v. Let u be as in the previous proof. Then we assume that v is precolored. We then perform the same procedure as we described in the previous paragraph.

In this way, we can keep list-coloring all the vertices in the block decomposition $(T, \mathcal{B})$ from the root of T to the leaves, for all the block decompositions in $D - C$. This completes the proof. □

Let us observe that this argument is algorithmic, since we know that there are linear time algorithms for Theorems 3 and 4, respectively. Other arguments are easy to implement in linear time.

4 Algorithm

Now we are ready to describe our algorithm.

Input: A graph G on a fixed surface.

Output: Give a number c that is at most $\chi_l(G)+2$. Also if each vertex has a list with $c \leq \chi_l(G)+2$ colors available, then the output gives a desired coloring. The time complexity is $O(n)$.

Description

Step 1. Check whether G is 2-choosable or not. If it is, then just list-color the graph G using Lemma 2, and output it. Otherwise, go to Step 2.

Step 2. Apply Lemma 1 to delete disjoint disks $D_1, \ldots, D_l$ so that there is no flat disk that contains 10 nested cycle in the resulting graph.

Actually, we shall adjust each disk so that each vertex in the resulting graph is adjacent to only one flat disk that was deleted. This can be done, for instance, by putting the vertices on the outer cycle C of each deleted flat disk back to the resulting graph.

Let G' be the resulting graph. Note that this operation tells us that C in Lemma 3 for each deleted disk is now in G'.

Let W be all the vertices in $G - G'$ each of which has at least 3 neighbors in G'. Then we add W to G'. Let G'' be the resulting graph. So this operation tells us that W in Lemma 3 for each deleted disk is now in G''.

Hence both W and C in Lemma 3 are in G''

Step 3. Determine $\chi_l(G'')$. At this moment, we may assume that $\chi_l(G) \geq 3$. So output $c = \max\{\chi_l(G'')+2, 5\}$. This can be done by the tree-width bounded method since G'' has small tree-width. But let us remark that at this moment, we do not give a valid coloring to G'' yet. We need to delete two colors from the list of each vertex in C and W as in Lemma 3 for each deleted disk. This will be done in Step 4.

Step 4. At this moment, $\chi_l(G) \geq 3$, and hence we may assume that each vertex in G has a list with $\max\{\chi_l(G'')+2, 5\}$ colors available. For each deleted disk D, we first add W to D, and then add the first nested cycle C of the deleted disk D to D. Note that C and W are as in Lemma 3. Then for this resulting disk D' and the vertices inside the disk, we perform the algorithm of Lemma 3 with C and W. Note that C and W in Lemma 3 are, at the moment, both in D' and in G''. When we perform the algorithm, as in the first half of the proof of Lemma 3, we first delete at most two colors from the list of each vertex in C and W. Then we apply the tree-width bounded method to G''. We now have a valid coloring of G'' from the new truncated list. Then we extend the coloring of C to the vertices inside the disk D' by possibly re-coloring the vertices of W, as we did in the second half of the proof of Lemma 3.

Finally, we put the coloring together. This completes the description of the algorithm.

For the correctness, if $\chi_l(G)$ is at most 2, then Lemma 2 gives the answer, and hence Step 1 works.

For Step 2, it is clear that no vertex of the resulting graph G' is surrounded by 20 nested cycles since we just put the outer cycle of each flat disk back to G'. So, by the result of Thomassen [21], the tree-width of G' is bounded. We also need to prove that in Step 2, after adding W to G', the resulting graph G'' has still bounded tree-width. Since we only add the first neighborhood of each disk to G', so the size of grid minor in G'' could increase only by factor 2 from G'. Therefore, by the mini-max formula of tree-width and the size of grid-minors, which is proved in [6,13,16,18], the tree-width of G'' is still bounded. So, in Step 3, we can determine the list-chromatic number of G'' in linear time.

At Step 4, we can assume that each vertex of G has a list with at least $\max\{\chi_l(G'') + 2, 5\}$ colors available. Before coloring G'', as proved in Lemma 3, for each deleted disk D and C, W for D as in Lemma 3, we can arrange lists of vertices in C in the boundary of the disk D (as well as a small buffer zone W inside the disk) so that we can apply Thomassen's results [19,22] to the vertices inside the disk D. We then give a valid coloring of G'' by the tree-width bounded method We have to extend the coloring of G'' to the vertices inside the disk. In fact, to color the vertices inside the disk, we shall use the original list of $V(D) - V(C) - W$, but possibly re-coloring some vertices in the buffer zone W (each vertex in W is on the boundary of D). Each vertex in W, except for the exceptional vertices as in Lemma 3, has a list with at least three colors available. We shall color the vertices in $V(D) - V(C)$ using the list of each vertex in W and in $V(D) - V(C) - W$). This can be done as in the second half of the proof of Lemma 3, and hence we can extend the precoloring of C to all the vertices of $V(D) - C$.

Therefore, if each list of the vertices in the disk has at least five colors in its list, then we can list-color all the vertices inside the disk. Now we just put all the colorings of deleted disks and G'' together. Lemma 3 tells us that this coloring is valid. This is clearly an additive approximation algorithm for list-coloring bounded genus graphs within 2 of the list-chromatic number.

This completes the correctness of the algorithm.

The time complexity is $O(n)$, since the tree-width bounded part takes linear time, deleting the flat disks takes linear time by Lemma 1, list-coloring the vertices in the deleted disks takes linear time by Theorems 3 and 4, and the rest of the algorithm just takes linear time. □

References

1. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Appl. Math. 23, 11–24 (1989)
2. Bodlaender, H.L.: A linear-time algorithm for finding tree-decomposition of small treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
3. Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: 46th Annual Sumposium on Foundations of Computer Science (FOCS 2005), pp. 637–646 (2005)

4. DeVos, M., Kawarabayashi, K., Mohar, B.: Locally planar graphs are 5-choosable. J. Combin. Theory Ser. B (to appear)
5. Diestel, R.: Graph Theory, 2nd edn. Springer, Heidelberg (2000)
6. Diestel, R., Gorbunov, K.Y., Jensen, T.R., Thomassen, C.: Highly connected sets and the excluded grid theorem. J. Combin. Theory Ser. B 75, 61–73 (1999)
7. Erdős, P., Rubin, Taylor: Choosability in graphs. In: Proc. West-Coast conference on Combinatorics, Graph Theory and Computing. Arcata Califonia, Congressus Numerantium vol. XXVI, pp. 125–157 (1979)
8. Feige, U., Kilian, J.: Zero-knowledge and the chromatic number. J. Comput. System Sci. 57, 187–199 (1998)
9. Fellows, M., Fomin, F., Lokshtanov, D., Rosamond, F., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA. LNCS, vol. 4616, pp. 366–377. Springer, Heidelberg (2007)
10. Gutner, S.: The complexity of planar graph choosability. Discrete Math. 159, 119–130 (1996)
11. Kawarabayashi, K., Mohar, B.: Approximating the chromatic number and the list-chromatic number of minor-closed family of graphs and odd-minor-closed family of graphs. In: Proceedings of the 38th ACM Symposium on Theory of Computing (STOC 2006), pp. 401–416 (2006)
12. Mohar, B., Thomassen, C.: Graphs on Surfaces. Johns Hopkins Univ. Press, Baltimore (2001)
13. Reed, B.: Tree width and tangles: a new connectivity measure and some applications. In: Surveys in Combinatorics, 1997, London. London Math. Soc. Lecture Note Ser, vol. 241, pp. 87–162. Cambridge Univ. Press, Cambridge (1997)
14. Reed, B.: Rooted Routing in the Plane. Discrete Applied Mathematics 57, 213–227 (1995)
15. Reed, B., Robertson, N., Schrijver, A., Seymour, P.D.: Finding disjoint trees in planar graphs in linear time. Graph structure theory (Seattle, WA, 1991). Contemp. Math, vol. 147, pp. 295–301. Amer. Math. Soc., Providenc (1993)
16. Robertson, N., Seymour, P.D.: Graph minors. V. Excluding a planar graph. J. Combin. Theory Ser. B 41, 92–114 (1986)
17. Robertson, N., Seymour, P.D.: Graph minors. XI. Circhits on a surface. J. Combin. Theory Ser. B 60, 72–106 (1994)
18. Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a planar graph. J. Combin. Theory Ser. B 62, 323–348 (1994)
19. Thomassen, C.: Every planar graph is 5-choosable. J. Combin. Theory Ser. B 62, 180–181 (1994)
20. Thomassen, C.: Color-critical graphs on a fixed surface. J. Combin. Theory Ser. B 70, 67–100 (1997)
21. Thomassen, C.: A simpler proof of the excluded minor theorem for higher surfaces. J. Combin. Theory Ser. B 70, 306–311 (1997)
22. Thomassen, C.: Exponentially many 5-list-colorings of planar graphs. J. Combin. Theory Ser. B 97, 571–583 (2007)
23. Voigt, M.: List colourings of planar graphs. Discrete Math. 120, 215–219 (1993)
24. Tuza, Z.: Graph colorings with local constraints—a survey. Discuss. Math. Graph Theory 17, 161–228 (1997)
25. Vizing: Coloring the vertices of a graph in prescribed colors. Metpdy Diskret. Anal. v Teorii Kodov i Schem 29, 3–10 (1976) (in Russian)

Asymptotically Optimal Hitting Sets Against Polynomials

Markus Bläser[1], Moritz Hardt[2], and David Steurer[2]

[1] Saarland University, Saarbrücken, Germany
[2] Princeton University, Princeton, NJ

Abstract. Our main result is an efficient construction of a hitting set generator against the class of polynomials of degree d_i in the i-th variable. The seed length of this generator is $\log D + \tilde{O}(\log^{1/2} D)$. Here, $\log D = \sum_i \log(d_i + 1)$ is a lower bound on the seed length of any hitting set generator against this class. Our construction is the first to achieve asymptotically optimal seed length for every choice of the parameters d_i. In fact, we present a nearly linear time construction with this asymptotic guarantee. Furthermore, our results extend to classes of polynomials parameterized by upper bounds on the number of nonzero terms in each variable. Underlying our constructions is a general and novel framework that exploits the product structure common to the classes of polynomials we consider. This framework allows us to obtain efficient and asymptotically optimal hitting set generators from primitives that need not be optimal or efficient by themselves.

As our main corollary, we obtain the first blackbox polynomial identity tests with an asymptotically optimal randomness consumption.

1 Introduction

Consider a class of polynomials F in n variables over some field K. A *hitting set* against F is a set of points $H \subseteq K^n$ such that no polynomial in F vanishes on all points in H. To give an example of a hitting set, consider the class F of nonzero polynomials of degree at most d_i in the i-th variable. If we fix arbitrary sets $S_i \subseteq K$ of size $d_i + 1$ assuming $|K| > d_i$, then the set $H = S_1 \times \cdots \times S_n$ is a hitting set against F of size $D = \prod_i (d_i + 1)$ (see, for instance, [1]). It is easy to argue that the *size* of this hitting set is optimal. But even for $d_i = 1$ the set is so large that we would like to have an efficient implicit representation of it. This would typically be a be a function called *hitting set generator* computable by a small circuit on $\log |S|$ inputs that serve the purpose of a random *seed*. A second observation is that there are polynomials in F vanishing on all except a single point in H. Here, it would be more desirable if the non-roots of any polynomial in F had *high density* in H. This requirement is met by the well-known *Schwartz-Zippel Lemma* [2,3,4]: If we replace each S_i by a set of $2nd_i$ points (rather than $d_i + 1$ points), then any polynomial in F vanishes on at most half of the points in H. However, the size of H increased to $(2n)^n \cdot \prod d_i$. Even in terms of $\log |S|$ this increase in size is only of lower order for large enough

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 345–356, 2008.

degree d_i. It is natural to ask whether this increase in size is inherent. It turns out the answer is negative. In this paper, we present efficient constructions of hitting sets for which the quantity $\log |S|$ is asymptotically optimal, but at the same time our hitting sets will have high density in the above sense.

The main motivation for our work is the closely related problem of *polynomial identity testing.* Here, we assume we are given access to a polynomial in some implicit representation. The problem is to distinguish the case where the given polynomial is identically zero from the case where the polynomial is a member of some class $F \subseteq K[x_1, \ldots, x_n]$. Provided with a hitting set generator against F, this can be done by picking a random seed and testing if the given polynomial is zero at the point produced by the generator. While the zero polynomial will always be zero on this point, any polynomial in F will evaluate to a nonzero value with high probability given that the hitting set has high density. Notice, these steps only require *blackbox access* to the polynomial. That is, we need not make any structural assumption about the input representation of the polynomial.

The study of polynomial identity testing was initiated by the work of DeMillo, Lipton, Schwartz and Zippel [2,3,4]. Many interesting problems have since turned out to reduce to checking polynomial identities [5,6,7,8,9,10,11]. Similarly, several results in complexity theory [12,13,14,15] involve hitting set generators against polynomials as a subroutine. What remained wide open after this initial work is the question how much randomness is required in testing polynomial identities.

There were two successful approaches: One is giving deterministic identity tests for restricted classes of arithmetic circuits [16,17,18,19]. As it turned out, testing general arithmetic circuits for identity in even subexponential deterministic time is linked to circuit lower bounds [20,16]. The other approach has been to minimize the seed length of hitting set generators against more general classes of polynomials [21,22,23,24]. In this work we continue the study of the latter problem. In this case, there is a natural lower bound on the number of random bits required that we are trying to match:

Suppose a class of polynomials $F \subseteq K[x_1, \ldots, x_n]$ contains a linear space $W \subseteq F \cup \{0\}$ of dimension at least d. Then, a dimension argument [22] shows that any hitting set generator of density $1 - \epsilon$ against F requires seed length at least $r \geq \log(d/\epsilon)$. Here, we appealed to the following definition.

Definition 1. *A* hitting set generator *of* density $\alpha > 0$ *against a class of polynomials* F *is a function* $G\colon \{0,1\}^r \to K^n$ *such that for all* $f \in F$ *we have* $\Pr[f(G(z)) \neq 0] \geq \alpha$ *where the* seed $z \in \{0,1\}^r$ *is drawn uniformly at random.*

We are interested in uniform constructions of hitting set generators. That is, we will consider classes of polynomials $F(t)$ given by a parameter t and we want to have a construction algorithm which on input of t and $\epsilon > 0$ constructs a circuit that computes a hitting set generator G of density $1 - \epsilon$ against $F(t)$. The running time of this algorithm will be measured in terms of the description length $|t|$; the runtime also serves as an upper bound on the size of the circuit.

1.1 Our Result

We introduce a general framework for obtaining efficient and asymptotically optimal constructions from primitives that need not be optimal or even efficient by themselves. Our framework requires the target class of polynomials to exhibit a typical product structure that we formalize. We exploit this structure by working with product operations on hitting set generators. Crucial primitives in our framework are hitting set generators which besides their seed have an additional source of randomness, called *random advice*. Random advice captures excess in randomness that can be shared when computing the product of two generators. Our constructions will generally be the product of several generators each working on one subset of the variables. A simple approximation algorithm determines a partition of the variables so as to minimize seed length, runtime or the required field size of our construction.

We say a polynomial f has degree $\boldsymbol{d} = (d_1, \ldots, d_n)$, if d_i is an upper bound on the degree of the i-th variable in f. We let $F(\boldsymbol{d}) \subseteq K[x_1, \ldots, x_n]$ denote the class of nonzero degree-$\boldsymbol{d}$ polynomials in n variables. We use the abbreviation $D = \prod_{i=1}^{n}(d_i + 1)$ throughout our work.

Theorem 1. *Given a degree $\boldsymbol{d}$, we can efficiently construct a hitting set generator G of density $1/2$ against $F(\boldsymbol{d})$ over any field of characteristic zero such that the seed length of G is* $\log D + O(\sqrt{\log D} \cdot \log\log D)$.

Since the quantity D is the dimension of the space $F(\boldsymbol{d}) \cup \{0\}$, the dimension lower bound implies that the seed length is asymptotically optimal for the entire family of parameters $d_1, \ldots, d_n$ where $n, d_i \in \mathbb{N}$. Our result also holds over large enough finite fields. Here, the requirement on the size of the field is roughly the same as in the Schwartz-Zippel Lemma. It is worth noting, over fields of characteristic zero, our construction does not depend on the size of the coefficients of the polynomials; the dependence on each degree d_i is only logarithmic.

We also show how to obtain a nearly linear time construction at the cost of slightly more but still asymptotically optimal seed length. More generally we have the trade-off between runtime $O(\log^{1+\delta} D)$ and seed length $\log D + O(\log^{1-\delta} D \cdot \log\log D)$ where $\delta \in (0, 1/2)$. A similar trade-off holds for the required size of finite fields.

Sparse Polynomials. We extend our work to classes of polynomials where we are given an upper bound on the number of nonzero terms. Our notion of sparsity is analogous to the previous notion of degree. We say a polynomial f has *sparsity* $\boldsymbol{m} = (m_1, \ldots, m_n)$, if f has at most m_i nonzero terms when written as a univariate polynomial in the i-th variable. For a tuple $\boldsymbol{m} = (m_1, \ldots, m_n)$ and an integer $d \in \mathbb{N}$, we define $F(\boldsymbol{m}, d)$ as the class of nonzero sparsity-$\boldsymbol{m}$ polynomials of total degree at most d. Henceforth, let $M = \prod_{i=1}^{n} m_i$.

Theorem 2. *Given sparsity $\boldsymbol{m}$ and degree $d \leq M$, we can efficiently construct a hitting set generator G of density $1/2$ against $F(\boldsymbol{m}, d)$ over any large enough finite field, say, $|K| \geq \mathrm{poly}(Mnd)$, such that the seed length of G is* $\log M + O(\sqrt{\log M \cdot \log d} \cdot \log\log M)$.

The lower bound shows that any hitting set generator of positive density against $F(\boldsymbol{m}, d)$ has seed length at least $\log M$, provided that d is sufficiently large, i.e., $d \geq \sum_{i=1}^{n} m_i$. Hence, the seed length of our generator is asymptotically optimal whenever $\log d = o(\log M/(\log\log M)^c)$ for some absolute constant c.

Theorem 3. *Given $\delta > 0$, $\boldsymbol{m}$, and d, we can construct in time polynomial in $n \log d \cdot \log^{1/\delta} M$ a hitting set generator G of density $1/2$ against $F(\boldsymbol{m}, d)$ over any field of characteristic zero such that the seed length of G is $(1+\delta)\log M + O(\log\log M + \log\log d)$.*

In the above theorem, for $\log\log d = o(\log M)$, the seed length can be made arbitrarily close in a multiplicative sense to the lower bound $\log M$ at the expense of a higher running time. This trade-off is comparable to the time-approximation trade-off in polynomial time approximation schemes (PTAS). The theorem is weaker than our other results in that it gives only quasi-polynomial time constructions of generators with asymptotically optimal seed length. However, in contrast to all previously known constructions against $F(\boldsymbol{m}, d)$, the dependence of the seed length on the total degree is not logarithmic but doubly-logarithmic. We obtain this exponential improvement by combining Descartes' Rule of Signs with an improved version of a reduction in [23].

1.2 Previous Work

The Schwartz-Zippel Lemma gives a generator against $F(\boldsymbol{d})$ of seed length $\log D + n \log n$ which is asymptotically optimal for large degree, i.e., $\log D = \omega(n \log n)$. Only recently, Bogdanov [24] obtained improvements in the case where the total degree d of the polynomials is much smaller than the number of variables n, e.g., $d = O(\log n)$. Several results are concerned with the case where $\log D$ is comparable to n. Chen and Kao [21] achieve the seed length $\sum_{i=1}^{n} \lceil \log(d_i + 1) \rceil$. Their construction works only for polynomials with integer coefficients and has some dependence on the size of those coefficients. Lewin and Vadhan [22] generalize the techniques of Chen and Kao to fields of positive characteristic. While these upper bounds are as good as $\log D$ for some configurations of the parameters, they come arbitrarily close to $\log D + n$ in general. As we think of $\log D = \Theta(n)$, this is not asymptotically optimal. In fact, speaking in terms of the size of hitting sets, this is a multiplicative excess of order 2^n. Furthermore, both constructions have a polynomial runtime dependence on each degree d_i. As soon as a single degree d_i is superpolynomial in n, their algorithms are not efficient. Notice, this range of d_i is natural even if $\log D = O(n)$. Small arithmetic circuits can compute polynomials of very high degree in a single variable.

In the arithmetic circuit model, Agrawal and Biswas [10] give a polynomial identity test that uses $\log D$ random bits. However, in this case we have no lower bound. In particular, if P = coRP, then there is a *deterministic* polynomial time arithmetic circuit identity test [4,25]. However, a particular tool introduced in their work turns out to give us hitting set generators of the optimal seed length $\log D$ over finite fields. This tool will be used and discussed later. We will see

how to achieve asymptotically the same seed length over significantly smaller finite fields (that is, $|K| > D^{o(1)}$ as opposed to $|K| > D$).

When it comes to sparse polynomials, Klivans and Spielman [23] construct a hitting set generator of seed length $O(\log(mnd))$ against the class of n-variate polynomials of total degree d and at most m nonzero terms. This is better than previous work when $\log m = o(n \log d)$. Although we use techniques from this work, our results are strictly speaking incomparable to those of Klivans and Spielman, since we consider a different class of "sparse" polynomials. However, we can think of the quantity $M = \prod m_i$ as some approximation of the number of nonzero terms m. Notice that always $M \geq m$ and in general M can be strictly larger than m. The polynomial $1 + x_1 \cdots x_n$ has only two nonzero terms, but $m_i = 2$ for all $i \in [n]$ and thus $M = 2^n$. In general, we may assume $M \geq 2^n$, since all variables with $m_i = 1$ can be fixed to an arbitrary nonzero constant.

Below we compare our results to the previous work in terms of the normalized size of the hitting set that we can efficiently represent and the time it takes to compute the implicit representation itself (neglecting constants and polylogarithmic factors). The density is fixed to be a constant, say, $1/2$. We put $q = \log |K|$ when K is finite.

Size/D	Runtime		Source
	$\mathrm{char}(K) = 0$	$\mathrm{char}(K) > 0$	
n^n	$\log D$	$\log D$	Schwartz-Zippel [4,3,2]
2^n	$\mathrm{poly}(nd)$	$\mathrm{poly}(dq)$	Chen-Kao, Lewin-Vadhan [21,22]
1	$2^{O(\log D)}$	$\mathrm{poly}(q \log D)$ for $\lvert K\rvert \geq D$	Kronecker substitution [10]
$D^{1/\log^{1/2} D}$	$\mathrm{poly}(\log D)$	—	This work, Thm. 1
$D^{o(1)}$	$\log D$	$\mathrm{poly}(q \log D)$ for $\lvert K\rvert \geq D^{o(1)}$	This work, cf. Thm. 5
Size/M			
$d \cdot M^c$	$\mathrm{poly}(\log M \cdot \log d)$	$\mathrm{poly}(q \log M)$	Klivans-Spielman [23]
$\log d \cdot M^\delta$	$\mathrm{poly}(\log^{1/\delta} M \cdot \log d)$	—	This work, Thm. 2
$d \cdot M^{\frac{\log^{1/2} d}{\log^{1/2} M}}$	—	$\mathrm{poly}(q \cdot \log M)$	This work, Thm. 3

2 Direct Products, Shared Advice, and Balanced Factors

In this section we give the technical exposition of our framework. It consists of three parts, product operations on hitting set generators and classes of polynomials, the notion of random advice, and an algorithmic approach working with these tools.

Definition 2 (Direct product). *For two generators* $G_1 \colon \{0,1\}^{r_1} \to K^{n_1}$ *and* $G_2 \colon \{0,1\}^{r_2} \to K^{n_2}$, *we define the* direct product $G_1 \otimes G_2 \colon \{0,1\}^{r_1+r_2} \to K^{n_1+n_2}$ *to be the function defined by* $G_1 \otimes G_2\ (z_1 z_2) = (G_1(z_1), G_2(z_2))$.

Clearly, if both G_1 and G_2 can be constructed efficiently, then so can the product $G_1 \otimes G_2$.

Now, suppose we have two hitting set generators with high density against two classes F_1 and F_2, respectively. We want to identify a large class of polynomials F_1F_2 against which the direct product still has high density.

Definition 3 (Schwartz-Zippel product). *Let $F_1 \subseteq K[\boldsymbol{x}_1]$ and $F_2 \subseteq K[\boldsymbol{x}_2]$ be two classes of polynomials on disjoint sets of variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively. Let $n_i = |\boldsymbol{x}_i|$. We define the* Schwartz-Zippel product *F_1F_2 to be the set of polynomials $f \in K[\boldsymbol{x}_1, \boldsymbol{x}_2]$ such that f as a polynomial in $\boldsymbol{x}_2$ has a coefficient $g \in K[\boldsymbol{x}_1]$ satisfying the following two properties: (1) g is a member of F_1, and (2) for every $\boldsymbol{a}_1 \in K^{n_1}$ with $g(\boldsymbol{a}_1) \neq 0 \in K$, the polynomial $f(\boldsymbol{a}_1, \boldsymbol{x}_2) \in K[\boldsymbol{x}_2]$ is a member of F_2.*

Intuitively, this is the same product structure required in the well-known proof of the Schwartz-Zippel Lemma. As desired, the next lemma is an immediate consequence of the definition.

Lemma 1. *Let G_1 and G_2 be two generators, and let $F_1 \subseteq K[\boldsymbol{x}_1]$ and $F_2 \subseteq K[\boldsymbol{x}_2]$ be two classes of polynomials. Suppose that G_1 has density α_1 against F_1 and G_2 has density α_2 against F_2. Then, the direct product $G_1 \otimes G_2$ has density $\alpha_1\alpha_2$ against the Schwartz-Zippel product F_1F_2.*

We introduce hitting set generators with an additional source of randomness, called *random advice*.

Definition 4 (Advised generator). *We call a function $G\colon \{0,1\}^a \times \{0,1\}^r \to K^n$ an* advised generator *with* seed length $r(G) := r$ *and* advice length $a(G) := a$. *We say an advised generator G has* quality $1-\epsilon$ *against a class F of polynomials, if the generator $G(y,\cdot)$ has density $1-\epsilon/2$ against F with probability $1-\epsilon/2$ for a randomly chosen string $y \in \{0,1\}^a$. Formally,*

$$\Pr_{y\in\{0,1\}^a}\left(\forall f \in F.\ \Pr_{z\in\{0,1\}^r}\left[f(G(y,z)) \neq 0\right] \geq 1-\tfrac{\epsilon}{2}\right) \geq 1-\tfrac{\epsilon}{2}.$$

We define the advice-less generator *$\bar{G}\colon \{0,1\}^{a+r} \to K^n$ corresponding to G to be the function defined by $\bar{G}(yz) = G(y,z)$. Here yz denotes the string obtained from y and z by concatenation.*

Fact 4. If G has quality α against F, then $\bar{G}$ has density α against F.

Definition 5 (Shared advice product). *For two advised generators $G_1\colon \{0,1\}^{a_1} \times \{0,1\}^{r_1} \to K^{n_1}$ and $G_2\colon \{0,1\}^{a_2} \times \{0,1\}^{r_2} \to K^{n_2}$ with $a = \max\{a_1, a_2\}$, we define the* shared-advice product *$G_1 \otimes G_2\colon \{0,1\}^a \times \{0,1\}^{r_1+r_2} \to K^{n_1+n_2}$ to be the function defined by $G_1 \otimes G_2\ (y, z_1z_2) = (G_1(y,z_1), G_2(y,z_2))$. Here we assume that G_i ignores all but the first a_i advice bits.*

We can compute the shared-advice product at a moderate loss of quality.

Lemma 2. *Let $\{G_i\}_{i\in[k]}$ be a set of advised generators, and let $\{F_i\}_{i\in[k]}$ be a set of classes of polynomials. Suppose the generator G_i has quality $1-\epsilon$ against F_i. Then, the shared-advice product $G = \bigotimes_i G_i$ has quality $1-k\epsilon$ against the Schwartz-Zippel product $\prod_{i\in[k]} F_i$.*

Proof. With probability $1 - k\epsilon/2$, each generator $G_i(y, \cdot)$ has density $1 - \epsilon/2$ against F_i. Condition on this event. By Lemma 1, the direct product $G(y, \cdot) = \bigotimes_i G_i(y, \cdot)$ has density $(1 - \epsilon/2)^k > 1 - k\epsilon/2$ against $\prod_i F_i$. □

Balanced Factors. The previous discussion gives rise to the following construction approach. Recall, our goal is a hitting set generator against some class of polynomials $F \subseteq K[x_1, \ldots, x_n]$. In a first step we identify classes $F_1, \ldots, F_k$ such that F is contained in the Schwartz-Zippel product $\prod_{i \in [k]} F_i$. We think of these classes F_i as factors of F. This step induces a partition of the variables into k parts. We will design advised generators G_i against each F_i, each working on one subset of the variables. Then we combine them into one generator G using the shared advice product. Our final candidate is the seedless generator $\bar{G}$.

A large number of factors k decreases the *relative* amount of advice. On the other hand, the quality of G suffers as k grows. Varying over k gives rise to interesting trade-offs. But once we fix k we want to determine a partition that minimizes the seed length of our construction.

So, suppose we can associate a weight with each variable such that the total weight of a set of variables corresponds to the length of advice needed by a generator G_i operating on this set of variables. Since we can share advice, the goal is to find a partition of the variables that distributes the weight equally among all parts. For technical reasons, we can allow that parts containing only a single variable have large weight.

Lemma 3. *Given a positive integer k and a polynomial ring $K[\boldsymbol{x}]$ with non-negative weights $w\colon [n] \to \mathbb{R}_{\geq 0}$ on the variables, we can efficiently compute a partition $(S_1, \ldots, S_k)$ of the set $S = [n]$ of variables such that each part S_i either contains only a single variable or else the total weight of the variables in S_i is at most $w(S_i) \leq 4w(S)/k$.*

Proof. There are at most $\lfloor k/2 \rfloor$ variables with $w(i) > 2w(S)/k$. Each of these variables is put in a singleton set. The remaining variables are distributed among the at least $\lceil k/2 \rceil$ remaining sets using a greedy algorithm that aims to minimize the maximum weight of a set. □

3 Polynomials of a Given Degree

We begin with the basic building blocks in our construction. For univariate polynomials we will need a simple generator that picks a random field element from a large enough range. We define the *trivial generator with seed length r* to be the generator $G\colon \{0,1\}^r \to K$ that outputs a field element that corresponds in fixed way to its seed. For example, if $\mathrm{char}(K) = 0$ or $\mathrm{char}(K) \geq 2^r$, G would output the field element corresponding to the binary number encoded by its seed, that is, $G(z_0 \cdots z_{r-1}) = \sum_{i=0}^{r-1} z_i(1+1)^i \in K$.

Proposition 1. *The trivial generator G with seed length $\log(d/\epsilon) + O(1)$ has density $1 - \epsilon$ against the class of univariate polynomials over a field K of degree at most d, provided that K has size at least d/ϵ.*

We also need the Kronecker substitution as introduced by [10] for our parameters.

Lemma 4. *Let* $\boldsymbol{d} = (d_1, \ldots, d_n) \in \mathbb{N}^n$ *and define the* Kronecker substitution *as* $\mathbf{kr}(X) = (X^{D_1}, \ldots, X^{D_n})$, *where* $D_i = \prod_{j<i}(d_j + 1)$. *Then, for every* $f \in F(\boldsymbol{d})$, *we have that* $f' = f(\mathbf{kr}(X)) \in K[X]$ *is a univariate polynomial of degree at most* $D - 1$ *such that any two distinct monomials* w *and* w' *in* f *map to distinct monomials in* f'. *In particular,* f' *is not identically zero in* $K[X]$.

Remark 1. Over finite fields of cardinality at least D/ϵ, this lemma immediately gives us a generator G of density $1 - \epsilon$ and *optimal* seed length. We simply combine the previous lemma with Proposition 1. More precisely, we generate points of the form $\mathbf{kr}(s)$ where the element s is drawn uniformly at random from a subset of the field of size D/ϵ.

Over fields of characteristic zero the bit size of $\mathbf{kr}(s)$ is at least D which is *exponential* in the desired runtime of our algorithm. It turns out, we can reduce the points of the hitting set modulo a $(\log D)$-bit prime number. However, this step seems to require at least $\log D$ additional random bits. Indeed, any method of computing an N-bit prime number in time $\mathrm{poly}(N)$ that we are aware of requires $\Omega(N)$ random bits. Computing an N-bit prime number efficiently with $o(N)$ random bits (or no random bits at all) is an intriguing open problem. Cramer's conjecture about prime gaps would imply such an algorithm. However, even if we assume the Generalized Riemann Hypothesis, the gaps between N-bit primes are only known to be bounded by $2^{N/2} \cdot \mathrm{poly}(N)$. And even if this were a density result, it would only imply an algorithm using $N/2$ random bits.

Surprisingly, we can circumvent this problem by modeling the additional $O(\log D)$ random bits as random advice. This way, we can exploit our framework in order to reduce the random advice to $o(\log D)$ bits and thus achieve an asymptotically optimal result.

Proposition 2. *Let* K *be of characteristic zero. For any degree* $\boldsymbol{d}$ *and any* $\epsilon > 0$ *we can construct a hitting set generator* G *of quality* $1 - \epsilon$ *against* $F(\boldsymbol{d})$ *in time polynomial in* $\log(D/\epsilon)$. *Furthermore,* $r(G) = \log(D/\epsilon) + O(1)$ *and* $a(G) = O(\log(D/\epsilon))$.

Proof (Sketch). First, the generator G uses its advice string y in order to obtain a number $p = p(y) > 2D/\epsilon$ such that $\Pr_y[p(y) \text{ is prime}] > 1 - \epsilon/2$. This can be done efficiently with an advice string of length $O(\log(D/\epsilon))$. An efficient algorithm for generating an N-bit prime number with high probability does not need more than $O(N + \log(1/\epsilon))$ random bits Second, G uses its seed to choose a random field element s from the range $R = \{1, \ldots, \lceil 2D/\epsilon \rceil\}$. Finally, G outputs the point $\boldsymbol{b}$ which is obtained by reducing $\mathbf{kr}(s)$ component-wise modulo p. We claim whenever $p(y)$ is a prime number, then $G(y, \cdot)$ has density $1 - \epsilon$ against $F(\boldsymbol{d})$. This can be shown by arguing since f is nonzero, we have that $f(\mathbf{kr}(X))$ vanishes on at most $D - 1$ in R points modulo p. The contrapositive of this

argument follows from a standard argument involving a Vandermonde matrix modulo p in which we observe that $f(\mathbf{kr}(s)) = f(\boldsymbol{b}) \mod p$. □

We proceed to prove a more general version of Theorem 1.

Theorem 5. *Let $\boldsymbol{d} = (d_1, \ldots, d_n)$ and $\epsilon > 0$. Then, for any $k \in \{1, \ldots, n\}$, we can efficiently construct a hitting set generator of density $1 - \epsilon$ against $F(\boldsymbol{d})$ and seed length $\log(D/\epsilon) + O(k \log(k/\epsilon)) + O(\log(D/\epsilon)/k)$. The construction works for any field of characteristic zero and any finite field of size at least $\frac{2}{\epsilon} \cdot k \cdot D^{4/k}$.*

Proof. Define the weight of the variable x_i as $w(i) = \log(d_i + 1)$. Apply Balanced Factors (Lemma 3) with the given choice of k so as to obtain a partition of the coordinates $[n]$ into sets $S_1, \ldots, S_k$. Let $\boldsymbol{d}_i$ denote the restriction of $\boldsymbol{d}$ to the coordinates in S_i. For each $i \in [k]$ we will construct an advised generator G_i against $F(\boldsymbol{d}_i)$ of quality $1 - \epsilon/2k$. If $|S_i| = 1$, then we obtain G_i from Proposition 1. In this case $a(G_i) = 0$. Whenever $|S_i| > 1$, we obtain G_i from Proposition 2 in case K is of characteristic zero. Consider the advised generator $G = \bigotimes_{i \in [k]} G_i$. This is a generator against the Schwartz-Zippel product $\prod_{i \in [k]} F(\boldsymbol{d}_i)$ which is a superset of $F(\boldsymbol{d})$. Its quality follows from Lemma 2. Notice, $r(G) = \sum_{i=1}^{k} r(G_i) = \sum_{i=1}^{k} \log(D_i) + O(k \log(k/\epsilon))$ where $D_i = \prod_{j \in S_i} (d_j + 1)$. But, $\sum_i \log(D_i) = \log D$. Hence, $r(G) = \log D + O(k \log(k/\epsilon))$. On the other hand, $a(G) = \max_i O(\log(D_i) + \log(1/\epsilon))$. But the Balanced Factors Lemma guarantees $\log(D_i) = w(S_i) \leq 4w(S)/k = 4 \log D/k$. Therefore, we obtain the desired generator by combining seed and advice of G (see Fact 4). If K is a finite field, we obtain the above G_i directly via 1. The required field size is $\max_i 2kD_i/\epsilon \leq 2kD^{4/k}/\epsilon$. □

For $k = \lceil \sqrt{\log D / \log(1/\epsilon)} \rceil$ we obtain Theorem 1.

Nearly Linear Time. The larger we choose k the more efficient is our construction. Notice the trivial generators from Proposition 1 can be constructed in time linear in their seed length. But to construct a generator from Proposition 2 we need more time. Let us say time $\bar{N}^c$ for some constant $c > 1$ where $\bar{N}$ is the length of the input parameters. In the context of the above theorem, let $N = \log D$. For simplicity fix the density to be some constant. The Balanced Factors Lemma guarantees that the seed and advice length of any advised generator used in our construction is bounded by $O(N/k)$. Hence, the time it takes to construct all advice generators will be no more than $O(k \cdot (N/k)^c) = O(N^c/k^{c-1})$. As we set $k = N/(\log N)^{c+1}$, the over all construction time becomes $\tilde{O}(N)$. The seed length remains within $(1 + o(1))$OPT. More generally, setting $k = N^{1-\delta}$ for any $\delta \in (0, 1/2)$ gives us the trade-off between time $N^{1+(c-1)\delta}$ and seed length $N + \tilde{O}(N^{1-\delta})$. It is easy to see that the exponent c need not be larger than 2. The prime number required in the proof of Lemma 4 can be computed once in cubic time (e.g., using the Rabin-Miller primality test) and passed on to all generators. Provided with this prime number, each generator can be constructed in quadratic time.

4 Polynomials with a Given Number of Nonzero Terms

Let K be a sufficiently large finite field. In this section, we give an efficient construction of hitting set generators against $F(\boldsymbol{m}, d)$ with asymptotically optimal seed length, provided $\log d$ is sufficiently smaller than $\log M$. In the previous section, our basic building blocks were generators against the target class $F(\boldsymbol{d})$ that have optimal seed length, but require some amount of advice. For the target class $F(\boldsymbol{m}, d)$, however, we do not have advised generators with optimal seed length, even if we allow an arbitrary amount of advice. Instead we will start from generators that have a close to optimal seed length against certain subclasses $F(w, W) \subseteq F(\boldsymbol{m}, d)$. Specifically, for a set of monomials W and a monomial $w \in W$, we let $F(w, W)$ be the set of polynomials over K that are in the linear span of W but not in the span of $W \setminus \{w\}$. In other words, $F(w, W)$ consists of all polynomials $f \in K[\boldsymbol{x}]$ such that w has a nonzero coefficient in f and all other monomials of f are in W. Note that all polynomials in $F(w, W)$ are nonzero.

Proposition 3. *Given $\boldsymbol{m}$, d, and $\epsilon > 0$, we can efficiently construct an advised generator G with $r(G) = \log M + O(\log nd/\epsilon)$ and $a(G) = O(\log(dM/\epsilon))$ such that G has quality $1 - \epsilon$ against every class $F(w, W) \subseteq F(\boldsymbol{m}, d)$.*

The proposition crucially relies on a multivariate to univariate reduction introduced by Klivans and Spielman [23]. This reduction maps a point $b \in K$ to the tuple $(b^{\lfloor k^{i-1} \rfloor_p})_{i \in [n]}$ where p is prime, k is a random number and $\lfloor k^{i-1} \rfloor_p$ denotes the remainder of k^{i-1} modulo p. In our case, p and k will be generated independently using the advice string (so that the substitution depends on the advice). The point b is simply drawn from a large enough range using the seed. Intuitively, the proposition then asserts that for every choice of w and W, most of the advice strings y give a generator $G(y, \cdot)$ that is dense against $F(w, W)$. Precisely, this will happen if the reduction induced by the advice string is "isolating" with respect to w and W. That is, no distinct monomial w' collides with w under the given reduction. But Klivans and Spielman showed that this isolation behavior occurs with high probability.

We remark that that possibly no single advice string above yields a generator that is dense against $F(\boldsymbol{m}, d)$.

Using Proposition 3 as our basic building block, our construction against $F(\boldsymbol{m}, d)$ essentially works as follows. First, we compute a *balanced partition* $(S_1, \ldots, S_k)$ of the coordinates $[n]$ (Lemma 3). Here we use $w(j) = \log m_j$ as the weight function. Then, from the above proposition, we obtain generators G_i that have high quality against any class $F(w_i, W_i)$ contained in $F(\boldsymbol{m}_i, d)$, where $\boldsymbol{m}_i$ is the restriction of $\boldsymbol{m}$ to the coordinates in S_i. Since the partition $(S_i)_{i \in [k]}$ was balanced, the *shared-advice product* $G = \bigotimes_i G_i$ has only advice length about $\frac{1}{k} \log M$. On the other hand, the seed length of G is close to the lower bound $\log M$. We claim that the advice-less generator $\bar{G}$ corresponding to G has high density against $F(\boldsymbol{m}, d)$. By Lemma 2, G has high quality against any product $\prod_i F(w_i, W_i)$ with $F(w_i, W_i) \subseteq F(\boldsymbol{m}_i, d)$. This implies that $\bar{G}$ has high density against the union of all such products. Finally, $\bar{G}$ has high density

against $F(\boldsymbol{m}, d)$, because every polynomial in $F(\boldsymbol{m}, d)$ is contained in one of the products $\prod_i F(w_i, W_i)$.

The details of the proof of Proposition 3 and Theorem 3 follow along the lines of our discussion and are omitted from this extended abstract. They will appear in the full version of the paper.

Over Fields of Characteristic Zero. Let K be a field of characteristic zero. Lipton and Vishnoi [26] point out the fact that a univariate polynomial with at most m nonzero terms has at most m positive rational roots over K (a consequence of Descartes' Rule of Signs).

Proposition 4. *For every $\epsilon > 0$, the trivial generator with seed length $\log(m/\epsilon) + O(1)$ has density $1 - \epsilon$ against the class of univariate polynomials with at most m nonzero terms.*

Let $F(W)$ denote the set of nonzero polynomials in the linear span of W.

Proposition 5. *Given $\epsilon > 0$, $\boldsymbol{m}$, and d, we can construct an advised generator G with $r(G) = \log M/\epsilon + O(1)$ and $a(G) = O(\log(Mn/\epsilon \cdot \log d))$ in time $2^{a(G)} = \mathrm{poly}(Mn/\epsilon \cdot \log d)$ such that G has quality $1 - \epsilon$ against every class $F(W) \subseteq F(\boldsymbol{m}, d)$.*

As in Proposition 3, the above generator first reduces the multivariate polynomial to a univariate one using the same substitution. Then it applies the generator from Proposition 4 against the resulting univariate polynomial. In contrast to the trivial generator, which was used in Proposition 3, this generator has no dependence on the degree of the polynomial. Another difference to Proposition 3 is that the construction time depends polynomially on the magnitude of the prime p number which is used to reduce the degrees in the substitution $(b^{\lfloor k^{i-1} \rfloor_p})_{i \in [n]}$. This is because over characteristic zero, the magnitude of the points blows up exponentially. In the case of sparse polynomials we do not know how to reduce the bit size of the points as we did in Proposition 2.

The doubly logarithmic dependence on d in the advice length is achieved by analyzing the effect of using a *uniformly random* prime in the substitution. This analysis improves the one given in [23] exponentially with respect to d. It is the main technical ingredient for the proof of Proposition 5. The proof of Theorem 3 follows our general framework and uses the previous two propositions as building blocks. Both proofs are omitted from this extended abstract.

References

1. Alon, N.: Combinatorial Nullstellensatz. Comb. Probab. Comput. 8(1-2), 7–29 (1999)
2. De Millo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. IPL 7 (1978)
3. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Proc. ISSAC, pp. 216–226. Springer, Berlin (1979)

4. Schwartz, J.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM 27, 701–717 (1980)
5. Chari, S., Rohatgi, P., Srinivasan, A.: Randomness-optimal unique element isolation with applications to perfect matching and related problems. SIAM J. Comput. 24(5), 1036–1050 (1995)
6. Lovász, L.: On determinants, matchings, and random algorithms. In: FCT, pp. 565–574 (1979)
7. Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. Combinatorica 7(1), 105–113 (1987)
8. Blum, M., Chandra, A.K., Wegman, M.N.: Equivalence of free boolean graphs can be decided probabilistically in polynomial time. IPL 10, 80–82 (1980)
9. Blum, M., Kannan, S.: Designing programs that check their work. J. ACM 42(1), 269–291 (1995)
10. Agrawal, M., Biswas, S.: Primality and identity testing via chinese remaindering. J. ACM 50(4), 429–443 (2003)
11. Agrawal, M., Kayal, N., Saxena, N.: PRIMES is in P. Ann. of Math (2) 160(2), 781–793 (2004)
12. Shamir, A.: IP = PSPACE. J. ACM 39(4), 869–877 (1992)
13. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM 39(4), 859–868 (1992)
14. Arora, S., Safra, S.: Probabilistic checking of proofs: a new characterization of NP. J. ACM 45(1), 70–122 (1998)
15. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM 45(3), 501–555 (1998)
16. Agrawal, M.: Proving lower bounds via pseudo-random generators. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 92–105. Springer, Heidelberg (2005)
17. Dvir, Z., Shpilka, A.: Locally decodable codes with two queries and polynomial identity testing for depth 3 circuits. SIAM J. Comput. 36(5), 1404–1434 (2007)
18. Kayal, N., Saxena, N.: Polynomial identity testing for depth 3 circuits. In: Proc. 21st CCC, pp. 9–17. IEEE, Los Alamitos (2006)
19. Shpilka, A.: Interpolation of depth-3 arithmetic circuits with two multiplication gates. In: Proc. 39th STOC, pp. 284–293. ACM, New York (2007)
20. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Comput. Complex. 13(1/2), 1–46 (2004)
21. Chen, Z.Z., Kao, M.Y.: Reducing randomness via irrational numbers. SIAM J. Comput. 29(4), 1247–1256 (2000)
22. Lewin, D., Vadhan, S.: Checking polynomial identities over any field: Towards a derandomization? In: Proc. 30th STOC, pp. 437–438. ACM, New York (1998)
23. Klivans, A., Spielman, D.A.: Randomness efficient identity testing of multivariate polynomials. In: Proc. 33th STOC, pp. 216–223. ACM, New York (2001)
24. Bogdanov, A.: Pseudorandom generators for low degree polynomials. In: Proc. 37th STOC, pp. 21–30. ACM, New York (2005)
25. Ibarra, O.H., Moran, S.: Probabilistic algorithms for deciding equivalence of straight-line programs. J. ACM 30(1), 217–228 (1983)
26. Lipton, R., Vishnoi, N.: Deterministic identity testing for multivariate polynomials. In: Proc. SODA, pp. 756–760. ACM, New York (2003)

The Smoothed Complexity of Edit Distance

Alexandr Andoni[1,⋆] and Robert Krauthgamer[2,⋆⋆]

[1] MIT
andoni@mit.edu
[2] Weizmann Institute and IBM Almaden
robert.krauthgamer@weizmann.ac.il

Abstract. We initiate the study of the smoothed complexity of sequence alignment, by proposing a semi-random model of edit distance between two input strings, generated as follows. First, an adversary chooses two binary strings of length d and a longest common subsequence A of them. Then, every character is perturbed independently with probability p, except that A is perturbed in exactly the same way inside the two strings.

We design two efficient algorithms that compute the edit distance on smoothed instances up to a constant factor approximation. The first algorithm runs in near-linear time, namely $d^{1+\varepsilon}$ for any fixed $\varepsilon > 0$. The second one runs in time sublinear in d, assuming the edit distance is not too small. These approximation and runtime guarantees are significantly better then the bounds known for worst-case inputs, e.g. near-linear time algorithm achieving approximation roughly $d^{1/3}$, due to Batu, Ergün, and Sahinalp [SODA 2006].

Our technical contribution is twofold. First, we rely on finding matches between substrings in the two strings, where two substrings are considered a match if their edit distance is relatively small, a prevailing technique in commonly used heuristics, such as PatternHunter of Ma, Tromp and Li [Bioinformatics, 2002]. Second, we effectively reduce the smoothed edit distance to a simpler variant of (worst-case) edit distance, namely, edit distance on permutations (a.k.a. Ulam's metric). We are thus able to build on algorithms developed for the Ulam metric, whose much better algorithmic guarantees usually do not carry over to general edit distance.

1 Introduction

The *edit distance* (aka *Levenshtein distance*) between two strings is the number of insertions, deletions, and substitutions needed to transform one string into the other. This distance is of key importance in several fields, such as computational biology and text processing, and consequently computational problems involving the edit distance were studied extensively, both theoretically and experimentally, see e.g. the detailed survey on edit distance by Navarro [Nav01]. Despite extensive research, the worst-case guarantees currently known for algorithms dealing with edit distance are quite poor, especially in comparison to the

⋆ Work done in part while visiting IBM Almaden Research Center.

⋆⋆ Work supported in part by a grant from the Fusfeld Research Fund.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 357–369, 2008.

Hamming distance (which is just the number of substitutions to transform one string into the other).

The most basic problem is to compute the edit distance between two strings of length d over alphabet Σ. The worst-case running time known for this problem has not improved in three decades — the problem can be solved using dynamic programming in time $O(d^2)$ [WF74], and in time $O(d^2/\log^2 d)$ when the alphabet has constant size [MP80]. Unfortunately, such near-quadratic time is prohibitive when working on large datasets, which is common in areas such as computational biology. The gold standard is to achieve a linear-time algorithm, or even sublinear in several cases, which has triggered the study of very efficient *distance estimation* algorithms – algorithms that compute an approximation to the edit distance. In particular, the best quasi-linear time algorithm, due to Batu, Ergün, and Sahinalp [BES06], achieves $d^{1/3+o(1)}$ approximation (improving over [BJKK04]), and the only known sublinear time algorithm, due to Batu, Ergün, Kilian, Magen, Raskhodnikova, Rubinfeld and Sami [BEK+03], decides whether the edit distance is $O(d^\alpha)$ or $\Omega(d)$ in time $O(d^{\max\{\alpha/2,1-2\alpha\}})$. In fact, distance estimation with sublogarithmic approximation factor was recently proved impossible in a certain model of low communication complexity [AK07]. In practice, this situation is mitigated by heuristic algorithms. In computational biology settings for instance, tools such as BLAST [AGM+90] are commonly used to solve the problem quickly, essentially by relying on heuristic considerations that sacrifice some sensitivity.

We initiate the study of the smoothed complexity of sequence alignment, by proposing a semi-random model of edit distance (the input is a worst-case instance modified by a random perturbation), and design for it very efficient approximation algorithms. Specifically, an adversary chooses two strings and a longest common subsequence of them, and every character is perturbed independently with probability $0 \leq p \leq 1$, except that every character in the common subsequence is perturbed in the same way in the two strings. Semi-random models appeared in the literature in other contexts, but to the best of our knowledge, not for sequence alignment problems; see Sect. 1.2 for more details. Our algorithms for the smoothed model approximate the edit distance within a constant factor in linear, and even sublinear time.

Why study semi-random models of sequence alignment? First, they elude the extreme difficulty posed by worst-case inputs, while avoiding the naivete of average-case (random) inputs. Using these models as a theoretical testbed for practical algorithms may lead to designing new algorithmic techniques, and/or to providing rigorous explanation for the empirical success of well-known heuristics. Second, studying algorithms for semi-random models may be viewed as an attack on the worst-case complexity. It is difficult to quantify the progress we manage to make in this direction, but we certainly achieve much better performance guarantees on a very large collection of inputs (including random inputs as an extreme case), by delineating rather general assumptions on the input, under which we have efficient algorithms.

1.1 Our Contribution

A smoothed model. Let $0 < p \leq 1$ be a *perturbation probability*. In our smoothed model for edit distance, an input consisting of two strings, x and y, is generated as follows. (A more formal description is given in Sect. 1.3.)

1. An adversary chooses two strings $x^*, y^* \in \{0,1\}^d$, and a longest common subsequence $\mathcal{A}$ of x^*, y^*.
2. Every character in x^* and y^* is replaced independently with probability p by a random bit, except that the perturbation of $\mathcal{A}$ inside x and that of $\mathcal{A}$ inside y are identical.

Results. We start by investigating the typical properties of a smoothed instance (x, y), including proving that the expected edit distance $\mathrm{ed}(x, y)$ is comparable to that of the generating strings, $\mathrm{ed}(x^*, y^*)$.

Our first result is a deterministic algorithm that approximates the edit distance within a constant factor, and its smoothed runtime complexity is near-linear. Specifically, for any desired $0 < \varepsilon < 1$, the algorithm always obtains $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$ approximation, and with high probability over the randomness in the smoothing, runs in time $O(d^{1+\varepsilon})$. For comparison, the algorithm of Batu, Ergün, and Sahinalp [BES06] for worst-case inputs requires a similar running time of $O(d^{1+\varepsilon})$ and achieves approximation $d^{(1-\varepsilon)/3+o(1)}$.

Our second result is a sublinear time algorithm for smoothed instances. Specifically, for every desired $0 < \varepsilon < 1$, the algorithm computes a $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$ approximation to $\mathrm{ed}(x, y)$ in time $O(d^{1+\varepsilon}/\sqrt{\mathrm{ed}(x, y)})$. For comparison, recall that the algorithm of Batu et al. [BEK+03] for worst-case inputs can only distinguish a polynomially large gap in the edit distance, and only at the highest regime $\Omega(d)$.

Techniques. Our algorithms are based on two new technical ideas. The first one is to find *matches* of blocks (substrings) of length $L = O(\frac{1}{p} \log d)$ between the two strings, where two blocks are considered a match if they are at a small edit distance (say εL). This same idea, but in a more heuristic form, is used by practical tools. In particular, PatternHunter [MTL02] uses such a notion of matches (to identify "seeds"), significantly improving over BLAST [AGM+90], which considers only identical blocks to be a match. Thus, our smoothed analysis may be viewed as giving some rigorous explanation for the empirical success of such techniques.

The second idea is to reduce the problem to edit distance on permutations (in worst-case), called in the literature *Ulam's distance*, or the *Ulam metric*. Here and throughout, a *permutation* is a string in which every symbol appears at most once.[1] The Ulam metric is a submetric of edit distance, but the algorithmic bounds known for it are significantly better than those for the general edit distance. In particular, Ulam's distance between permutations of length d can be

[1] It is sometimes convenient, though not crucial, to use an alphabet Σ with size larger than d. We then define a permutation as a string whose characters are all distinct.

computed in linear time $O(d \log d)$, e.g. using Patience Sorting. The main challenge we overcome is to design a reduction that distorts distances by at most a constant factor. Indeed, there is an easy reduction with distortion $L = O(\frac{1}{p} \log d)$, that follows simply because with high probability, in each string, the blocks of length L are all distinct, see [CK06, Section 3.1].

1.2 Related Work

Average-case analysis of edit distance. Random models for edit distance were studied in two contexts, for pattern matching and for nearest neighbor searching. In the former, the text is typically assumed to be random, i.e., each character is chosen uniformly and independently from the alphabet, and the pattern is usually not assumed to be random. We refer the reader to the survey [Nav01, Section 5.3] for details and references. For nearest neighbor search, the average-case model is quite similar, see [NBYST01, GP06].

Our model is considerably more general than the random strings model. In particular, the average-case analysis often relies on the fact that no short substring of the text is identical to any substring of the pattern, to quickly "reject" most candidate matches. In fact, for distance estimation, it is easy to distinguish the case of two random strings from the case of two (worst-case) strings at a smaller edit distance — just choose one random block of logarithmic length in the first string and check whether it is close in edit distance to at least one block in the second string. We achieve a near-linear time algorithm for a more adversarial model, albeit by allowing constant factor approximation.

Smoothed complexity and semi-random models. Smoothed analysis was pioneered by Spielman and Teng [ST04] as a framework aimed to explain the practical success of heuristics that do not admit traditional worst-case analysis. They analyzed the simplex algorithm for linear programming, and since then researchers investigated the smoothed complexity of several other problems, mostly numerical ones, but also some discrete problems. An emerging principle in smoothed analysis is to perform *property-preserving* perturbation [ST03], example of which is our model. Specifically, our model may be seen as performing a perturbation of x^* and y^* that preserves the common subsequence $\mathcal{A}$.

In combinatorial optimization problems, smoothed analysis is closely related to an earlier notion of semi-random models, which were initiated by Blum and Spencer [BS95]. This research program encompasses several interesting questions, such as *what algorithmic techniques are most effective (spectral methods?)*, and *when is the optimum solution likely to be unique, hard to find, or easy to certify*, see e.g. [FM97, FK01] and the references therein.

To the best of our knowledge, smoothed analysis and/or semi-random models were not studied before for sequence alignment problems.

Distance estimation. Algorithms for distance estimation are studied also in other scenarios, using different notions of efficiency. One such model is the communication complexity model, where two parties are each given a string, and they wish to estimate the distance between their strings using low communication. The

sketching model falls into this category, with further restriction to simultaneous communication protocols. A communication lower bound was recently proved in [AK07] for the edit distance metric, even on permutations, and it holds for approximations as large as $\Omega(\log d/\log\log d)$.

1.3 Preliminaries

Strings. Let x be a string of length d over alphabet Σ. A *position* in the string is an index $i \in [d]$. We write $x[i]$ or x_i to denote the symbol appearing in position i in x. Let $[i:j]$ denote the sequence of positions $(i, i+1, \ldots, j)$. We write $x[i:j]$ or $x_{[i:j]}$ for the corresponding substring of x. A *block* is a substring, often of a predetermined length.

A variant of edit distance. Let x, y be two strings. Define $\underline{\mathrm{ed}}(x,y)$ to be the minimum number of character insertions and deletions needed to transform x into y. Character substitution are not allowed, in contrast to $\mathrm{ed}(x,y)$, but a substitution can be simulated by a deletion followed by an insertion, and thus $\mathrm{ed}(x,y) \le \underline{\mathrm{ed}}(x,y) \le 2\,\mathrm{ed}(x,y)$. Observe that

$$\underline{\mathrm{ed}}(x,y) = |x| + |y| - 2\,\mathrm{LCS}(x,y),$$

where $\mathrm{LCS}(x,y)$ is the length of the longest common subsequence of x and y.

Alignments. For two strings x, y of length d, an *alignment* is a function $A : [d] \to [d] \cup \{\perp\}$ that is monotonically increasing on $A^{-1}([d])$ and satisfies $x[i] = y[A(i)]$ for all $i \in A^{-1}([d])$. Define the *length* (or size) of the alignment as $\mathrm{len}(A) = |A^{-1}([d])|$, i.e., the number of positions in x that are matched by A. Let the *cost* of A be $\mathrm{cost}(A) = 2(d - \mathrm{len}(A)) = 2|A^{-1}(\perp)|$, i.e. the number of positions in x and in y that are not matched by A. Observe that an alignment between x and y corresponds exactly to a common subsequence to x and y. Thus, if A is an alignment between x and y, then

$$\mathrm{cost}(A) = 2(d - \mathrm{len}(A)) \ge 2d - 2\,\mathrm{LCS}(x,y) = \underline{\mathrm{ed}}(x,y),$$

with equality if and only if A is an alignment of maximum length.

Block matches. Consider two strings x, y and a block length $L \in [d]$. For blocks $x_{[i:i+L-1]}$ and $y_{[j:j+L-1]}$ of length L, we let $\mathrm{ed}_A(x_{[i:i+L-1]}, y_{[j:j+L-1]})$ be the number of positions $k \in [i : i+L-1]$ such that $A(k) \notin [j : j+L-1]$. We let $\mathrm{match}(x_{[i:i+L-1]})$ denote the block $y_{[j:j+L-1]}$, where $j \in [d-L+1]$ minimizes $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]})$, breaking ties arbitrarily. For an alignment A between x and y, let $\mathrm{match}_A(i, L)$ be the block $y_{[j:j+L-1]}$, where $j \in [d-L+1]$ minimizes $\mathrm{ed}_A(x_{[i:i+L-1]}, y_{[j:j+L-1]})$. Slightly abusing notation, we sometimes let match and match_A represent the corresponding position j (instead of the substring $y_{[j:j+L-1]}$), but the distinction will be clear from the context.

Smoothed model. Let $0 \le p \le 1$, let $x^*, y^* \in \{0,1\}^d$ be two strings, and fix a maximum-length alignment A^* between x^* and y^*. Let $x, y \in \{0,1\}^d$ be the strings obtained from x^*, y^* respectively, by replacing, independently with probability p, each character with a random one, except that the positions aligned by

A^* are kept correlated. Formally, let $\pi_x \in \{0,1\}^d$ be a string where each $\pi_x[j]$ is drawn independently to be 1 with probability $p/2$ and 0 otherwise, and let π_y be defined similarly (and independently), except for position $j \in A^*([d])$, for which we set $\pi_y[j] = \pi_x[(A^*)^{-1}(j)]$. Now let $x[i] = x^*[i] + \pi_x[i]$ and $y[i] = y^*[i] + \pi_y[i]$, where addition is done modulo 2. We call the pair (x, y) a *smoothed instance* of edit distance, and denote its distribution by $\mathrm{SMOOTH}_p(x^*, y^*, A^*)$.

2 Typical Properties of Smoothed Instances

We first show that the edit distance of a smoothed instance is likely to be similar to that of the strings used to generate it. We then turn our attention to the distance between different substrings of the smoothed strings x and y. Specifically, we show that blocks of length $L = O(p^{-1} \log d)$ are likely to be far from each other in terms of edit distance, with the few obvious exceptions of overlapping blocks and blocks that are aligned via the original alignment A^*.

Besides the inherent interest, these bounds are useful in the smoothed analysis of our algorithms carried out in subsequent sections.

2.1 Edit Distance of a Smoothed Instance

Theorem 1. *Let A^* be an optimal alignment between $x^*, y^* \in \{0,1\}^d$, and fix $0 < p \le 1$. Then a smoothed instance $(x, y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$ satisfies*

$$\Pr_{(x,y)}\left[\Omega(\tfrac{p}{\log(1/p)})\,\mathrm{ed}(x^*, y^*) \le \mathrm{ed}(x, y) \le \mathrm{ed}(x^*, y^*)\right] \ge 1 - 2^{-\Omega(p)\,\mathrm{ed}(x^*,y^*)}.$$

Proof. Observe that $\mathrm{ed}(x, y) \le \mathrm{ed}(x^*, y^*)$ always holds (i.e. with probability 1). We proceed to show that with high probability, $\underline{\mathrm{ed}}(x, y) \ge \Omega(\frac{p}{\log(1/p)}) \cdot \underline{\mathrm{ed}}(x^*, y^*)$, which by Sect. 1.3 would complete the proof. We let U denote the unaligned positions in x under A^*, i.e. $U = (A^*)^{-1}(\perp)$ and $|U| = \frac{1}{2}\,\underline{\mathrm{ed}}(x^*, y^*)$.

Consider a *potential alignment* A between x and y, i.e. a map $A : [d] \mapsto [d] \cup \{\perp\}$ that is monotonically increasing on $A^{-1}([d])$, and suppose that $\mathrm{cost}(A) = 2|A^{-1}(\perp)|$ is at most $\alpha \cdot \underline{\mathrm{ed}}(x^*, y^*)$ for a small $0 < \alpha \le 1/4$ to be chosen later. For A to be an actual alignment, we must additionally have that $x[i] = y[A(i)]$ for every position $i \notin A^{-1}(\perp)$, and in particular for every position $i \in U \setminus A^{-1}(\perp)$. The number of such positions is at least $|U| - |A^{-1}(\perp)| \ge \frac{1}{2}\,\underline{\mathrm{ed}}(x^*, y^*) - \frac{1}{2}\alpha\,\underline{\mathrm{ed}}(x^*, y^*) \ge \frac{1}{4}\,\underline{\mathrm{ed}}(x^*, y^*)$. For each of them, $x^*[i]$ is perturbed independently of $y^*[A(i)]$, and thus $x[i] \ne y[A(i)]$ occurs with probability at least $p/2$. These events might not be mutually independent due to correlations via A^*, but it is easy to see that for at least half of such i, the probability is at least $p/2$ even when conditioned on earlier events (namely, $x[i]$ is independent of $x[i']$ and $y[A(i')]$ for all $i' < i$). Thus, the probability that A is an actual alignment is at most

$$\Pr\left[x[i] = y[A(i)] \text{ for all } i \in U \setminus A^{-1}(\perp)\right] \le \left(1 - \tfrac{p}{2}\right)^{\underline{\mathrm{ed}}(x^*,y^*)/8} \le e^{-p \cdot \underline{\mathrm{ed}}(x^*,y^*)/16}.$$

We will apply a union bound on all potential alignments, and thus it suffices to have an upper bound on the number of different values taken by $A_{|U}$, the restriction of A to the positions in U. We note that $A_{|U}$ is determined by the number of insertions and deletions occurring between every two successive positions in U (including the insertions and deletions before the first position in U and after the last position in U). Thus we can count the number of $A_{|U}$ as:

$$\#\{A_{|U}\} \le \binom{|U| + \frac{1}{2}\alpha\,\underline{\mathrm{ed}}(x^*,y^*)}{\frac{1}{2}\alpha\,\underline{\mathrm{ed}}(x^*,y^*)}^3 \le \left(\tfrac{e(1+\alpha)}{\alpha}\right)^{1.5\alpha\,\underline{\mathrm{ed}}(x^*,y^*)} \le (\tfrac{1}{\alpha^2})^{1.5\alpha\,\underline{\mathrm{ed}}(x^*,y^*)}.$$

Choosing $\alpha = \frac{cp}{\log(1/p)}$ for a sufficiently small constant $c > 0$, we get by a union bound $\Pr\left[\underline{\mathrm{ed}}(x,y) \le \alpha\,\underline{\mathrm{ed}}(x^*,y^*)\right] \le e^{[3\alpha\ln(1/\alpha)-p/16]\cdot\underline{\mathrm{ed}}(x^*,y^*)} \le e^{-(p/32)\,\underline{\mathrm{ed}}(x^*,y^*)}$. □

2.2 Edit Distance between Different Blocks

Our next lemma concerns typical distances between two blocks from x and y. The main technical difficulty in this lemma, beyond technique used to prove Theorem 1, is that here we consider blocks whose perturbations are correlated, e.g. overlapping blocks in the same string, thus impeding direct concentration bounds. The proof of this lemma is deferred to the full version of the article.

Lemma 1. *Let A^* be an optimal alignment between $x^*, y^* \in \{0,1\}^d$ and fix $0 < p \le 1$. Let $L \ge \frac{C}{p}\log d$ for a sufficiently large constant $C > 0$, and let $c_a, c_b, c_c > 0$ be sufficiently small constants. Then with probability at least $1 - d^{-\Omega(C)}$, a smoothed instance $(x,y) \in \mathrm{SMOOTH}_p(x^*,y^*,A^*)$ satisfies the following for all $i,j \in [d]$:*

(a). $\mathrm{ed}(x_{[i:i+L-1]}, x_{[j:j+L-1]}) \ge c_a \cdot \min\{pL, |j-i|\}$, *and similarly in y.*
(b). If $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]}) \ge L/4$, *then* $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge c_b \cdot pL$.
(c). Let $k^* = \mathrm{match}_{A^*}(i,L)$, *then*

$$\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge \min\{c_c \cdot pL, c_c \cdot |j-k^*| - \mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[k^*:k^*+L-1]})\}.$$

Furthermore, if $|j-k^*| \ge L$, then $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge c_c \cdot pL$.

3 Near-Linear Time Distance Estimation

Our first algorithm is guaranteed to give a correct answer for any input strings, but has an improved runtime for smoothed inputs, coming from a distribution $\mathrm{SMOOTH}_p(x^*,y^*,A^*)$.

Theorem 2. *For every $\varepsilon > 0$ and $p > 0$ there is a deterministic algorithm that, given as input two strings $x, y \in \{0,1\}^d$, approximates $\mathrm{ed}(x,y)$ within factor $O(\frac{1}{\varepsilon p}\log\frac{1}{\varepsilon p})$, and on a p-smoothed instance, with high probability its running time is $O(d^{1+\varepsilon})$.*

We will need three lemmas, the first two of which do not deal directly with smoothed instances and may be useful in other scenarios as well.

Lemma 2. *Consider a bipartite graph $G = ([d], [d], E)$, and call two edges $(i, j) \in E$ and $(k, l) \in E$* intersecting *if $(i - k)(j - l) \leq 0$. Then a maximum-cardinality subset of non-intersecting edges can be found in time $O(d + |E| \log d)$ by reducing the problem to Patience Sorting.*

Proof (sketch). Construct a string z of length $|E|$ as follows. Start with an empty string. For each node $i = 1, \ldots, d$, append to the end of z the list of symbols $(j_1, -i), \ldots, (j_k, -i)$ where $j_1 > j_2 > \ldots > j_k$ are the neighbors of i, i.e. $\{j_1, \ldots j_k\} = \{j \mid (i, j) \in E\}$. Then the longest increasing subsequence of z (when the order on $(j, -i)$ is lexicographic) gives a maximum-size subset of non-intersecting edges. We can find it using Patience Sorting. □

Lemma 3. *Fix an optimal alignment A between two strings $x, y \in \{0,1\}^d$. Let $L \in [d]$ divide d. Partition x into successive blocks of length L, denoted $(X_i)_{i=1}^{d/L}$, and let $Y_i = \mathrm{match}_A(X_i)$. Then $\sum_{i=1}^{d/L} \mathrm{ed}_A(X_i, Y_i) \leq 2\,\mathrm{ed}(x, y)$. Hence, for every $\varepsilon > 0$, the number of indices $i \in [d/L]$ such that $\mathrm{ed}(X_i, Y_i) > \varepsilon L$ is at most $\frac{4}{\varepsilon L} \cdot \mathrm{ed}(x, y)$.*

Proof (sketch). For $i \in [d/L]$, let MIN_i and MAX_i, respectively, be the positions of the first and last aligned symbol in X_i, i.e., $MIN_i = \min\{j \in [iL - i + 1 : iL] \mid A(j) \in [d]\}$ and similarly for MAX_i. Let u_i^x be the number of unaligned positions in $X_i = x_{[iL-L+1:iL]}$, i.e. $u_i^x = |\{j \in [iL - L + 1 : iL] \mid A(j) = \perp\}|$. Also, let u_i^y be the number of unaligned position in $y[A(MIN_i) : A(MAX_i)]$.

If $A(MAX_i) - A(MIN_i) < L$, then $\mathrm{ed}_A(X_i, Y_i) = u_i^x$. If $A(MAX_i) - A(MIN_i) \geq L$, then $\mathrm{ed}_A(X_i, Y_i) \leq u_i^x + u_i^y$. Observing that each of $\sum_i u_i^x$ and $\sum_i u_i^y$ is bounded by $\mathrm{ed}(x, y)$ proves the first part. The second part follows immediately because for such blocks $\mathrm{ed}_A(X_i, Y_i) \geq \frac{1}{2}\,\mathrm{ed}(X_i, Y_i) > \frac{1}{2}\varepsilon L$. □

Lemma 4. *Let $C > 1$ and $0 < c' < 1$ be sufficiently large and sufficiently small constants, respectively, and let $L = \frac{C}{p} \log d$. Let A^* be a maximum-length alignment between $x^*, y^* \in \{0,1\}^d$. Then for every $i \in [d]$ there is $j_i^* \in [d]$ such that, for $(x, y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$, with probability at least $1 - d^{-2}$, for all j with $|j - j_i^*| > L$ we have $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) > c'pL$.*

Proof. Take $j_i^* = \mathrm{match}_{A^*}(x^*_{[i:i+L-1]})$. If $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j_i^*:j_i^*+L-1]}) < L/4$, then for all j with $|j - j_i^*| > L$ we have $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]}) \geq L - L/4$. Otherwise, for all j we have $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j^+L-1]}) \geq L/4$. In both cases the conclusion results by applying Lemma 1(b). □

Proof (of Theorem 2). We use as a building block a near neighbor (NN) data structure under edit distance, defined as follows. Preprocess a database of m strings each of length L, so that given a query string, the algorithm returns *all* database strings at distance $\leq \varepsilon L$ from the query. We will construct such

data structure at the end, and for now assume it can be implemented with preprocessing $P(m, L)$ and query time $Q(m, L) + O(|output|)$, where *output* is the list of points reported by the query.

Let $C > 1$ and L be as in Lemma 4 and assume $\varepsilon < c'p$. Our algorithm proceeds in two stages. The first one uses the NN data structure to find, for each position in x, a few "candidate matches" in y, presumably including the correct match (under optimal alignment) for a large fraction of positions in x. The second stage views the candidate matches between positions in x and in y as the edge-set E of a bipartite graph and applies the algorithm from Lemma 2, thereby reconstructing an alignment.

Let us describe the algorithm in more detail. The first stage builds an NN data structure on all the substrings of length L in y. Then, it partitions x into successive blocks $x_{[iL-L+1:iL]}$, and for each such block, queries the NN data structure to identify all blocks in y that are within distance εL. For each such block in y, collect all the character matches between the two blocks, i.e., every zero in the block in x with every zero in the block in y, and same for ones. Let E be the resulting list of all candidate matches. The second stage simply applies Lemma 2 to this list E to retrieve an alignment between x and y. The reported approximation to $\mathrm{ed}(x, y)$ is then twice the cost of this alignment.

Next we argue the correctness of the algorithm. Consider an optimal alignment A between x and y. Lemma 3 guarantees that for all but $4\,\mathrm{ed}(x, y)/\varepsilon L$ blocks from x, there exists a corresponding block $y_{[s_i:s_i+L-1]}$ at distance $\leq \varepsilon L$. Since the algorithm detects all pairs of blocks at distance $\leq \varepsilon L$, the lemma implies that all but $O(\frac{1}{\varepsilon})\,\mathrm{ed}(x, y)$ of aligned pairs from the alignment A will appear in the list of candidate matches. The algorithm will then compute an alignment A' that has at least $d - O(\frac{1}{\varepsilon})\,\mathrm{ed}(x, y)$ aligned pairs. Concluding, the algorithm will output a distance D such that $\mathrm{ed}(x, y) \leq D \leq O(\frac{1}{\varepsilon})\,\mathrm{ed}(x, y)$.

Next we show that, with high probability, the running time of the algorithm is $O(dL \log d + P(d, L) + \frac{d}{L} \cdot Q(d, L))$. Indeed, by Lemma 4, for each query block $x_{[iL-L+1:iL]}$, only blocks $y_{[j:j+L-1]}$ for $|j - j^*_{iL-L+1}| \leq L$ can be at distance εL. Thus, for each position in $x_{[iL-L+1:iL]}$, we have at most $3L$ candidate matches, hence $|E| \leq O(dL)$. We can now conclude that the first stage runs in $O(P(d, L) + d/L \cdot (Q(d, L) + L^2))$, and the second stage runs in $O(|E| \log d) = O(dL \log d)$.

Finally, it remains to describe the NN data structure. We achieve $P(m, L) = m \log m \cdot 2^{L \cdot O(\varepsilon \log 1/\varepsilon)}$ preprocessing and $Q(m, L) = O(L)$ query time. The data structure simply prepares all answers in advance: for each string σ in the database and every string τ at edit distance $\leq \varepsilon L$ from σ, store the pair (σ, τ) in a trie keyed by τ. To query a string q, the algorithm accesses the trie using q as the key, and for every pair (η, q) returned by the trie, it reports the string η. Recall that a trie with t strings of length L, has query time $O(L)$, and preprocessing time $O(tL \log t)$. Thus, $Q(m, L) \leq O(L)$ and since there are at most $\binom{2L}{\varepsilon L}^3$ strings at edit distance $\leq \varepsilon L$ from a given string,

$$P(m, L) \leq O(m \log m \cdot \tbinom{2L}{\varepsilon L}^4 \cdot L) \leq m \log m \cdot 2^{L \cdot O(\varepsilon \log(1/\varepsilon))}.$$

The overall running time becomes $d^{1+O(p^{-1}\varepsilon \log(1/\varepsilon))}$ for $O(1/\varepsilon)$ approximation. To complete the proof, apply the above to $\varepsilon' = \Theta(\varepsilon p/\log \frac{1}{p\varepsilon})$. The resulting running time is $d^{1+\varepsilon}$ and the approximation is $O(1/\varepsilon') = O(\frac{1}{\varepsilon p}\log\frac{1}{\varepsilon p})$. □

4 Sublinear Time Distance Estimation

We now present a sublinear time algorithm that estimates the edit distance of a smoothed instance (x, y) within a constant factor. Full proof of the following theorem is deferred to the full version of this paper.

Theorem 3. *For every $\varepsilon > 0$ there is a randomized algorithm that, given as input $(x, y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$, approximates $\mathrm{ed}(x, y)$ within factor $O(\frac{1}{\varepsilon p}\log\frac{1}{\varepsilon p})$ in time $O(d^{1+\varepsilon}/\sqrt{\mathrm{ed}(x, y)})$, with success probability at least $1 - d^{-2}$ (over the randomness in the smoothing operation and the algorithm's coins).*

The high-level approach is to map the smoothed instance (x, y) to a pair of permutations (P, Q), such that the edit distance between x and y is approximately equal to the Ulam distance between P and Q. We can then estimate the Ulam distance between P and Q using an off-the-shelf sublinear algorithm for estimating Ulam distance. Specifically, we use the following algorithm of [AIK08].

Theorem 4 ([AIK08]). *There exists a randomized algorithm that, given access to two permutations P, Q of length d, approximates $\mathrm{ed}(P, Q)$ up to a constant factor in time $\tilde{O}(d/\sqrt{\mathrm{ed}(P, Q)})$, with success probability at least $2/3$.*

The first key observation is that every algorithm for Ulam distance estimation can work independently of the actual names of symbols it reads from P, Q. Specifically, when the algorithm queries one character, say position i in P, it suffices to know whether it is identical to a previously queried character $Q[j]$, and vice versa. This observation can be leveraged in the following way: if at the time that Ulam algorithm asks to query $P[i]$, the matching character $Q[j]$ (i.e. position j such that $P[i] = Q[j]$) was not queried yet, then we may "delay" revealing the actual symbol $P[i]$ until $Q[j]$ is queried (if at all, as the running time is sublinear). Hence, for the sake of analysis we may decide (by relabeling symbols) that Q is a fixed permutation, say the identity ($Q[j] = j$ for all $j \in [d]$). In the sequel, P, Q will be permutation of length d over the alphabet $\Sigma = [2d]$.

Our construction of P, Q is based on the following principle. Let A be an alignment between x and y. Then we can construct P (while Q is the identity) so that A is the *optimal* alignment between P and Q, as follows: set $P[i] = Q[A(i)]$ whenever $A(i) \in [d]$, and set $P[i] = d + i$ whenever $A(i) = \perp$. To be useful for our sublinear algorithm (when x, y is a smoothed instance), the alignment A must have cost $O(\mathrm{ed}(x, y))$, and furthermore it has to be computable "on the fly". More precisely we require that, for queried positions i, j in P, Q respectively, if $A(i) = j$, then we can detect this by only querying $x[i]$, possibly together with a small local neighborhood around $x[i]$ and around $y[j]$ (in particular, the question whether $A(i) \stackrel{?}{=} j$ is independent of the rest of the strings x, y). We term this

property "locality"; ensuring locality of the alignment A we construct is the main technical part of the proof of the theorem. We note that, for worst-case strings (x, y), constructing a near-optimal alignment A that satisfies the locality property seems hard; for a smoothed instance, on the other hand, we show this is possible, due to, in part, Lemma 1 and a stronger version of Lemma 3. For the sake of presentation, we show how to construct P directly.

4.1 Reducing Smoothed Instances to Ulam's Metric

We proceed to show how to efficiently translate a smoothed instance of edit distance into an instance of Ulam's distance, while distorting the distance by only a constant factor.

As mentioned above we set Q to be the identity permutation, and construct P as a function of x and y. (We now define the entire permutation P, even though only a sublinear portion of it will be queried by the algorithm.) The basic idea appears simple. First, we partition P into blocks of length $L = O(\frac{1}{p} \log d)$. Then, each such block $x_{[i:i+L-1]}$ we match to its closest block in y, say $y_{[j:j+L-1]}$, and define $P_{[i:i+L-1]}$ based on $Q_{[j:j+L-1]}$ and $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]})$ only. Namely, it is such that $\mathrm{ed}(P_{[i:i+L-1]}, Q_{[j:j+L-1]}) = \mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]})$. One difference from our earlier high-level description using A is that we work at the level of blocks, not single characters. But the main problem we now face is that some characters may repeat in P, because the blocks we match against in y may have overlaps. Once we fix this issue, we can apply a form of Lemma 3 to argue that $\mathrm{ed}(P, Q)$ is approximately $\mathrm{ed}(x, y)$. Unfortunately, a straightforward fix to the above issue would introduce dependencies between different blocks in P, violating the locality requirement. We thus need additional transformations of P, under which each block can locally certify it does not interfere with other blocks.

Lemma 5 (Reduction Lemma). *Fix $\varepsilon > 0$, $0 < p < 1$, and and an optimal alignment A^* between strings x^*, y^*. Let $(x, y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$ and let $L = \frac{C}{p} \log d$ for a large constant $C > 0$. Then there exists two permutations P and $Q = (1, 2, \ldots d)$ such that, with high probability, the following hold:*

Distance. $\Omega(1) \cdot \mathrm{ed}(x, y) \le \mathrm{ed}(P, Q) \le O(\frac{\log 1/p}{p} + \frac{1}{\varepsilon p}) \cdot \mathrm{ed}(x, y)$.

Locality. *For $k \in [d/L]$, $j \in [kL - L + 1 : kL]$, and $s_k = \mathrm{match}(x_{[kL-L+1:kL]})$,*

- *$P[j]$ can be computed from only s_k, $x_{[kL-2L+1:kL+L-1]}$, and $y_{[s_k-4L+1:s_k+5L-1]}$, in time $O(L^3)$.*
- *Unless $P[j] = d + j$, we have: $P[j] \in [d]$, $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[s_k:s_k+L-1]}) \le \varepsilon p L$ and $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[z:z+L-1]}) > \Omega(pL)$ for all z s.t. $|z - s_k| \ge 2L$.*

Next we describe the construction of the permutation P, deferring the proof of the distance and locality properties to the full version of the paper.

Proof (Sketch). Some positions in P will be "invalidated", which means that we set $P[j] = d + j$ for such a position j. However for the other positions we will have $P[j] \in [d]$. We construct P in three stages: first we define a permutation P^1, then we invalidate some of the positions in P^1 to obtain P^2, and again invalidate more positions to obtain the final P.

Let $L = \frac{C}{p} \log d$ denote the block length. Partition x into d/L blocks of length L, called X_k, and for each $k \in [d/L]$, let $Y_k = \text{match}(X_k)$. Let M be the set of k's such that $\text{ed}(X_k, Y_k) \leq \varepsilon pL$. Let s_k be the starting position of Y_k and let $c_k = \text{ed}(X_k, Y_k)$.

We construct P^1 by setting, for every $k \in M$, $P^1_{[kL-L+1:kL]}$ to be equal to the block $Q[s_k : s_k + L - 1]$, except that the first c_k symbols are invalidated (and thus $\text{ed}(P^1_{[kL-L+1:kL]}, Q_{[s_k:s_k+L-1]}) = c_k$). For $k \in [d/L] \setminus M$, we simply invalidate the entire block $P^1_{[kL-L+1:kL]}$.

In the second stage, we construct P^2 from P^1. We start by defining a set $F \subseteq M$. For $k \in M$, $k \geq 1$, consider a block X_k and the matching Y_k. We put k into F iff either of the following holds: (**i**) $k - 1 \notin M$, or (**ii**) $k - 1 \in M$ and $s_k - s_{k-1} > 2L$. We obtain P^2 by invalidating all blocks $P^1_{[s_k:s_k+L-1]}$ for $k \in F$.

In the third stage, to obtain from P^2 a permutation P, we invalidate all positions $j \in [d]$ such that $P^2[j]$ occurs also somewhere else in P^2 (all such symbols are invalidated concurrently). □

5 Conclusions

It seems challenging to obtain a distance estimation algorithm whose smoothed running time is quasi-linear, i.e. $d \cdot \log^{O(1)} d$, or whose approximation is independent of the smoothing parameter p at the expense of only $O(1/p)$ increase in the runtime. Perhaps more important is to extend the smoothed analysis framework to other problems, such as nearest neighbor search (or pattern matching). One may hope to match the $O(\log \log d)$ approximation that was recently obtained for the Ulam metric [AIK08].

Acknowledgments. We thank Dick Karp for useful discussions at an early stage of this research.

References

[AGM+90] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: A basic local alignment search tool. J. of Molecular Biology 215(3), 403–410 (1990)

[AIK08] Andoni, A., Indyk, P., Krauthgamer, R.: Overcoming the ℓ_1 non-embeddability barrier: Algorithms for product metrics (manuscript, 2008)

[AK07] Andoni, A., Krauthgamer, R.: The computational hardness of estimating edit distance. In: 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 724–734. IEEE, Los Alamitos (2007)

[BEK+03] Batu, T., Ergün, F., Kilian, J., Magen, A., Raskhodnikova, S., Rubinfeld, R., Sami, R.: A sublinear algorithm for weakly approximating edit distance. In: Proceedings of the Symposium on Theory of Computing, pp. 316–324 (2003)

[BES06] Batu, T., Ergün, F., Sahinalp, C.: Oblivious string embeddings and edit distance approximations. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, pp. 792–801 (2006)

[BJKK04] Bar-Yossef, Z., Jayram, T.S., Krauthgamer, R., Kumar, R.: Approximating edit distance efficiently. In: Proceedings of the Symposium on Foundations of Computer Science, pp. 550–559 (2004)

[BS95] Blum, A., Spencer, J.: Coloring random and semi-random k-colorable graphs. J. Algorithms 19(2), 204–234 (1995)

[CK06] Charikar, M., Krauthgamer, R.: Embedding the ulam metric into ℓ_1. Theory of Computing 2(11), 207–224 (2006)

[FK01] Feige, U., Kilian, J.: Heuristics for semirandom graph problems. J. Comput. Syst. Sci. 63(4), 639–673 (2001)

[FM97] Frieze, A., McDiarmid, C.: Algorithmic theory of random graphs. Random Structures and Algorithms 10(1-2), 5–42 (1997)

[GP06] Gollapudi, S., Panigrahy, R.: A dictionary for approximate string search and longest prefix search. In: 15th ACM international conference on Information and knowledge management, pp. 768–775. ACM, New York (2006)

[MP80] Masek, W.J., Paterson, M.: A faster algorithm computing string edit distances. J. Comput. Syst. Sci. 20(1), 18–31 (1980)

[MTL02] Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. Bioinformatics 18(3), 440–445 (2002)

[Nav01] Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. 33(1), 31–88 (2001)

[NBYST01] Navarro, G., Baeza-Yates, R., Sutinen, E., Tarhio, J.: Indexing methods for approximate string matching. IEEE Data Engineering Bulletin 24(4), 19–27 (2001); Special issue on Text and Databases. Invited paper

[ST03] Spielman, D.A., Teng, S.-H.: Smoothed analysis: Motivation and discrete models. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748. Springer, Heidelberg (2003)

[ST04] Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. J. ACM 51(3), 385–463 (2004)

[WF74] Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM 21(1), 168–173 (1974)

Randomized Self-assembly for Approximate Shapes

Ming-Yang Kao[1] and Robert Schweller[2]

[1] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA
kao@northwestern.edu
[2] Department of Computer Science, University of Texas-Pan American, Edinburg, TX 78539, USA
schwellerr@cs.panam.edu

Abstract. In this paper we design tile self-assembly systems which assemble arbitrarily close approximations to target squares with arbitrarily high probability. This is in contrast to previous work which has only considered deterministic assemblies of a single shape. Our technique takes advantage of the ability to assign tile concentrations to each tile type of a self-assembly system. Such an assignment yields a probability distribution over the set of possible assembled shapes. We show that by considering the assembly of close approximations to target shapes with high probability, as opposed to exact deterministic assembly, we are able to achieve significant reductions in tile complexity. In fact, we restrict ourselves to constant sized tile systems, encoding all information about the target shape into the tile concentration assignment. In practice, this offers a potentially useful tradeoff, as large libraries of particles may be infeasible or require substantial effort to create, while the replication of existing particles to adjust relative concentration may be much easier. To illustrate our technique we focus on the assembly of $n \times n$ squares, a special case class of shapes whose study has proven fruitful in the development of new self-assembly systems.

Keywords: Self-Assembly, Randomized Algorithms, Approximation Algorithms.

1 Introduction

Self-assembly is the process by which simple objects autonomously assemble into complexes. This phenomenon is common in nature and is the mechanism behind many natural occurrences such as crystal growth. Current research is particularly interested in understanding and harnessing the power of self-assembly for the purpose of massive fabrication of nanoscale devices such as computer circuits. In particular, researchers have identified DNA molecules as a promising medium in which to design controlled self-assembly systems for nanomanufacturing and biologically based computing.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 370–384, 2008.

The leading theoretical model for self-assembly is the tile assembly model introduced by Winfree [15]. The tile assembly model extends the theory of Wang tilings of the plane [14] by adding a natural mechanism for growth. Informally, particles of a self-assembly system are modeled by four-sided Wang tiles, each with a type of glue assigned to each side. The tiles float about in the plane and stick together when they bump if the affinity between touching glues is strong enough. In this way, the particles or tiles of the system self-assemble into a complex pattern or shape. The goal is then as follows: Given a target shape or pattern, design a system of tiles that will assemble into the target. The quality or efficiency of the system is then measured by how few distinct tile types are used. This measurement is motivated by the fact that each distinct type of tile must be manufactured if the system is to be implemented.

The tile model of self-assembly is primarily motivated by a DNA based implementation. Double and triple crossover DNA molecules have been designed that can act as four-sided building blocks (tiles) for DNA self-assembly [7,9]. Experimental work has been done to show the effectiveness of using these tiles to assemble DNA crystals and perform DNA computation [10,11,16,17].

Traditional work in this field has taken the approach of encoding information about the target shape into the tile types of the system [12,1,2,3,13]. In particular, Rothemund et al. [12] and Adleman et al.[1] show how the assembly of $n \times n$ squares can be assembled using $\Theta(\frac{\log n}{\log \log n})$ distinct tile types. However, the design of large sets of distinct tile types can be problematic. Many mediums of self-assembly may have small practical limitations on the number of glues or tiles that can be manufactured. Even in the most promising scenario of DNA self-assembly in which glues and tiles are encoded with long strands of DNA, there is an associated computational complexity with the design of large sets of DNA glues, as well as the likely need to redesign DNA tile structure for increasingly large tile sets.

In contrast, recent work has be done examining the possibility of encoding the complexity of the target shape outside of the particles in the system entirely. In [8], we showed that there exists a constant sized tile set that can effectively be programmed to assemble any $n \times n$ square with affecting a short sequence of temperatures in the system. Further, Demaine et. al. [6] showed how constant size tile sets can build arbitrary shapes by mixing intermediate self-assembly batches together in a sequence of stages. However, with these techniques the complexity of the target shape is contained within a sequence of laboratory steps, rather than within the system of particles itself.

In this paper we take a new approach. We do encode the complexity of the target shape within the particles of the system itself. But, we avoid the problems of encoding complexity into the distinct tile types of the system. Rather, we encode the complexity into the relative concentrations of tiles in the system. While completely encoding the complexity of the target shape into the particles of the system, we use only a $O(1)$ size set of distinct tiles. In many instances, in particular in the case of DNA, design of distinct, new particles or tiles is much more difficult than creating a large number of copies of a given particle

or tile type. Thus, the implementation of a set of relative tile concentrations for a small, universal set of tile types is potentially a more practical approach than the implementation of a completely new set of tiles. Further, for systems based on an implementation other than DNA, there are likely even more difficulties for the design of large distinct tile systems. In particular, the design of different glues based on protein-protein interactions is very limited, making a scheme with a fixed number of required glue types even more desirable.

1.1 Our Technique

Our motivating example in this paper is the assembly of $n \times n$ squares. The key to the assembly of an $n \times n$ square is the ability to create a supertile that encodes an arbitrary length $\log n$ binary string [12]. At a high level, our technique is to design a fixed size tile set that is capable of encoding, with high precision, such a string into a supertile as a function of tile concentration assignment. In particular, we design a tile set that assembles a large two dimensional array of tiles whose inner pattern of tiles constitutes a sampling of tile types. The relative number of occurrences of certain tile types within this structure provides sampled information about the relative tile concentration assignment of the tile set. By combining this sampling array with tiles capable of counting and performing arithmetic, this sampled information can be extracted into the form of a binary string displayed on the surface of the assembled supertile. Once the estimated binary string is displayed it is a straightforward extension to utilize binary counters, as in [12], to complete an $n \times n$ square where the n is specified by the estimated binary string.

To make this technique work, we need to accomplish two contradictory objectives: First, the dimensions of the sampling array supertile must be small so that they do not go beyond the width or height of the target $n \times n$ square. Second, the area of the sampling array must be large enough so that the sample obtained yields a provable accuracy guarantee from Chernoff bounds.

1.2 Our Results

Our results are summarized as follows. First, we consider the (ϵ, δ)-approximate assembly of $n \times n$ squares. A system is said to achieve an (ϵ, δ)-approximate assembly of an $n \times n$ square if the system will assemble an $n' \times n'$ square with probability at least $1 - \delta$ such that $(1 - \epsilon)n \leq n' \leq (1 + \epsilon)n$. We show that for any ϵ and δ and sufficiently large n (as a function of ϵ and δ), there exists a tile set of size $O(1)$ that achieves an (ϵ, δ)-approximate assembly of an $n \times n$ square. In contrast, the best result for the exact, deterministic assembly of $n \times n$ squares requires $O(\frac{\log n}{\log \log n})$ distinct tiles.

1.3 Related Work

Our results build on a recent technique proposed by Becker et al. [5] showing that if we wish to assemble several shapes, it is possible to reduce the total complexity needed by assembling the shapes together using a combined tile system

rather than by assembling the shapes individually using separate tile systems. Specifically, they provide $O(1)$ size tile sets that build squares and rectangles of expected dimension n, where the n is specified by tile concentrations. While this is an initial step towards concentration based control, their techniques yield a large variance and thus do not provide the precise control achieved in our work.

Previous research has considered the use of tile concentration assignments for the purpose optimizing assembly time [1]. However, they do not consider the effect concentration assignment has on the final assembled shape.

1.4 Paper Layout

The remainder of this paper is organized as follows. In Section 2 we introduce the tile assembly model. In Section 3 we discuss a preliminary, low precision technique for controlling the expected size of a target shape. In Section 4 we introduce a randomized sampling technique to assemble high precision approximations of target binary numbers. In Section 5 we apply the randomized sampling technique to the assembly of (ϵ, δ)-approximate squares to obtain our main result. In Section 6 we conclude with a discussion of future research directions.

2 Basics

2.1 Definitions

To describe the tile self-assembly model, we make the following definitions. A tile t in the model is a four sided Wang tile denoted by the ordered quadruple $(\text{north}(t), \text{east}(t), \text{south}(t), \text{west}(t))$. The entries of the quadruples are glue types taken from an alphabet Σ representing the north, east, south, and west edges of the Wang tile, respectively. A *tile system* is an ordered quadruple $\langle T, s, G, \tau, P\rangle$ where T is a set of tiles called the *tileset* of the system, τ is a positive integer called the *temperature* of the system, $s \in T$ is a single tile called the *seed* tile, G is a function from Σ^2 to $\{0, 1, \ldots, \tau\}$ called the *glue function* of the system, and P is a function denoting a probability distribution over the set of tiles in T representing the relative concentrations of each tile type. It is assumed that $G(x, y) = G(y, x)$, and there exists a `null` in Σ such that $\forall x \in \Sigma$, $G(\texttt{null}, \texttt{x}) = \texttt{0}$. In this paper we assume the glue function is such that $G(x, y) = 0$ when $x \neq y$ and denote $G(x, x)$ by $G(x)$ (see [3,4] for the effect of removing this restriction). $|T|$ is referred to as the *tile complexity* of the system. In this paper we also only consider temperature $\tau = 2$.

Define a *configuration* to be a mapping from $\mathbb{Z}^2$ to $T \bigcup \{\texttt{empty}\}$, where `empty` is a special tile that has the `null` glue on each of its four edges. The *shape* of a configuration is defined as the set of positions (i, j) that do not map to the empty tile. For a configuration C, a tile $t \in T$ is said to be *attachable* at the position (i, j) if $C(i, j) = \texttt{empty}$ and $G(\text{north}(t), \text{south}(C(i, j+1))) + G(\text{east}(t), \text{west}(C(i+1, j))) + G(\text{south}(t), \text{north}(C(i, j-1))) + G(\text{west}(t), \text{east}(C(i-1, j))) \geq \tau$. For configurations C and C' such that $C(x, y) = \texttt{empty}$, $C'(i, j) = C(i, j)$ for all $(i, j) \neq (x, y)$, and $C'(x, y) = t$ for some $t \in T$, define the act of *attaching* tile t

to C at position (x, y) as the transformation from configuration C to C'. For a given tile system $\mathbf{T}$, if a supertile B can be obtained from a supertile A by the addition of a single tile we write $A \rightarrow_T B$. Further, we denote $A \rightarrow_T$ as the set of all B such that $A \rightarrow_T B$ and $\rightarrow_T^*$ as the transitive closure of $\rightarrow_T$.

Define the *adjacency graph* of a configuration C as follows. Let the set of vertices be the set of coordinates (i, j) such that $C(i, j)$ is not empty. Let there be an edge between vertices (x_1, y_1) and (x_2, y_2) iff $|x_1 - x_2| + |y_1 - y_2| = 1$. We refer to a configuration whose adjacency graph is finite and connected as a *supertile*. For a supertile S, denote the number of non-empty positions (tiles) in the supertile by size(S). We also note that each tile $t \in T$ can be thought of as denoting the unique supertile that maps position $(0, 0)$ to t and all other positions to `empty`. Throughout this paper we will informally refer to tiles as being supertiles.

2.2 The Assembly Process

Deterministic Assembly. Assembly takes place by *growing* a supertile starting with tile s at position $(0, 0)$. Any $t \in T$ that is attachable at some position (i, j) may attach and thus increase the size of the supertile. For a given tile system, any supertile that can be obtained by starting with the seed and attaching arbitrary attachable tiles is said to be *produced*. If this process comes to a point at which no tiles in T can be added, the resultant supertile is said to be *terminally* produced. For a given shape Υ, a tile system Γ *uniquely produces* shape Υ if for each produced supertile A, there exists some terminally produced supertile A' of shape Υ such that $A \rightarrow_T^* A'$. That is, each produced supertile can be grown into a supertile of shape Υ. This definition of unique assembly is introduced in [3] and differs slightly from previous work [12,1,1] in that we do not require that a unique supertile be terminally assembled. The *tile complexity* of a shape Υ is the minimum tile set size required to uniquely assemble Υ.

Probabilistic Assembly. In addition to considering tile systems that uniquely assemble a given shape, we can consider systems that can potentially build multiple shapes, but will build one of a desired class of shapes with high probability. To study this model we can think of the assembly process as a Markov chain where each producible supertile is a state and transitions occur with non-zero probability from supertile A to each $B \in A \rightarrow_T$. For each $B \in A \rightarrow_T$, let t_B denote the tile added to A to get B. The transition probability from A to B is defined to be

$$\text{TRANS}(A, B) = \frac{P(t_B)}{\sum_{C \in A \rightarrow_T} P(t_C)}.$$

The probability that a tile system T terminally assembles a supertile A is thus defined to be the probability that the Markov chain ends in state A. Further, the probability that a system terminally assembles a shape Υ is the probability the chain ends in a supertile state of shape Υ.

3 Low Precision Technique (Line Approximation)

Becker et al. [5] first considered the assignment of tile type concentrations for the control of assembled shapes. They consider a tileset of 5 tile types whose set of terminally produced supertiles is the set of all squares. Further, they show how for any given n, a corresponding tile concentration assignment ensures that the expected dimension of the assembled square is n. A modified version of this tile set is described in Figure 1.

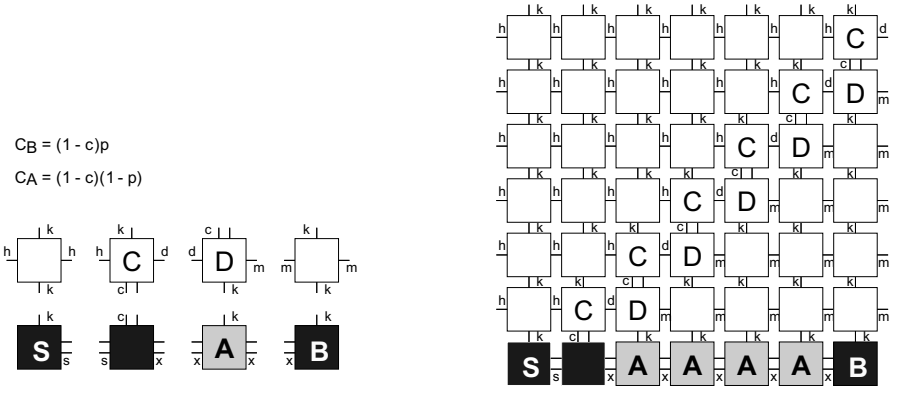


Fig. 1. With S as the seed tile, these tiles can assemble into all $n \times n$ squares with $n \geq 3$. The concentrations of tile types A and B are denoted as C_A and C_B. With $p = \frac{1}{n}$ and c being the sum of all tile concentrations other than those for A and B, the expected dimension of the assembled square is n.

The basic method of this tile system is the assembly of the line consisting of the seed, tile A, and the final tile B. As the line grows from left to right, each position can potentially be filled with an A tile, in which case the line continues to grow, or a B tile, in which case the line stops growing. If the probability of placing the B tile is p and placing the A tile is $1 - p$, then the length of the assembled line follows a geometric distribution and has expected length $\frac{1}{p}$. An expected dimension of n can thus be achieved by setting concentrations so that $p = \frac{1}{n}$.

With this method, a square is assembled whose dimension is determined by a single line of tiles whose length is a random variable with a geometric distribution. The problem with this technique is that the length of the line has a high variance. Our improved technique will create a different supertile for encoding the target dimension n. This supertile will provided an estimate for the target dimension that follows a binomial distribution instead of a geometric distribution. With this technique it is then possible to apply Chernoff bounds to achieve much more precise results.

4 The Basic Idea (Sampling Approximation)

Our goal is to assemble a supertile that encodes an x-bit binary string b. Let n be the value of the string when interpreted as a binary number. We will say that our scheme builds an (ϵ, δ)-approximate version of n if the supertile assembled encodes a value of n', $(1-\epsilon)n \leq n' \leq (1+\epsilon)n$, with probability at least $1-\delta$. For the low precision technique, the encoding of the target n is simply the length of the assembled a, b line. Note that this does not yield an (ϵ, δ)-approximate scheme for small ϵ and δ.

Our technique combines the line approximation with tiles capable of performing binary counting and binary division. First consider the line technique modified in Figure 2 so that there are two types of A tile, one of them red. Conditional on the event that one of the two types of A tiles is placed, we can control with tile concentrations that the probability of placing a red tile be some desired value q.

Now consider the random variable R denoting the sum of all red tiles placed before the final B tile is placed. Let L denote the length of the line. R then has a binomial distribution with $\mu = qL$. The goal is then to add to this construction (1) tiles capable of computing R, (2) tiles for computing L, and (3) tiles capable of performing division, in particular, tiles for computing $\frac{R}{L}$. With such tiles, we can assemble a string of tiles that encode the random variable $\frac{L}{R}$. By setting $q = \frac{1}{n}$, this random variable has expectation $\frac{L}{qL} = n$. And since R has a binomial distribution, we can apply Chernoff bounds to bound the tail probabilities of this variable when its expectation qL is high enough. While we cannot control L with high precision, we can ensure that with high probability it is sufficiently large, making the expectation of R large as well. In more detail, the tiles for the construction are as follows.

Sampling Line. The sampling line consists of the seed tile S and the three tiles in Figure 2. The expected length of the line will be $\frac{1}{p}$, and the expected ratio of the length to the number of red tiles will be n. Further, with high probability (at least $1-\delta$) the sampling line will be long enough to guarantee that this fraction is within an ϵ factor of the target n.

Double Counter. Tiles capable of counting in binary are known [12]. Straightforward modifications are possible to permit the simultaneous counting of multiple values as shown in Figure 2. Here each column of tiles represents two counts, one for the total length of the sampling line covered from the seed up until the current column, and the other the number of red tiles covered. These numbers are represented in binary with the duple label of the tile in the i^{th} row representing the ith bit of the length counter as the first coordinate and the i^{th} bit of the red tile counter as the second. Exact details for this construction are omitted in this extended abstract.

Binary Division. Work has been done to show how to perform arithmetic with self-assembly [15]. We can apply a modified set of division tiles to compute

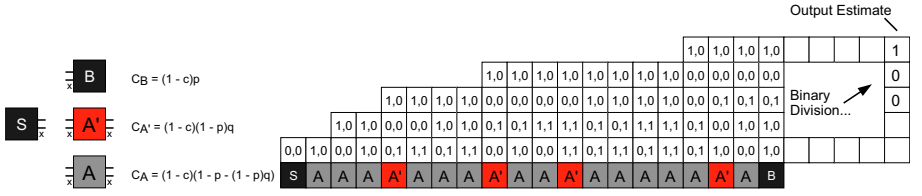


Fig. 2. With S as the seed tile, these tiles form the sampling line. The concentrations of each tile are noted as C_A, $C_{A'}$, and C_B. Here $q = \frac{1}{n}$ and p is any value at most $1 - (1 - \delta)^{\frac{\epsilon^2}{(1+\epsilon)^2 3n \ln \frac{3}{\delta}}}$. The value c denotes the total concentration from all other tiles within the tile system. These tiles create a line, sampling a red tile with a density of $1/n$. A binary counter sums up the total length, as well as the total number of red tiles. These two values are then divided to yield an estimate for n.

$\frac{L}{R}$ from the values L and R encoded in the double counter. Details are omitted in this extended abstract.

Theorem 1. *For any given $\epsilon, \delta \leq 0$ and positive integer n, the sampling approximation tile system creates an estimate whose value n' is such that $(1 - \epsilon)n \leq n' \leq (1 + \epsilon)n$ with probability at least $1 - \delta$.*

Proof. Let L be the random variable denoting the length of the sampling line and the R be the random variable denoting the number of red tiles in the line. From the assigned tile concentrations, L is has a geometric distribution with mean $\frac{1}{p}$ and R has a binomial distribution with mean qL. By applying Chernoff bounds we get that

$$P[R > (1 + \frac{\epsilon}{1+\epsilon})qL] < \exp(-qL\epsilon^2/3(1+\epsilon)^2) \tag{1}$$

and

$$P[R < (1 - \frac{\epsilon}{1+\epsilon})qL] < \exp(-qL\epsilon^2/2(1+\epsilon)^2). \tag{2}$$

Our goal is now to ensure that L is large enough so that the above equations are bounded by $(1 - \frac{\delta}{3})$. Since L has a geometric distribution, for any positive integer x we have that $P[L > x] = (1 - p)^x$. Thus, for $x = \frac{(1+\epsilon)^2 3n \log \frac{3}{\delta}}{\epsilon^2}$ and $p = 1 - (1 - \frac{\delta}{3})^{\frac{\epsilon^2}{(1+\epsilon)^2 3n \log \frac{3}{\delta}}}$, we get that $(1 - p)^x = 1 - \frac{\delta}{3}$. Thus,

$$P[L > \frac{(1+\epsilon)^2 3n \log \frac{3}{\delta}}{\epsilon^2}] = 1 - \frac{\delta}{3}. \tag{3}$$

Further, by plugging x into the right hand sides of equations (1) and (2), we get that $(1 - \frac{\epsilon}{1+\epsilon})qL \leq R \leq (1 + \frac{\epsilon}{1+\epsilon})qL$ with probability at least $1 - \frac{2\delta}{3}$ when $L \geq x$. Combining this with the probability bound from equation (3) yields the following

$$(1 - \frac{\epsilon}{1+\epsilon})qL \leq R \leq (1 + \frac{\epsilon}{1+\epsilon})qL \text{ with probability at least } 1 - \delta. \tag{4}$$

Finally, now consider the tile system's estimate of $\frac{L}{R}$. From equation (4) we get that with probability at least $1-\delta$

$$\frac{L}{(1+\frac{\epsilon}{1+\epsilon})qL} \leq \frac{L}{R} \leq \frac{L}{(1-\frac{\epsilon}{1+\epsilon})qL} \tag{5}$$

$$\Rightarrow \quad \frac{(1+\epsilon)n}{1+2\epsilon} \leq \frac{L}{R} \leq (1+\epsilon)n \tag{6}$$

$$\Rightarrow \ (1-\epsilon)n \leq \frac{L}{R} \leq (1+\epsilon)n. \tag{7}$$

This completes the proof of Theorem 1. □

5 $n \times n$ Squares

In this section we apply the basic technique of sampling approximation to the assembly of approximate $n \times n$ squares. As shown in [12], there exists a general set of *square building* tiles of constant size that, given a supertile that encodes a length $\log n$ binary string n' (the string encoded is not exactly n but uniquely identifies it) will uniquely assemble into an $n \times n$ square. The key is then to efficiently build such a supertile. This can be done trivially with $\log n$ distinct tile types, while a more efficient method yields $O(\frac{\log}{\log \log n})$ tile types [1], which is optimal for almost all n. We instead will use the sampling approximation to achieve the result with only $O(1)$ total tiles.

However, there is a problem with directly using the line approximation from Section 4. To approximate a value n' with small values of ϵ and δ, the length of the sample line must be many times larger than n'. As n' can be almost as large as n, the length of the estimation line will far exceed the width n of the square and will thus fail to build the square.

However, we can get around this problem by taking advantage of the extra dimension of the square. That is, while the width of the square is n, there is actually n^2 space within the bounds of the square. It is plausible then that an approximation line of length many times n, broken up into pieces of length at most n, could fit within the boundary of an $n \times n$ square. We do exactly this with what we call the *approximation frame*.

5.1 Approximation Frame

The basic idea for the approximation frame is to use two separate line approximations, one line growing east of the seed and another growing north. Then, for each of the tiles in the vertical line, create a sampling line that grows east up to the length of the initial horizontal line approximation. By further providing sufficient space between each tile in the vertical approximation line, we can use counter tiles to sum the number of red tiles in each approximation line. Each individual sum for each sampling line can then be summed to gain a total number of red tiles, as well as the total length, to compute an estimate for the target n'.

The key to this technique is two fold. First, we need to be able to say with high probability that both dimensions of the frame are small enough so that they do not exceed the width n of the target square. Second, we need that with high probability the total length of all the approximation lines, which is the length of the initial horizontal approximation line multiplied by the length of the initial vertical approximation line, is sufficiently large to provide an accurate estimate of the target n'. In this section we show that for any given ϵ and δ this is possible for large enough n'.

In Figure 3 the high level structure of the approximation frame is described. Starting from the seed tile, a sampling line grows east with length X, which has expectation $\frac{1}{p_x}$, and the expected ratio of red tiles to the total length equal to $q = \frac{1}{n'}$. A double counter computes both the length and the number of red tiles, as in the basic sampling line. However, in this case the final value of the counter seeds a new row of tiles that grows back in the direction of the seed, again sampling a red tile with probability q. This return row places the black S' tile which seeds a new approximation line, this time growing north. However, each placement of a C tile in this approximation line is buffered by a distance of $\log X$ tiles, i.e., the length of the counter that computes the length X. It is straightforward to maintain this buffer by using the initial length between S and S' as a *yardstick*. A diagonal growth of tiles, depicted as the grey tiles in the figure, can translate a vertical length into a horizontal length, and vice versus. With these tiles the original length between S and S' can be placed before each C tile until the final D is placed. Let Y denote the length of this approximation

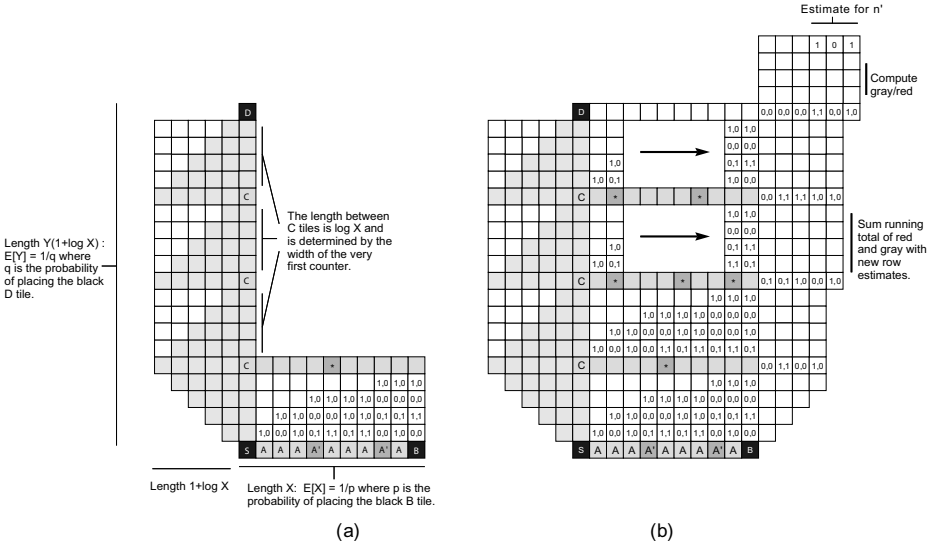


Fig. 3. This is a high level schemata for how the estimation frame can be used to build $n \times n$ squares

line when only counting the C and D tiles. Thus, the total length is $Y \log X$ with expectation $\frac{\log X}{p_y}$ if p_y is the probability of placing D instead of C.

For each C, a new sampling line is constructed, with the distinction that the length is deterministically equal to the initial random length X. For each of these rows a double counter is used to calculate the length and number of red tiles. Since the maximum bits of the double counter is the same as with the initial count, we are guaranteed to have enough room for the counter to complete.

Finally, tiles for summation are used along the east side of the frame to calculate the total sum of red tiles as well as the total length of the sampling lines (which is XY). A final division is performed to compute the ratio of length to red tiles for the estimate.

Dimensions of Frame. As initially mentioned in this section, one potential problem with this construction is that the estimation frame can grow to exceed n in one of its dimensions, making the assembly of any square of dimension at most n impossible. The dimensions of the frame are as follows.

Horizontal Dimension. The length of the sampling line is X, the diagonal tiles for propagating the height of the double counter extend to at most and additional $\log X$, and the tiles for summation of all double counters can be implemented to extend to at most $\log XY$. The total length is thus at most $X + 2 \log X + \log Y$.

Vertical Dimension. The total height of the frame is at most $Y \log X$ for the line approximation, plus the space used to compute the ratio of the total sample length divided by the number of red tiles. The exact amount of space for division depends on the implementation, but a straightforward implementation of division using using tiles for simulating the execution of a Turing machine can divide an x bit input using space within an $x \times 3x^3$ box. Growing the larger dimension in the vertical direction and noting that the sum is at most $\log^3 XY$ yields a total length of $Y \log X + (\log XY)^3$.

5.2 Approximate Squares

Theorem 2 is the main theorem of this paper.

Theorem 2. *For any given* $\epsilon, \delta \leq 1$ *and* $\frac{n}{\log n} \geq c \frac{log^3(1/\frac{\delta}{4})(1+\epsilon)^2}{\log \frac{1}{1-\frac{\delta}{4}}\epsilon^2}$, *c a constant, there exists a tile system that with probability at least* $1 - \delta$ *will assemble an* $m \times m$ *square with* $(1-\epsilon)n \leq m \leq (1+\epsilon)n$.

Proof. Consider a sufficiently large n. Let $n' = n - 2 \log n$ and consider the approximation frame tile system to estimate n'. That is, set tile concentrations such that the probability q of placing a red tile for each of the sampling lines in the frame is $\frac{1}{n'}$. Let X denote the horizontal length of the sampling line growing east of the seed tile, and let Y denote the length of the line growing north of the seed tile (not counting the size $\log X$ buffer between each C tile).

First, we want to ensure that X and Y are short enough so that the entire frame will fit within the boundary of the target square. These bounds are

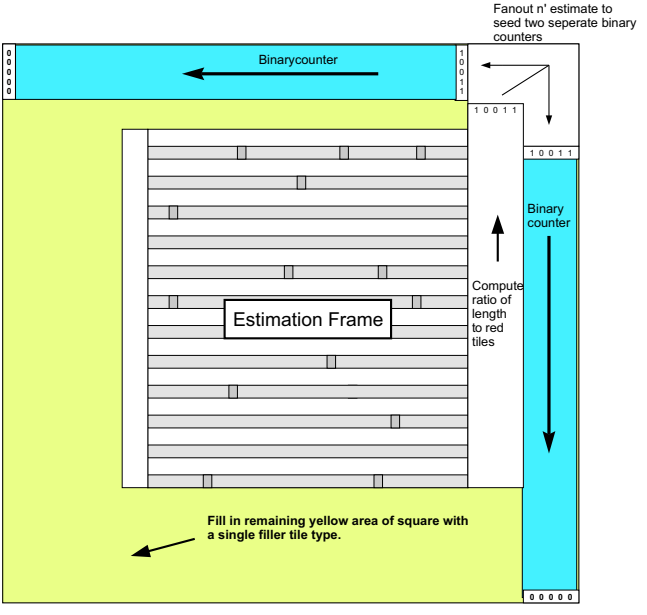


Fig. 4. This is a high level schemata for how the estimation frame can be used to build $n \times n$ squares

$n_x = n - 4\log n$ for X and $n_y = \frac{n}{\log n} - 8\log^2 n - 1$ for Y. We also want to ensure that the total length of all sampling lines is sufficiently long to guarantee a good approximation. The following probability assignments ensure both constraints are met with high probability. Let the probability of placing the B and D tiles be $p_x = 1 - \frac{\delta}{4}^{\frac{1}{n_x}}$ and $p_y = 1 - \frac{\delta}{4}^{\frac{1}{n_y}}$, respectively.

First we show that $X \leq n_x$ and $Y \leq n_y$ with probability at least $1 - \frac{\delta}{2}$. Since X has a geometric distribution, we have that

$$P[X > n_x] = (1-p)^{n_x}, \tag{8}$$

$$= (1 - (1 - (\frac{\delta}{4})^{\frac{1}{n_x}}))^{n_x}, \tag{9}$$

$$= \frac{\delta}{4} \text{ and,} \tag{10}$$

$$P[Y > n_y] = \frac{\delta}{4}. \tag{11}$$

Having shown that the frame will fit within the dimensions of the target square, we now show that the total length of all sampling lines XY will be sufficiently large under the assumption of a large n'.

Since X has the geometric distribution, we have that

$$P[X > n_x \frac{\log(1-\frac{\delta}{4})}{\log(\frac{\delta}{4})}] = (1-(1-(\frac{\delta}{4})^{\frac{1}{n_x}}))^{n_x \frac{\log(1-\frac{\delta}{4})}{\log(\frac{\delta}{4})}} \tag{12}$$

$$= 1 - \frac{\delta}{4} \quad \text{and,} \tag{13}$$

$$P[Y > n_y \frac{\log(1-\frac{\delta}{4})}{\log(\frac{\delta}{4})}] = 1 - \frac{\delta}{4}. \tag{14}$$

Now consider the total length of the sample lines XY. With a similar analysis as is done for Theorem 1 we know that an (ϵ, δ)-approximation of n' can be achieved if the total length of the sample lines is

$$XY \geq \frac{3n'(1+\epsilon)^2 \log \frac{1}{\delta/4}}{\epsilon^2}. \tag{15}$$

From equations 13 and 14, we have that with probability greater than $1 - \frac{\delta}{2}$

$$XY \geq n_x n_y (\frac{\log(1-\frac{\delta}{4})}{\log \frac{\delta}{4}})^2 \tag{16}$$

$$= \Omega(\frac{n^2}{\log n})(\frac{\log(1-\frac{\delta}{4})}{\log \frac{\delta}{4}})^2. \tag{17}$$

Combining equation 17 with inequality 15 shows that an (ϵ, δ)-estimate can be achieved in the case for a constant c where

$$\frac{n}{\log n} \geq c \frac{\log^3(1/\frac{\delta}{4})(1+\epsilon)^2}{\log \frac{1}{1-\frac{\delta}{4}}\epsilon^2}. \tag{18}$$

Given an (ϵ, δ)-approximation of n', it is straightforward to add a group of tiles that fanout the n' estimate into two identical copies which seed a binary counter. Each binary counter will begin counting down (decrementing rather than incrementing) until 0 is reached. The two counters then form the two axis of an $n' + 2\log n'$ square. Given this, it is straight forward to fill in the rest of the square at temperature 2 with a constant number of tiles. Thus, the approximation accuracy for the estimate n' yields a corresponding approximation accuracy for the dimension of the square assembled and thus proves Theorem 2. □

6 Future Work

This work is a preliminary theoretical look into the feasibility of precisely controlling assembled structures by manipulating tile concentrations. There are many directions for continued research.

One direction is to consider the probabilistic assembly of approximate scalings of general shapes. That is, given a tile system that assembles a given shape,

modify the system so that a factor n magnification of the input shape is assembled. Along this line, it is should be possible to achieve (ϵ, δ)-approximate scalings with no increase in tile complexity for a large class of assembled shapes.

Another direction for future work is the design of probabilistic self-assembly systems for the assembly of exact shapes. That is, how can a tile system be designed so that a target shape is assembled exactly (no ϵ error factor) with high probability? It would be interesting to know if an alternate technique, or an improved version of our technique, could achieve this for $n \times n$ squares. An alternate approach would be to consider 3 dimension assemblies, such as $n \times n \times n$ cubes. In such a case, our approximation frame would be able to achieve a much higher asymptotic accuracy in n. With a tighter analysis, it is plausible that this could yield the exact assembly of cubes with high probability.

Yet another research direction involves the assembly of general shapes. In [13] a technique for the assembly of general scaled shapes is presented. To work, this technique first requires the assembly of a binary string of tiles encoding a description of the target shape. An interesting direction would be to combine this technique with the approximation frame from this paper to provide the input string assembled with $O(1)$ tile complexity. As the input string for the general shape system must be exact, a key step would be to ensure that the approximation frame is large enough to provide enough binary digits that contain no error (with high probability).

Finally, as our work here is theoretical, an important next step is simulation and lab experimentation to test and validate our results. Such experiments will likely provide key insights regarding how our model and technique can be improved.

References

1. Adleman, L., Cheng, Q., Goel, A., Huang, M.: Running time and program size for self-assembled squares. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, pp. 740–748 (2001)
2. Adleman, L., Cheng, Q., Goel, A., Huang, M., Kempe, D., Espanes, P., Rothemund, P.: Combinatorial optimization problems in self-assembly. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 23–32 (2002)
3. Aggarwal, G., Cheng, Q., Goldwasser, M.H., Kao, M.-Y., de Espanes, P.M., Schweller, R.T.: Complexities for generalized models of self-assembly. SIAM Journal on Computing 34, 1493–1515 (2005)
4. Aggarwal, G., Goldwasser, M.H., Kao, M.-Y., Schweller, R.T.: Complexities for generalized models of self-assembly. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 880–889 (2004)
5. Becker, F., Remila, E., Rapaport, I.: Self-assemblying classes of shapes with a minimum number of tiles, and in optimal time. In: Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (2006)
6. Demaine, E., Demaine, M., Fekete, S., Ishaque, M., Rafalin, E., Schweller, R., Souvaine, D.: Staged self-assembly: Nanomanufacture of arbitrary shapes with O(1) glues. In: Proceedings of the 13th International Meeting on DNA Computing (2007)

7. Fu, T.-J., Seeman, N.C.: DNA double-crossover molecules. Biochemistry 32, 3211–3220 (1993)
8. Kao, M.-Y., Schweller, R.: Reducing tile complexity for self-assembly through temperature programming. In: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithms, pp. 571–580 (2006)
9. LaBean, T.H., Yan, H., Kopatsch, J., Liu, F., Winfree, E., Reif, H.J., Seeman, N.C.: The construction, analysis, ligation and self-assembly of DNA triple crossover complexes. J. Am. Chem. Soc. 122, 1848–1860 (2000)
10. Lagoudakis, M.G., Labean, T.H.: 2D DNA self-assembly for satisfiability. In: Proceedings of the 5th DIMACS Workshop on DNA Based Computers, June 26 1999, pp. 459–468 (1999)
11. Reif, J.: Local parallel biomolecular computation. In: Proceedings of the 3rd Annual DIMACS Workshop on DNA Based Computers, June 23-26 (1997)
12. Rothemund, P., Winfree, E.: The program-size complexity of self-assembled squares. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, pp. 459–468 (2000)
13. Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes. In: Tenth International Meeting on DNA Computing, pp. 344–354 (2005)
14. Wang, H.: Proving theorems by pattern recognition. Bell System Technical Journal 40, 1–42 (1961)
15. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Institute of Technology, Pasadena (1998)
16. Winfree, E., Liu, F., Wenzler, L., Seeman, N.: Design and self-assembly of two-dimensional DNA crystals. Nature 394, 539–544 (1998)
17. Winfree, E., Yang, X., Seeman, N.C.: Universal computation via self-assembly of DNA: Some theory and experiments. In: Proceedings of the 2nd International Meeting on DNA Based Computers, June 10-12 1996, pp. 191–213 (1996)

Succinct Data Structures for Retrieval and Approximate Membership* (Extended Abstract)

Martin Dietzfelbinger[1] and Rasmus Pagh[2]

[1] Technische Universität Ilmenau, 98684 Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de
[2] IT University of Copenhagen, 2300 København S, Denmark
pagh@itu.dk

Abstract. The *retrieval problem* is the problem of associating data with keys in a set. Formally, the data structure must store a function $f: U \to \{0,1\}^r$ that has specified values on the elements of a given set $S \subseteq U$, $|S| = n$, but may have any value on elements outside S. All known methods (e. g. those based on perfect hash functions), induce a space overhead of $\Theta(n)$ bits over the optimum, regardless of the evaluation time. We show that for any k, query time $O(k)$ can be achieved using space that is within a factor $1 + e^{-k}$ of optimal, asymptotically for large n. The time to construct the data structure is $O(n)$, expected. If we allow logarithmic evaluation time, the additive overhead can be reduced to $O(\log \log n)$ bits whp. A general reduction transfers the results on retrieval into analogous results on *approximate membership*, a problem traditionally addressed using Bloom filters. Thus we obtain space bounds arbitrarily close to the lower bound for this problem as well. The evaluation procedures of our data structures are extremely simple. For the results stated above we assume free access to fully random hash functions. This assumption can be justified using space $o(n)$ to simulate full randomness on a RAM.

1 Introduction

Suppose we want to build a data structure that is able to distinguish between girls' and boys' names, in a collection of n names. Given a string not in the set of names, the data structure may return any answer. It is clear that in the worst case this data structure needs at least n bits, even if it is given access to the list of names. The previously best solution that does not require the set of names to be stored uses more than $1.22n$ bits. Surprisingly, as we will see in this paper, $n + o(n)$ bits is enough, still allowing fast queries. If "global" hash functions, shared among all data structures, are available the space usage drops all the way to $n + O(\log \log n)$ bits whp. This is a rare example of a data structure with

* The main ideas for this paper were conceived while the authors were participating in the 2006 Seminar on Data Structures at IBFI Schloss Dagstuhl, Germany.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 385–396, 2008.

non-trivial functionality and a space usage that essentially matches the entropy lower bound.

1.1 Problem Definition and Motivation

The *dictionary problem* consists of storing a set S of n keys, and r bits of data associated with each key. A *lookup* query for x reports whether or not $x \in S$, and in the positive case reports the data associated with x. We will denote the size of S by n, and assume that keys come from a set U of size $n^{O(1)}$. In this paper, we restrict ourselves to the *static* problem, where S and the associated data are fixed and do not change. We study two relaxations of the static dictionary problem that allow data structures using less space than a full-fledged dictionary:

- The *retrieval problem* differs from the dictionary problem in that the set S does not need to be stored. A retrieval query on $x \in S$ is required to report the data associated with x, while a retrieval query on $x \notin S$ may return any r-bit string.
- The *approximate membership problem* consists of storing a data structure that supports membership queries in the following manner: For a query on $x \in S$ it is reported that $x \in S$. For a query on $x \notin S$ it is reported with probability at least $1 - \varepsilon$ that $x \notin S$, and with probability at most ε that $x \in S$ (a "false positive"). For simplicity we will assume that ε is a negative power of 2.

Our model of computation is a unit cost RAM with a standard instruction set. For simplicity we assume that a key fits in a single machine word, and that associated values are no larger than keys. Some results will assume free access to fully random hash functions, such that any function value can be computed in constant time. (This is explicitly stated in such cases.)

The approximate membership problem has attracted significant interest in recent years due to a number of applications, mainly in distributed systems and database systems, where false positives can be tolerated and space usage is crucial (see [4] for a survey). Often the false positive probability that can be tolerated is relatively large, say, in the range $1\% - 10\%$, which entails that the space usage can be made much smaller than what would be required to store S exactly.

The retrieval problem shows up in situations where the amount of data associated with each key is small, and it is either known that queries will only be asked on keys in S, or where the answers returned for keys not in S do not matter. As an example, suppose that we have ranked the URLs of the World Wide Web on a 2^r step scale, where r is a small integer. Then a retrieval data structure would be able to provide the ranking of a given URL, without having to store the URL itself. The retrieval problem is also the key to obtaining a space-optimal RAM data structure that answers range queries in constant time [1,19].

1.2 Previous Results

Approximate membership. The study of approximate membership was initiated by Bloom [2] who described the *Bloom filter* data structure which provides an

elegant, near-optimal solution to the problem. Bloom showed [2,4] that a space usage of $n \log_2(1/\varepsilon) \log_2 e$ bits suffices for a false positive probability of ε. Carter *et al.* [7] showed that $n \log_2(1/\varepsilon)$ bits are required for solving the approximate membership problem when $|U| \gg n$ (see also [12] for details). Thus Bloom filters have space usage within a factor $\log_2 e \approx 1.44$ of the lower bound, which is tight.

Another approach to approximate membership is *perfect hashing*. A function $h\colon U \to [n]$ is a minimal perfect hash function for S if it maps the keys of $S \subseteq U$ bijectively to $[n] = \{0, \ldots, n-1\}$, where $n = |S|$. Hagerup and Tholey [16] showed how to store a minimal perfect hash function h in a data structure of $n \log_2 e + o(n)$ bits such that h can be evaluated on a given input in constant time. This space usage is optimal. Now store an array of n entries where, for each $x \in S$, entry $h(x)$ contains a $\log_2(1/\varepsilon)$-bit hash signature $q(x)$. When looking up a key x, we answer "$x \in S$" if and only if the hash signature at entry $h(x)$ is equal to $q(x)$. The origin of this idea is unknown to us, but it is described e.g. in [4]. The space usage for the resulting data structure differs from the lower bound $n \log_2(1/\varepsilon)$ by the space required for the minimum perfect hash function, and improves upon Bloom filters when $\varepsilon \leq 2^{-4}$ and n is sufficiently large.

Mitzenmacher [18] considered the *encoding* problem where the task is to represent and transmit an approximate set representation (no fast queries required). However, even in this case existing techniques have a space overhead similar to that of the perfect hashing approach.

Retrieval. The retrieval problem has traditionally been addressed through the use of perfect hashing. Using the Hagerup-Tholey data structure yields a space usage of $nr + n \log_2 e + o(n)$ bits with constant query time. Recently, Chazelle *et al.* [8] presented a different approach to the problem. Each key is associated with $k = O(1)$ locations in an array with $O(n)$ entries of r bits. The answer to a retrieval query on x is found by combining the values of entries associated with x, using bit-wise XOR. In place of the XOR operation, any abelian group operation may be used. In fact, this idea was used earlier by Majewski, Wormald, Havas, and Czech [17] and by Seiden and Hirschberg [23] to address the special case of order-preserving minimal perfect hashing. It is not hard to see that these data structure in fact solve the retrieval problem. The main result of [17] is that for $k = 3$ a space usage of around $1.23nr$ bits is possible, and this is the best possible using the construction algorithm of [8,17] (other values of k give worse results). Though this space usage is larger than when using perfect hashing, asymptotically for large n, the simplicity and the lack of lower order terms in the space usage that may dominate for small n makes it interesting from a practical viewpoint. A particular feature is that (like for Bloom filters) all memory lookups are nonadaptive, i.e., the memory addresses can be determined from the query only. This can be exploited by modern CPU architectures that are able to parallelize memory lookups (see e.g. [24]). In fact, Chazelle *et al.* also show how approximate membership can be incorporated into their data structure by extending array entries to $r + \log_2(1/\varepsilon)$ bits. This generalized data structure is called a *Bloomier filter*. Again, the space usage is a constant factor higher, asymptotically, than the solution based on perfect hashing.

Approximate membership by retrieval. We observe that there exists a simple reduction from approximate membership problem to the retrieval problem. Though it is used in the approximate membership data structure based on perfect hashing, we do not believe that it has been stated in this generality before (for a proof, see the full version of this paper [12]).

Observation 1. *Assuming free access to fully random hash functions, any static retrieval data structure can be used to implement an approximate membership data structure having false positive probability* 2^{-r}*, with no additional cost in space, and* $O(1)$ *extra time.*

If we drop the assumption of fully random hash functions being provided for free, only a $o(1)$ term has to be added to the false positive probability (for details see [12]).

Parallel work. Immediately after a draft full version of this work appeared ([12], March 26, 2008), we were informed that E. Porat had independently worked on the same problems. His results are described in a report ([22], April 11, 2008). He also uses linear equations, however without restricting the weight of rows. The resulting problems with construction and evaluation time are circumvented by using a two-level splitting technique similar to the one used in [16]. The space usage is asymptotically smaller than in our Theorem 1(a).

1.3 New Contributions

Our main contribution shows that the approach of Chazelle *et al.* [8], Majewski *et al.* [17], and Seiden and Hirschberg [23] can be used to achieve space for retrieval that is very close to the lower bound, while retaining efficient evaluation.

Theorem 1. *There exist data structures for the retrieval problem having the following space and time complexity on a unit cost RAM with free access to a fully random hash function* ($c > 0$ *is any constant*): (a) *For any fixed* $\gamma > 0$*, for any sufficiently large* n*, and for every* r *with* $1 \le r \le c \log n$*: space* $(1+\gamma)nr$ *bits, constant query time* $O(1+\log(\frac{1}{\gamma}))$*, and expected construction time* $O(n)$; (b) *For any sufficiently large* n *and every* r *with* $1 \le r \le c \log n$*: space* $nr + O(\log \log n)$ *bits whp.*[1]*, query time* $O(\log n)$*, and expected construction time* $O(n^3)$.

The basic data structure and query evaluation algorithm is the same as in [8]. The new contribution is to analyze a different construction algorithm (suggested in [23]) that is able to achieve a space usage arbitrarily close to the optimum. Our analysis needs tools and theorems from linear algebra, while that of [8] was based on elementary combinatorics ([23] provided only experimental results). To get a data structure that allows expected linear construction time we devise a new variant of the data structure and query evaluation algorithm, retaining simplicity and non-adaptivity. (We note that the data structure of [22] has an adaptive evaluation procedure, using many auxiliary tables.)

[1] "whp." means with probability $1 - O(\frac{1}{\mathrm{poly}(n)})$.

The papers on Bloom filters, and the work of Chazelle *et al.* [8] all make the assumption of access to fully random hash functions. We state that our data structures can be realized on a RAM, with a small additional cost in space (proof in [12]).

Theorem 2. *In the setting of Theorem 1, for some $\varepsilon > 0$, we can avoid the assumption of fully random hash functions and get data structures with the following space and time complexities*:
(a) *Space $(1+\gamma)nr$ bits, query time $O(1+\log(\frac{1}{\gamma}))$, expected construction time $O(n)$, for any constant $\gamma > 0$;*
(b) *Space $nr + O(n^{1-\varepsilon})$ bits, query time $O(\log n)$, expected construction time $O(n^{1+\delta})$, for an arbitrary constant $\delta > 0$.*

1.4 Overview of Paper

Section 2 describes our basic retrieval data structure and its analysis. This leads to part (a) of Theorem 1, except that the construction time is $O(n^3)$. For lack of space, the approach to the proof of part (b) is only sketched briefly. Section 3 completes the proof of part (a) of Theorem 1 by showing how the construction algorithm can be made to run in linear time. Section 4 describes a close relationship between the space requirements for dictionary implementations based on the multiple-choice paradigm (like k-ary cuckoo hashing [15]) and the space requirements for retrieval structures.

2 Retrieval in Constant Time and Almost Optimal Space

In this section, we give the basic construction of a data structure for retrieval with constant time lookup operation and $(1+\delta)nr$ space. As a technical basis, we first describe results by Calkin [6].

2.1 Calkin's Results

All calculations are over the field $\mathrm{GF}(2) = \mathbb{Z}_2$ with 2 elements. We consider binary matrices $M = (p_{ij})_{1\le i\le n, 0\le j<m}$ with n rows and m columns. If M is such a matrix, then row vector $(p_{i0}, \ldots, p_{i,m-1})$ is called p_i, for $1 \le i \le n$.

Theorem 3 (Calkin [6, Theorem 1.2]). *For every $k > 2$ there is a constant $\beta_k < 1$ such that the following holds*: *Assume the n rows $p_1, \ldots, p_n$ of a matrix M are chosen at random from the set of binary vectors of length m and weight (number of 1s) exactly k. Then we have the following*:
(a) *If $n/m \le \beta < \beta_k$, then $\Pr(M$ has full row rank$) \to 1$ (as $n \to \infty$).*
(b) *If $n/m \ge \beta > \beta_k$, then $\Pr(M$ has full row rank$) \to 0$ (as $n \to \infty$).*
Furthermore, $\beta_k - (1 - (e^{-k}/(\ln 2)) \to 0$ for $k \to \infty$ (exponentially fast in k).

Remark 1. (a) The case $k = 2$ is omitted in this discussion. The threshold value for this case is $\beta_2 = 2$, as is well known from the theory of random graphs. In

Table 1. Approximate threshold values from Theorem 3, using (1) and (2)

k	3	4	5	6
β_k	0.88949	0.96714	0.98916	0.99622
β_k^{appr}	0.9091	0.9690	0.9893	0.99624
β_k^{-1}	1.1243	1.034	1.011	1.0038

[17] and [3] this fact is used for constructing perfect hash functions, in a way that implicitly includes the construction of retrieval structures.
(b) A closer look into the proof of Theorem 1.2 in [6] reveals that for each k there is some $\varepsilon = \varepsilon_k > 0$ such that in the situation of Theorem 3(a) we have $\Pr(M \text{ has full row rank}) = 1 - O(n^{-\varepsilon})$. The following values are suitable: $\varepsilon_3 = \frac{2}{7}$, $\varepsilon_4 = \frac{5}{7}$, $\varepsilon_k = 1$ for $k \geq 5$.
(c) According to [6], the threshold value β_k is characterized as follows: Define

$$f(\alpha, \beta) = -\ln 2 - \alpha \ln \alpha - (1-\alpha)\ln(1-\alpha) + \beta \ln(1 + (1-2\alpha)^k), \qquad (1)$$

for $0 < \alpha < 1$. Let β_k be the minimal β so that $f(\alpha, \beta)$ attains the value 0 for some $\alpha \in (0, \frac{1}{2})$. Using a computer algebra system, it is easy to find approximate values for β_k and β_k^{-1} for small k, see Table 1. Calkin further proves that

$$\beta_k = 1 - \frac{e^{-k}}{\ln 2} - \frac{1}{2\ln 2}\left(k^2 - 2k + \frac{2k}{\ln 2} - 1\right) \cdot e^{-2k} \pm O(k^4) \cdot e^{-3k}, \qquad (2)$$

as $k \to \infty$. It seems that the approximation obtained by omitting the last term in (2) is quite good already for small values of k. (See the row for β_k^{appr} in Table 1.)

2.2 The Basic Retrieval Data Structure

Now we are ready to describe a retrieval data structure. Assume $f\colon S \to \{0,1\}^r$ is given, for a set $S = \{x_1, \ldots, x_n\}$. For a given (fixed) $k \geq 3$ let $1 + \delta > \beta_k^{-1}$ be arbitrary and let $m = (1+\delta)n$. We can arrange the lookup time to be $O(k)$ and the number of bits in the data structure to be $mr = (1+\delta)nr$ plus lower order terms.

We assume that we have access to a mapping $A\colon U \to \binom{[m]}{k}$, $x \mapsto A_x$, where $\binom{X}{k} = \{Y \subseteq X \mid |X| = k\}$, so that A is fully random on S. We write $A_x = \{h_1(x), \ldots, h_k(x)\}$ (the order is irrelevant). It must be possible to repeatedly choose a new function A if the need arises. We need to store an index to identify the function A that was actually used. It is not hard to see that using standard hash functions with ranges $[m], [m-1], \ldots, [m-k+1]$, such random sets with exactly k elements can be constructed in time $O(k)$. (For details see [12].)

The construction starts from $= \{x_1, \ldots, x_n\}$ and the bit strings $u_i = f(x_i) \in \{0,1\}^r$, $1 \leq i \leq n$. We consider the matrix

$$M = (p_{ij})_{1 \leq i \leq n, 0 \leq j < m}, \text{ with } p_{ij} = 1 \text{ if } j \in A_{x_i} \text{ and } p_{ij} = 0 \text{ otherwise.} \qquad (3)$$

Theorem 3(a) (with Remark 1(b)) says that M has full row rank with probability $1 - O(n^{-\varepsilon})$ for some $\varepsilon > 0$. Assume n is so large that this happens with probability at least $\frac{3}{4}$. If M does have full row rank, the column space of M is all of $\{0,1\}^n$, hence for all $u \in \{0,1\}^n$ there is some $a \in \{0,1\}^m$ with $M \cdot a = u$. More generally, we arrange the bit strings $u_1, \ldots, u_n \in \{0,1\}^r$ as a column vector $u = (u_1, \ldots, u_n)^\mathsf{T}$. We stretch notation a bit (but in a natural way) so that we can multiply binary matrices with vectors of r-bit strings: multiplication is just bit/vector multiplication and addition is bitwise XOR. It is then easy to see, working with the components of the u_i separately, that there is a (column) vector $a = (a_0, \ldots, a_{m-1})^\mathsf{T}$ with entries in $\{0,1\}^r$ such that $M \cdot a = u$.

We can rephrase this as follows (using $\oplus$ as notation for bitwise XOR): For $a \in (\{0,1\}^r)^m$ and $x \in U$ define

$$h_a(x) = \bigoplus_{j \in A_x} a_j. \tag{4}$$

Then for an arbitrary sequence $(u_1, \ldots, u_n)$ of prescribed values from $\{0,1\}^r$ there is some $a \in (\{0,1\}^r)^m$ with $h_a(x_i) = u_i$, for $1 \le i \le n$. Such a vector $a \in (\{0,1\}^r)^m$, together with an identifier for the function A used in the successful construction, is a data structure for retrieving the value $u_i = f(x_i)$, given x_i. There are k accesses to the data structure, plus the effort to calculate the set A_x from x.

Remark 2. A similar construction (over arbitrary fields GF(q)) was described by Seiden and Hirschberg [23]. However, those authors did not have Calkin's results, and so could not give theoretical bounds on the number m of columns needed. Also, our construction generalizes the approach of [8] and [17], where it was required that M could be transformed into echelon form by permuting rows and columns, which is sufficient, but not necessary, for M to have full row rank. Using these constructions it is not possible to work with $m \le 1.22n$ [17].

Some details of the construction are missing. We describe one of several possible ways to proceed. —From S, we first calculate the sets A_{x_i}, $1 \le i \le n$, in time $O(n)$. Using Gaussian elimination, we can check whether the induced matrix $M = (p_{ij})$ has full row rank. If this is not the case, we start all over with a new mapping $A\colon x \mapsto A_x$. This is repeated until a suitable matrix M is obtained. The expected number of repetitions is $1 + O(n^{-\varepsilon})$. For a matrix M with independent rows Gaussian elimination will also yield a "pseudoinverse" of M, that is, an invertible $n \times n$-matrix C (coding a sequence of elementary row transformations without row exchanges) with the property that in $C \cdot M$ the n unit vectors $e_i^\mathsf{T} = (0, \ldots, 0, 1, 0, \ldots, 0)^\mathsf{T}$ occur as columns:

$$\forall i,\ 1 \le i \le n,\ \exists\ b_i \in [m]\text{: column } b_i \text{ of } C \cdot M \text{ equals } e_i^\mathsf{T}. \tag{5}$$

Given $u = (u_1, \ldots, u_n) \in \{0,1\}^n$ we wish to find $a \in \{0,1\}^m$ such that

$$(C \cdot M) \cdot a = C \cdot u = u' = (u'_1, \ldots, u'_n)^\mathsf{T}. \tag{6}$$

Since $C \cdot M$ has the unit vectors in columns $b_1, \ldots, b_n$, we can easily read off a special a that solves (6): Let $a_j = 0$ for $j \notin \{b_1, \ldots, b_n\}$, and let $a_{b_i} = u'_i$ for $1 \le i \le n$. Exactly the same formula works if u, u', and a are vectors of r-bit strings. — We have established the following.

Theorem 4. *Assume that a mapping $A\colon U \to \binom{[m]}{k}$ is available that is fully random on S (with the option to choose such functions repeatedly and independently). Let $k > 2$ be fixed, let $1 + \delta > \beta_k^{-1}$, and assume n is sufficiently large. Then given $S = \{x_1, \ldots, x_n\}$ and a sequence $(u_1, \ldots, u_n)$ of prescribed elements in $\{0,1\}^r$, we can find a vector $a = (a_0, \ldots, a_{m-1})$ with elements in $\{0,1\}^r$ such that $h_a(x_i) = u_i$, for $1 \le i \le n$. The expected construction time is $O(n^3)$, and the scratch space needed is $O(n^2)$.*

Remark 3. At the first glance, the time complexity of the construction seems to be forbiddingly large. However, using a trick ("split-and-share" described in [11] and in [12]) makes it possible to obtain a data structure with the same functionality and space bounds (up to a $o(n)$ term) in time $O(n^{1+\delta})$ for any given $\delta > 0$. In Section 3 we show how to construct a retrieval structure with essentially the same space requirements in expected linear time.

We briefly give some ideas how Theorem 1(b) can be proved. The basic approach is similar to the above, but we use $k(x)$ hash functions for key x, where $k(x)$, $x \in S$, are independent random variables, each approximately binomially distributed with expectation $\Theta(\log n)$, and a range size $m = n$. Theorem 2(a) in [9] entails that the resulting square matrix will be regular with probability at least 0.28. It takes $O(n^3)$ time to test one matrix; trying $O(\log n)$ sets of hash functions will be sufficient whp. to find a set of hash functions that induces a matrix with full rank. Storing the index of this set of functions takes an extra $O(\log \log n)$ bits. The rest of the construction is similar to the above. A lookup then requires evaluating $O(\log n)$ hash functions. A splitting trick can be used to reduce the construction time to $O(n^{1+\delta})$, without changing the functionality, leading to Theorem 2(b). (Details in [12].)

3 Retrieval in Almost Optimal Space, with Linear Construction Time

In this section we show how, using a variant of the retrieval data structure described in Section 2.2, we can achieve linear expected construction time and still get arbitrarily close to optimal space. This will prove Theorem 1(a). The reader should be aware that the results in this section hold asymptotically, only for rather large n.

Using the notation of Sections 2.1 and 2.2, we fix some k and some $\delta > 0$ such that $(1 + \delta)\beta_k > 1$. Further, some constant $\varepsilon > 0$ is fixed. We assume that the required fully random hash functions and mappings from keys to sets are at our disposal, and in case the construction fails we can choose a new set of such functions, even repeatedly. (In [12] it is explained how this can be justified.)

Define $b = \frac{1}{2}\sqrt{\log n}$. We assume that ε and δ are so small that $(1+\varepsilon)^2(1+\delta) < 4$, and hence that $b \cdot 2^{(1+\varepsilon)^2(1+\delta)b^2} = o(n/(\log n)^3)$.

Assume $f\colon S \to \{0,1\}^r$ is given, the value $f(x)$ being denoted by u_x. The global setup is as follows: We use one fully random hash function φ to map S into the range $[m_0]$ with $m_0 = n/b$. In this way, m_0 blocks $B_i = \{x \in S \mid \varphi(x) = i\}$, $0 \le i < m_0$, are created, each with expected size b. The construction has two parts: a primary structure and a secondary structure for the "overflow keys" that cannot be accommodated in the primary structure. This is similar to the global structure of a number of well-known dictionary implementations. For the primary structure, we try to apply the construction from Section 2.2 to each of the blocks separately, but only once, with a fixed set of k hash functions. This construction may fail for one of two reasons: (i) the block may be too large — we do not allow more than $b' = (1+\varepsilon)b$ keys in a block if it is to be treated in the primary structure, or (ii) the construction from Section 2.2 fails because the row vectors in the matrix M_i induced by the sets A_x, $x \in B_i$, are not linearly independent.

For the primary structure, we set up a table T with $(1+\delta)(1+\varepsilon)n$ entries, partitioned into m_0 segments of size $(1+\delta)(1+\varepsilon)b = (1+\delta)b'$. Segment number i is associated with block B_i. If the construction from Section 2.2 fails, we set all the bits in segment number i to 0 and use the secondary structure to associate keys in B_i with the correct values. As secondary structure we choose a retrieval structure as in [8,17], built on the basis of a second set of three hash functions, which are used to associate sets $A'_x \subseteq [1.3n']$ with the keys $x \in S'$, and a table $T'[0..1.3n'-1]$. This uses space $1.3n'r$ bits, where n' is the size of the set S' of keys for which the construction failed (the "*overflow keys*"). Of course, the secondary structure associates a value $f'(x)$ with *any* key $x \in S$. Rather than storing information about which blocks succeed we compensate for the contribution from $f'(x)$ as follows: If the construction succeeds for B_i, we store $(1+\delta)b'$ vectors of length r in segment number i of table T so that $x \in B_i$ is associated with the value $f(x) \oplus f'(x)$. On a query for $x \in U$, calculate $i = \varphi(x)$, then the offset $d_i = (i-1)(1+\delta)b'$ of the segment for block B_i in T', and return

$$\bigoplus_{j \in A_x} T[j + d_i] \;\oplus\; \bigoplus_{j \in A'_x} T'[j].$$

It is clear that for $x \in S$ the result will be $f(x)$: For $x \in S'$ the two terms are $\mathbf{0}$ and $f(x)$, and for $x \notin S'$ the two terms are $f'(x)$ and $f(x) \oplus f'(x)$. Note that the accesses to the tables are nonadaptive: all $k+3$ lookups may be carried out in parallel. In fact, if T and T' are concatenated this can be seen as the same evaluation procedure as in our basic algorithm (4), the difference being that the hash functions were chosen in a different way (e.g., do not all have the same range).

Lemma 1. *The expected number of overflow keys is $o(n)$.*

The proof is a standard application of Chernoff bounds—we refer to [12] for details. The overall space is $(1+\delta)(1+\varepsilon)n(r+1/b) + c|S'|r$ bits (apart from

lower order terms). If $\gamma > 0$ is given, we may choose ε and δ (and k) so that this bound is smaller than $(1+\gamma)nr$ for n large enough.

Lemma 2. *The primary structure can be constructed in time $O(n)$.*

Proof: It is clear that linear time is sufficient to find the blocks B_i and identify the blocks that are too large. Now consider a fixed block B_i of size at most $(1+\varepsilon)b$. We must evaluate $|B_i| \cdot k$ hash functions to find the sets A_x, $x \in B_i$, and can piece together the matrix M_i that is induced by these sets in time $O(b)$ (assuming one can establish a word of $O(b)$ 0s in constant time and set a bit in such a word given by its position in constant time). The whole matrix has fewer than $\log n$ bits and fits into a single word. This makes it possible to use precomputed tables to speed up the computations we need. (The number of relevant matrixes is $o(\frac{n}{(\log n)^3})$ so that it is possible to calculate and store pseudoinverses and some matrix-vector products that we need in time and space $o(n)$. The details can be found in the full paper [12].)

Now assume a bit vector $u = (u_1, \ldots, u_{|B_i|})^{\mathsf{T}} \in \{0,1\}^{|B_i|}$, is given. Using C_i and a lookup table we can find $C_i \cdot u$ in constant time. A bit vector $a = (a_j)_{1 \le j \le (1+\delta)b'}$ that solves $M_i \cdot a = u$ can then be found in time $O(b)$. This leads to an overall construction time of $O(n)$ for the whole primary structure.

If the values in the range are bit vectors $f(x) = u_x \in \{0,1\}^r$, $x \in B_i$, a construction in time $O(nr)$ follows trivially. We may improve this time bound by arranging lookup tables that make it possible to multiply C_i even with vectors $U = (u_1, \ldots, u_{|B_i|})$ of bit vectors of length up to $O(\log n)$ in constant time. □

Note that the lookup tables are needed only by the construction algorithm, and not as part of the resulting data structure.

4 Retrieval and Dictionaries by Balanced Allocation

In several recent papers, the following scenario for (statically) storing a set $S \subseteq U$ of keys was studied. A set $S = \{x_1, \ldots, x_n\} \subseteq U$ is to be stored in a table `T[0..m-1]` of size $m = (1+\delta)n$ as follows: To each key x we associate a set $A_x \subseteq [m]$ of k possible table positions. Assume there is a mapping $\sigma\colon \{1, \ldots, n\} \to [m]$ that is one-to-one and satisfies $\sigma(i) \in A_{x_i}$, for $1 \le i \le n$. (In this case we say $(A_x, x \in S)$ is *suitable* for S.) Choose one such mapping and store x_i in `T[`$\sigma(i)$`]`. Examples of constructions that follow this scheme are cuckoo hashing [20], k-ary cuckoo hashing [15], blocked cuckoo hashing [13,21], and perfectly balanced allocation [10]. In [5,14] threshold densities for blocked cuckoo hashing were determined exactly. These schemes are the most space-efficient dictionary structures known, among schemes that store the keys explicitly in a hash table. For example, k-ary cuckoo hashing [15] works in space $m = (1+\varepsilon_k)n$ with $\varepsilon_k = e^{-\Theta(k)}$. Perfectly balanced allocation [10] works in optimal space $m = n$ with A_x consisting of 2 contiguous segments of $[n]$ of length $O(\log n)$ each.

Here, we point out a close relationship between dictionary structures of this kind and retrieval structures for functions $f\colon S \to R$, whenever the range R is

not too small. We will assume that $R = \mathbb{F}$ for a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq n$. (Using a simple splitting trick, this condition can be attenuated to $|\mathbb{F}| \geq n^\delta$.) From Section 2.2 we recall equation (3) where the matrix $M = (p_{ij})_{1 \leq i \leq n,\, 0 \leq j < m}$ was defined from the sets A_x, $x \in S$.

Observation 2. (*For arbitrary fields* $\mathbb{F}$.) *If the 1s in* M *can be replaced by elements of* $\mathbb{F}$ *in such a way that the resulting matrix* $M' = (p'_{ij})$ *has full row rank over* $\mathbb{F}$, *then* $(A_x, x \in S)$ *is suitable for* S.

Proof: If M' has full row rank, it has an $n \times n$ submatrix N with nonzero determinant. By the definition of the determinant there must be a mapping $\sigma\colon \{1, \ldots, n\} \to [m]$ with $\prod p'_{i\sigma(i)} \neq 0$, hence $p_{i\sigma(i)} = 1$ for $1 \leq i \leq n$. □

Observation 2 implies that Calkin's bounds give upper space bounds for dictionary constructions like k-ary cuckoo hashing, which match values observed in experiments in [15]. Surprisingly, for fields that are not too small, the observation also works the other way around: existence of a dictionary implies existence of a retrieval structure.

Theorem 5. *Assume a mapping* $x \mapsto A_x$ *is given that is suitable for* S. *Then the following holds*: *If* $g_1, \ldots, g_k\colon S \to \mathbb{F}$ *are random, then with probability* $\geq 1 - \frac{n}{|\mathbb{F}|}$ *the matrix* $M' = (p'_{ij})_{1 \leq i \leq n 0 \leq j < m}$, *where* $p'_{ij} = g_\ell(x_i)$ *if* $j = h_\ell(x_i)$ *and* $p'_{ij} = 0$ *otherwise, has full row rank over* $\mathbb{F}$.

The proof uses the Schwartz-Zippel Theorem; it is given in the full paper [12].

The theorem implies the following: If the mapping $x \mapsto A_x$ is suitable for S, if $|\mathbb{F}| \geq 2n$, and if we have hash functions $g_1, \ldots, g_k\colon U \to \mathbb{F}$ that are random on S, then with probability at least $\frac{1}{2}$ we can build a retrieval structure for a function $f\colon S \to \mathbb{F}$ consisting of a table $T[0..m-1]$ with entries from $\mathbb{F}$ with $f(x) = \sum_{1 \leq \ell \leq k} g_\ell(x) \cdot T[h_\ell(x)]$. If we can switch to new functions $g_1, \ldots, g_k$ if necessary, this construction succeeds in $O(1)$ iterations in the expected case and in $O(\log n)$ iterations whp.

For example, from the dictionary constructions in [13] or [10], resp., we obtain retrieval structures with a table of size $\leq (1 + e^{-k})nr$ and lookup time $O(k)$, or optimal size n and lookup time $O(\log n)$, resp. In both cases for one retrieval operation we need to access only two contiguous segments of the table T, which makes these implementations very cache-friendly.

Acknowledgement. The authors thank Philipp Woelfel for several motivating discussions on the subject.

References

1. Alstrup, S., Brodal, G.S., Rauhe, T.: Optimal static range reporting in one dimension. In: Proc. 33rd ACM STOC, pp. 476–482 (2001)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)

3. Botelho, F.C., Pagh, R., Ziviani, N.: Simple and space-efficient minimal perfect hash functions. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 139–150. Springer, Heidelberg (2007)
4. Broder, A.Z., Mitzenmacher, M.: Network applications of Bloom filters: A survey. In: Proc. 40th Annual Allerton Conference on Communication, Control, and Computing, pp. 636–646. ACM Press, New York (2002)
5. Cain, J.A., Sanders, P., Wormald, N.C.: The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In: Proc. 18th ACM-SIAM SODA, pp. 469–476 (2007)
6. Calkin, N.J.: Dependent sets of constant weight binary vectors. Combinatorics, Probability and Computing 6(3), 263–271 (1997)
7. Carter, L., Floyd, R.W., Gill, J., Markowsky, G., Wegman, M.N.: Exact and approximate membership testers. In: Proc. 10th ACM STOC, pp. 59–65 (1978)
8. Chazelle, B., Kilian, J., Rubinfeld, R., Tal, A.: The Bloomier filter: an efficient data structure for static support lookup tables. In: Proc. 15th ACM-SIAM SODA, pp. 30–39 (2004)
9. Cooper, C.: On the rank of random matrices. Random Struct. Algorithms 16(2), 209–232 (2001)
10. Czumaj, A., Riley, C., Scheideler, C.: Perfectly Balanced Allocation. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 240–251. Springer, Heidelberg (2003)
11. Dietzfelbinger, M.: Design strategies for minimal perfect hash functions. In: Proc. 4th Int. Symp. on Stochastic Algorithms: Foundations and Applications (SAGA). LNCS, vol. 4665, pp. 2–17. Springer, Heidelberg (2007)
12. Dietzfelbinger, M., Pagh, R.: Succinct data structures for retrieval and approximate membership, Technical Report, arXiv:0803.3693v1 [cs.DS] (March 26, 2008)
13. Dietzfelbinger, M., Weidling, C.: Balanced allocation and dictionaries with tightly packed constant size bins. Theoret. Comput. Sci. 380(1–2), 47–68 (2007)
14. Fernholz, D., Ramachandran, V.: The k-orientability thresholds for $G_{n,p}$. In: Proc. 18th ACM-SIAM SODA, pp. 459–468 (2007)
15. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. Theory Comput. Syst. 38(2), 229–248 (2005)
16. Hagerup, T., Tholey, T.: Efficient minimal perfect hashing in nearly minimal space. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 317–326. Springer, Heidelberg (2001)
17. Majewski, B.S., Wormald, N.C., Havas, G., Czech, Z.J.: A family of perfect hashing methods. Computer J. 39(6), 547–554 (1996)
18. Mitzenmacher, M.: Compressed Bloom filters. IEEE/ACM Transactions on Networking 10(5), 604–612 (2002)
19. Mortensen, C.W., Pagh, R., Pătraşcu, M.: On dynamic range reporting in one dimension. In: Proc. 37th ACM STOC, pp. 104–111 (2005)
20. Pagh, R., Rodler, F.F.: Cuckoo Hashing. J. Algorithms 51, 122–144 (2004)
21. Panigrahy, R.: Efficient hashing with lookups in two memory accesses. In: Proc. 16th ACM-SIAM SODA, pp. 830–839 (2005)
22. Porat, E.: An optimal Bloom filter replacement based on matrix solving, Technical Report, arXiv:0804.1845v1 [cs.DS] (April 11, 2008)
23. Seiden, S.S., Hirschberg, D.S.: Finding succinct ordered minimal perfect hash functions. Inf. Process. Lett. 51(6), 283–288 (1994)
24. Zukowski, M., Heman, S., Boncz, P.A.: Architecture-conscious hashing. In: Proc. Int. Workshop on Data Management on New Hardware (DaMoN), Chicago, 8 pages, Article No. 6 (2006)

Competitive Weighted Matching in Transversal Matroids

Nedialko B. Dimitrov* and C. Greg Plaxton**

University of Texas at Austin
1 University Station C0500
Austin, Texas 78712–0233
{ned,plaxton}@cs.utexas.edu

Abstract. Consider a bipartite graph with a set of left-vertices and a set of right-vertices. All the edges adjacent to the same left-vertex have the same weight. We present an algorithm that, given the set of right-vertices and the number of left-vertices, processes a uniformly random permutation of the left-vertices, one left-vertex at a time. In processing a particular left-vertex, the algorithm either permanently matches the left-vertex to a thus-far unmatched right-vertex, or decides never to match the left-vertex. The weight of the matching returned by our algorithm is within a constant factor of that of a maximum weight matching.

1 Introduction

Motivated by applications related to auctions, mechanism design, and revenue management, Babaioff et al. recently introduced a generalization of the secretary problem called the online matroid problem [1]. In the online matroid problem, the goal is to build a maximum weight independent set, but we are constrained from knowing the full input to the problem. Instead, a uniformly random permutation of the matroid elements is revealed, one element at a time, and we must immediately decide whether to include the revealed element in the independent set. In such a setting, an online algorithm is said to be *c-competitive* if it is able to produce an independent set with weight within a factor of c of the weight of a maximum weight independent [2]. We say that an online algorithm is *competitive* if it is c-competitive for some constant c.

Babaioff et al. present competitive algorithms for the online matroid problem on bounded left-degree transversal matroids and graphic matroids. They also present a reduction showing that if we have a competitive algorithm for a matroid M, then we can construct a competitive algorithm for a truncated version of M. Babaioff et al. leave open the general online matroid problem and the central case of transversal matroids. As discussed later in this section, the case of transversal matroids unifies the existing results on the online matroid problem. In this paper we present a competitive online algorithm for weighted matching in transversal

* Supported by an MCD Fellowship from the University of Texas at Austin.
** Supported by NSF Grants CCF–0635203 and ANI–0326001.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 397–408, 2008.

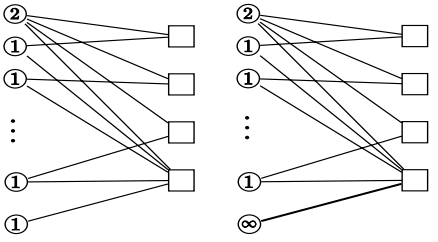
matroids, generalizing the results of Babaioff et al. Along with the reduction in Babaioff et al., our results also lead to competitive algorithms on truncated transversal matroids.

Informally, the online weighted transversal matroid matching problem can be described as follows. Consider a bipartite graph, with a set of left-vertices and a set of right-vertices. All edges adjacent to the same left-vertex have the same weight – we associate this weight with the left-vertex. The weighted transversal matroid matching problem (WTMM) asks us to find a maximum weight matching in this bipartite graph, and is solvable with the standard matroid greedy algorithm. In the *online* weighted transversal matroid matching problem (OWTMM), we are initially given only the total number of left-vertices, and then a uniformly random permutation of the left-vertices is revealed, one left-vertex at a time. When a vertex is revealed, we learn of both its weight and its incident edges. Upon seeing a particular left-vertex, without knowing the details of the remaining unrevealed left-vertices, we must immediately decide which right-vertex to match it to, if any. An open problem left by Babaioff et al. is to find an algorithm for OWTMM returning a matching with expected weight within a constant of the optimal matching in the corresponding WTMM problem. Theorem 1 presents such an algorithm.

In the literature, a transversal matroid is often specified by a set of elements E, and a set of subsets $A_1, \ldots, A_n$ of E [8]. A subset $I = \{a_1, \ldots, a_k\}$ of E is considered independent if there is an injective function f mapping I to $\{A_1, \ldots, A_n\}$ such that $x \in f(x)$ for all inputs x. In our presentation, the set of elements E corresponds to the left-vertices, the sets $A_1, \ldots, A_n$ correspond to the right-vertices, and there is an edge between an element of E and a set A_j if the element belongs to the set. An independent set then corresponds to a set of left-vertices for which there exists a matching to the right-vertices.

Perhaps the most well studied online matroid problem is the secretary problem, which first appeared as a folklore problem in the 1950's and has a long history [4,5]. The problem was first solved by Lindley, who also presents a competitive algorithm for the secretary problem [9]. Competitive algorithms also exist for uniform matroids [7], bounded left-degree transversal matroids, graphic matroids, and truncated matroids [1]. For general matroids, the best known competitive ratio is $O(\log r)$ where r is the rank of the matroid [1].

With the exception of truncated matroids, where the result depends on Karger's matroid sampling theorem [6], all of the matroids for which a competitive algorithm is known are a special case of the transversal matroid. For example, the secretary problem is a transversal matroid with a single right-vertex. The uniform matroid of rank r is a transversal matroid on a complete bipartite graph with r right-vertices. Of course, bounded left-degree transversal matroids are a special case of the transversal matroids. And, finally, the competitive results for graphic matroids follow from a reduction to bounded-left degree transversal matroids. Thus, indeed, transversal matroids play a central role to the theory. For some remarks on the strong connection between general matroids and transversal matroids, see Section 8.



(a) Example 1 (b) Example 2

Fig. 1. Two example transversal matroids exhibiting the tension between using sampled heavy left-vertices for pricing and over-pricing the right-vertices. The figures are meant only to be illustrative, but can be extended to become counter-examples for certain pricing strategies. In 1(a), we do not want to price all the right-vertices at 2, since we would miss many left-vertices of weight 1. In 1(b), we want to price the bottom right-vertex at 2, since otherwise we would miss the infinite weight left-vertex.

1.1 Algorithm Motivations

Recall that the secretary problem is OWTMM with a single right-vertex and consider the following classic algorithm for the secretary problem. We sample the first $m/2$ left-vertices we see, rejecting all of them, but recording their edge weights. We set a price for the right-vertex equal to the maximum weight edge we see in the sample. We then match the right-vertex with the first non-sampled left-vertex whose edge weight exceeds the price, if we see such a left-vertex. The algorithm is competitive since with probability at least $1/4$, the second heaviest edge is sampled and the heaviest edge is not sampled.

This simple sample-and-price algorithm is the motivation for most of the competitive algorithms known for online matroid problems, again with the exception of truncated matroids. However, extending this algorithm to work for all, general transversal matroids is not straightforward. For example, Babaioff et al. show that a sample-and-price algorithm with an adaptive sampling time which sets the same price for all the right-vertices does not work. Babaioff et al. also show that a more complicated scheme, where the price required of a non-sampled left-vertex is determined by a circuit of sampled left-vertices also does not work.

One of the main issues that arises in trying to generalize the sample-and-price algorithm is a tension between the need to use sampled heavy left-vertices to price the right-vertices and the requirement that we not over-price too many right-vertices. Consider the example in Figure 1(a). If in the sample we see the left-vertex of weight 2, we should not over-price all the right-vertices at 2, since that prevents us from matching a large number of vertices of weight 1. The figure is only meant as an illustration, but can be extended to a counter-example by adding $\log m$ clones of the left-vertex of weight 2. On the other hand, consider the example in Figure 1(b). If we do not set a price of 2 for the bottom-most right-vertex, we would prematurely match that right-vertex to a left-vertex of weight 1 instead of the infinite weight left-vertex. It is natural to consider more complex pricing schemes, such as dynamic prices that change throughout processing, or

picking a random subset of the neighbors of a heavy left-vertex and pricing only those neighbors. However, it is both unclear if such schemes are effective and it is difficult to analyze them as they often introduce complicated probabilistic dependencies. It is this tension that leads Babaioff et al. to consider bounded left-degree transversal matroids.

For our results, we avoid the difficulties arising from more complex schemes with the concept of "candidate edges." The candidate edges we introduce have the following important properties. First, each left-vertex i has exactly zero or one candidate edges, uniquely determined by the sampled left-vertices heavier than i. In other words, given the sampled left-vertices heavier than i, the candidate edge is the same regardless of whether i is sampled, or where in the random order of non-sampled vertices it appears. Second, the candidate edges of the sampled left-vertices constitute a matching that is within a constant-factor of the max-weight matching on the sampled subgraph.

The analysis following from our definition of candidate edges is essentially the original sample-and-price analysis from the secretary problem, but applied to each right-vertex separately. The algorithm prices right-vertices using only the candidate edges. Furthermore, a non-sampled left-vertex can only be matched using its candidate edge. For a particular right-vertex, as in the secretary problem, we hope that the second-heaviest left-vertex with a candidate edge to the right-vertex is sampled, but the heaviest left-vertex with a candidate edge to the right vertex not sampled. Similarly to the secretary problem, this happens with at least $1/4$ probability.

The overall argument structure is as follows. In Section 2, we define some useful notation. In Section 3, we define candidate edges and show that they constitute a matching with weight within a constant factor of optimal on the sampled subgraph. In Section 4, to avoid any confusion from probabilistic dependencies, we analyze sampled and non-sampled matchings through counting arguments. Our counting argument immediately imply that a matching resulting from candidate edges of non-sampled left-vertices has expected weight within a constant factor of the expected weight of the matching of candidate edges of sampled left-vertices. In Section 5, we show that the expected weight of the sampled candidate edge matching is within a constant factor of the max-weight matching on the entire graph. This completes the main technical arguments, since the non-sampled matching is within a constant factor of the sampled matching, which is within a constant of the optimal matching on the whole graph. In Section 6, we present a small but clarifying intermediate algorithm between the final online algorithm and the counting arguments presented earlier. Finally, in Section 7, we present the online algorithm and conclude the analysis.

2 Definitions

In this section, we formally define some quantities and notation we will use throughout the paper.

Fix a set of n right-vertices, numbered 0 to $n-1$.

Fix a set of m left-vertices, where each left-vertex i is described by a triple of 1) a real number weight, $w(i)$ 2) a unique integer ID and 3) a subset of the right vertices, Right(i). We define a total order on the left-vertices: we say a left-vertex i is less than a left-vertex i' if $w(i) > w(i')$ or $w(i) = w(i')$ and i has a smaller unique integer ID. We draw the reader's attention to the fact that smaller left-vertices have greater weight. From here on, we use the integers to denote the left-vertices, with 0 denoting the minimum left-vertex, 1 denoting the second minimum left-vertex and so forth. We draw the reader's attention to the fact that the ordering on the left-vertices is the same as the ordering on the corresponding integers.

For a nonempty subset A of left-vertices or right-vertices let Min(A) return the minimum vertex, as defined by the corresponding total order.

An edge is a pair (i, j), where i is a left-vertex and j belongs to Right(i). A matching is a set of edges M such that each vertex appears in at most one edge. For a matching M, let Left(M), Right(M), denote the left-vertices, right-vertices, in the matching, respectively.

For a set of left-vertices, A, we say $w(A) = \sum_{i \in A} w(i)$. For a matching M, we say $w(M) = w(\text{Left}(M))$.

To facilitate our proofs, we define the following notation. For a subset of left-vertices L, let Prefix$(L, i) = \{i' \in L \mid i' < i\}$. Similarly, for a matching M, let Prefix$(M, i) = \{(i', j) \in M \mid i' < i\}$.

3 Algorithm A

In this section we define candidate edges and show the two main properties discussed in Section 1.1. The first property, "each left-vertex i has exactly zero or one candidate edges, uniquely determined by the sampled left-vertices heavier than i" corresponds to Lemmas 1 and 2. The second property, "the candidate edges of the sampled left-vertices constitute a matching that is within a constant-factor of the max-weight matching on the sampled subgraph" corresponds to Lemma 5.

First, we define a function Cands(i, M) that receives a left-vertex i and a matching M, and returns an edge set. The Cands(i, M) function is as follows:

$M' :=$ Prefix(M, i)
$A :=$ Right(i) − Right(M')
if $A = \emptyset$
    return $\emptyset$
else
    return $\{(i, \text{Min}(A))\}$

Lemma 1. *For any left-vertex i and matching M,* Cands(i, M) *either returns the empty set, or* $\{(i, j)\}$*, where j is a right-vertex unmatched in* Prefix(M, i)*.*

Proof. Follows from the definition of Cands. □

Lemma 2. *For any left-vertex i and matchings M and M' with* $\text{Prefix}(M, i) = \text{Prefix}(M', i)$*, we have* $\text{Cands}(i, M) = \text{Cands}(i, M')$.

Proof. Follows from the definition of Cands. □

We now define an algorithm for WTMM. Algorithm $\text{AlgA}(L)$ takes a subset of left-vertices L and returns a matching and the algorithm is performed as follows:

$M := \emptyset$
for i in increasing order in L
    $M := M \cup \text{Cands}(i, M)$
return M

Recall that the total order on left-vertices is defined such that i is less than i' if $w(i) > w(i')$ or $w(i) = w(i')$ and i has a smaller unique integer ID.

Lemma 3. *For a subset of left-vertices L, let $M = \text{AlgA}(L)$, then M is a matching on L and $M = \cup_{k \in L} \text{Cands}(k, M)$.*

Proof. We prove the lemma by first proving the following loop invariant in $\text{AlgA}(L)$: M is a matching on $\text{Prefix}(L, i)$ and $M = \cup_{k \in \text{Prefix}(L,i)} \text{Cands}(k, M)$.

The claimed invariant hold initially since $M := \emptyset$ and $i = \text{Min}(L)$. Suppose the claim is true for M and i on entering the loop on which we process i. Let $M' = M \cup \text{Cands}(i, M)$ and i' be the next left-vertex in order from L.

Let $A = \text{Cands}(i, M)$. We split the analysis in two cases. First, suppose $A = \emptyset$. Then, $M' = M$ and the claim holds for M' and i' simply because it holds for M and i. Second, suppose $A = \{(i, j)\}$, for a right-vertex j unmatched in $\text{Prefix}(M, i)$ (Lemma 1). Since $\text{Prefix}(L, i') = \text{Prefix}(L, i) \cup \{i\}$, the first part of the invariant holds.

For the second part of the invariant, we have $M' = M \cup \text{Cands}(i, M) = \bigcup_{k \in \text{Prefix}(L,i)} \text{Cands}(k, M) \cup \text{Cands}(i, M) = \bigcup_{k \in \text{Prefix}(L,i')} \text{Cands}(k, M)$, which equals $\bigcup_{k \in \text{Prefix}(L,i')} \text{Cands}(k, M')$. The second equality holds because the loop invariant holds for M and i, the third equality holds by the definition of i'; and the final equality holds by Lemma 2. This proves the invariant.

The lemma statement follows from following the same reasoning as in the inductive step above, but taken for the final iteration of the loop. □

Lemma 4. *Let M^* be a max-weight matching on L and M be the matching returned by* $\text{AlgA}(L)$*. If $(i, j) \in M^*$ and i is unmatched in M, then there is a i' such that $(i', j) \in M$ and $w(i') \geq w(i)$.*

Proof. By Lemma 3, $M = \cup_{k \in L} \text{Cands}(k, M)$. Since i is not matched in M and by Lemma 1, we have $\emptyset = \text{Cands}(i, M)$. By the definition of Cands, the empty set can only be returned if $\text{Right}(i) \subseteq \text{Right}(\text{Prefix}(M, i))$. In other words, every right-vertex in $\text{Right}(i)$ is matched to a left-vertex less than i in M, completing the proof. □

Lemma 5. *Let M^* be a max-weight matching on L, and M be the matching returned by* $\text{AlgA}(L)$*. Then $w(M) \geq \frac{1}{2} w(M^*)$.*

Proof. By summing the inequality in Lemma 4 over left-vertices matched in $M^* - M$, we have $w(M) \geq w(M^* - M)$. By definition of intersection, we have $w(M) \geq w(M^* \cap M)$. Combining the two inequalities, we have $2w(M) \geq w(M^*)$. □

4 Counting Arguments

In this section, to avoid confusion with probabilistic dependencies, we analyze sampled and non-sampled matchings through counting arguments. As stated in Section 1.1, our counting arguments, in specific Lemma 13, immediately imply that a matching resulting from candidate edges of non-sampled left-vertices has expected weight at least 1/4 of the expected weight of the matching of candidate edges of sampled left-vertices. From this section we only export Lemma 6, which is used to connect the counting arguments with the final online algorithm, and Lemma 13.

Let α be a binary string and α_i be the ith character in the string. Intuitively, the reader should think of a 0 in the ith position of α as sampling the left-vertex i and of a 1 in the ith position as not sampling i. For two binary strings α and β, let $\alpha\beta$ denote concatenation. For a binary string α of length at most m, we define the sets of edges $M_0(\alpha), M_2(\alpha), E_0(\alpha)$ recursively as follows.

$$\begin{aligned} M_0(\epsilon) &= E_0(\epsilon) = \emptyset \\ M_2(\alpha) &= \text{Cands}(|\alpha|, M_0(\alpha)) \\ M_0(\alpha 0) &= M_0(\alpha) \cup M_2(\alpha) \\ M_0(\alpha 1) &= M_0(\alpha) \\ E_0(\alpha 0) &= E_0(\alpha) \\ E_0(\alpha 1) &= E_0(\alpha) \cup M_2(\alpha) \end{aligned}$$

Finally, we also define $E_1(\alpha) = M_0(\alpha) \cup E_0(\alpha)$ and $M_1(\alpha)$ to be $\{(i,j) \in E_0(\alpha) \mid j \text{ appears at most once in } E_0(\alpha)\}$. It is not difficult to show that $M_0(\alpha)$, $M_1(\alpha)$ and $M_2(\alpha)$ are matchings while $E_0(\alpha)$ and $E_1(\alpha)$ are sets of edges.

We give the reader a loose intuitive interpretation of these definitions. Intuitively, one can think of processing the left-vertices in order of increasing weight as we increase the length of α. Then, $M_2(\alpha)$ represents the $|\alpha|$th candidate edge; $M_0(\alpha)$ represents a matching created from the sampled left-vertices; $E_0(\alpha)$ represents a set of edges created from the non-sampled left-vertices such that each non-sampled left-vertex appears at most once; $E_1(\alpha)$ represents a set of all candidate edges, regardless of whether the corresponding left-vertex is sampled; and $M_1(\alpha)$ represents a matching created from the non-sampled left-vertices.

Lemma 6. *For a binary string α of length at most m, let $A = \{i \mid \alpha_i = 0\}$ and $B = \{i \mid \alpha_i = 1\}$. We have, $M_0(\alpha) = \bigcup_{i \in A} \text{Cands}(i, M_0(\alpha))$ and $E_0(\alpha) = \bigcup_{i \in B} \text{Cands}(i, M_0(\alpha))$.*

Proof. We prove the claim by induction on the length of α. For $\alpha = \epsilon$, the claim follows from the definition of $M_0(\epsilon)$ and $E_0(\epsilon)$. The inductive claim follows from Lemma 2 and the recursive definitions of $M_0(\alpha)$ and $E_0(\alpha)$. □

For a set of edges A, let $\deg(A, j)$ denote the degree of the right-vertex j in A. For a left-vertex i and a right-vertex j, we partition the set of binary strings to assist in our counting arguments as follows.

$\alpha \in S_0(i,j)$ if $|\alpha| < i, \deg(E_1(\alpha), j) = 0$
$\alpha \in S_1(i,j)$ if $|\alpha| = i, \deg(E_1(\alpha), j) = 0, M_2(\alpha) = \{(i,j)\}$
$\alpha \in S_2(i,j)$ if $\deg(E_0(\alpha), j) = \deg(E_1(\alpha), j) = 1, \alpha = \beta\gamma, \beta \in S_1(i,j)$
$\alpha \in S_3(i,j)$ if $\deg(E_0(\alpha), j) = 1, \deg(E_1(\alpha), j) > 1, \alpha = \beta\gamma, \beta \in S_2(i,j)$
$\alpha \in S_4(i,j)$ otherwise.

We give the reader some intuitive interpretation of the above sets. For a particular pair (i,j): $S_0(i,j)$ represents strings where j has never been returned by Cands and we have not yet reached i; $S_1(i,j)$ represents strings where Cands has never before returned j, we have just now reached i and Cands returns $\{(i,j)\}$; $S_2(i,j)$ represents strings where j has been returned exactly once by Cands, when j was returned by Cands it was along with i and i was non-sampled; $S_3(i,j)$ represents strings where the first time Cands returned j it was along with i and i was non-sampled, then j was returned again with some other, sampled vertex i'; Finally, $S_4(i,j)$ represents all other strings.

Due to space limitations, the proofs of Lemmas 7 and 11 bellow are omitted from this extended abstract. The reader is referred to our companion technical report for the proofs of these lemmas [3].

Lemma 7. *For any right-vertex j, left-vertex i and integer k such that $i < k \leq m$, we have $|S_2(i,j) \cap \{0,1\}^k| + 2|S_3(i,j) \cap \{0,1\}^k| = 2^{k-i-1}|S_1(i,j)|$.*

Lemma 8. *For any right-vertex j, left-vertex i and integer k such that $i < k \leq m$, we have $|(S_2(i,j) \cup S_3(i,j)) \cap \{0,1\}^k| \geq 2^{k-i-2}|S_1(i,j)|$.*

Proof. By Lemma 7, we have that $|S_2(i,j) \cap \{0,1\}^k| + 2|S_3(i,j) \cap \{0,1\}^k|$ is equal to $2^{k-i-1}|S_1(i,j)|$. We can increase the left-hand side to get $2|S_2(i,j) \cap \{0,1\}^k| + 2|S_3(i,j) \cap \{0,1\}^k| \geq 2^{k-i-1}|S_1(i,j)|$. Since $S_2(i,j)$ and $S_3(i,j)$ are disjoint, we have $|(S_2(i,j) \cup S_3(i,j)) \cap \{0,1\}^k| \geq 2^{k-i-2}|S_1(i,j)|$. □

Lemma 9. *For any right-vertex j and integer k such that $k \leq m$, we have $\sum_{\alpha \in \{0,1\}^k} w(M_1(\alpha), j) \geq \sum_{0 \leq i < k} w(i) 2^{k-i-2} |S_1(i,j)|$, where $w(M_1(\alpha), j)$ is denotes the weight of the left-vertex matched to j in $M_1(\alpha)$ or zero if j is unmatched.*

Proof. By the definitions of M_1,S_2 and S_3, the left-hand side of the claimed inequality is equal to $\sum_{0 \leq i < k} w(i) |(S_2(i,j) \cup S_3(i,j)) \cap \{0,1\}^k|$, since $(i,j) \in M_1(\alpha)$ implies $\alpha \in (S_2(i,j) \cup S_3(i,j))$. Applying Lemma 8 gives the desired result. □

Lemma 10. *For any integer k such that $k \leq m$, we have* $\sum_{\alpha\in\{0,1\}^k} w(M_1(\alpha)) \geq \sum_{0\leq i<k}\sum_{0\leq j<n} w(i)2^{k-i-2}|S_1(i,j)|$.

Proof. Follows from summing the result of Lemma 9 over all $0 \leq j < n$. □

Lemma 11. *For any right-vertex j and integer k such that $k \leq m$, we have* $\sum_{\alpha\in\{0,1\}^k} w(M_0(\alpha), j) \leq \sum_{0\leq i<k} w(i)2^{k-i}|S_1(i,j)|$, *where $w(M_0(\alpha), j)$ is equal to the weight of the left-vertex matched to j in $M_0(\alpha)$ or zero if j is unmatched.*

Lemma 12. *For any integer k such that $k \leq m$, we have* $\sum_{\alpha\in\{0,1\}^k} w(M_0(\alpha)) \leq \sum_{0\leq i<k}\sum_{0\leq j<n} w(i)2^{k-i}|S_1(i,j)|$.

Proof. Follows from summing the result of Lemma 11 over all $0 \leq j < n$. □

Lemma 13. *For any integer k such that $k \leq m$, we have* $\sum_{\alpha\in\{0,1\}^k} w(M_1(\alpha)) \geq \frac{1}{4}\sum_{\alpha\in\{0,1\}^k} w(M_0(\alpha))$.

Proof. Follows from Lemmas 12 and 10. □

5 Analysis under a Probability Distribution

In this section we begin working with probability distributions and show that the expected weight of the sampled candidate edge matching is within a constant factor of the max-weight matching on the entire graph (Lemma 15). We tie these results with Section 4, to show that the expected weight of the non-sampled matching is within a constant the weight of a max-weight matching on the entire graph (Lemma 17), completing the main technical portion of the argument. The only result exported from this section is Lemma 17.

Define a function Sample, which takes an m-bit binary string string α such that $\text{Sample}(\alpha) = \{i \mid \alpha_i = 0\}$. We introduce a probability distribution $\mathcal{P}$ on strings m-bit binary strings α. In $\mathcal{P}$ each α_i independently has an equal chance of $\alpha_i = 0$ and $\alpha_i = 1$.

Lemma 14. *Let M^* be a max-weight matching and M_α denote a max-weight matching on* $\text{Sample}(\alpha)$ *for a binary string α. Then,* $\text{Exp}[w(M_\alpha)] \geq \frac{1}{2}w(M^*)$.

Proof. We have $\text{Exp}[w(M_\alpha)] \geq \sum_{i\in\text{Left}(M^*)} \Pr[\alpha_i = 0]w(i) = \frac{1}{2}w(M^*)$, where the first step follows from the linearity of expectation and observing that the matching M_α as a weight at least as big as the weight of a matching $M'_\alpha = \{(i,j) \in M^* \mid \alpha_i = 0\}$. □

Lemma 15. *Let M^* be a max-weight matching. Then, the following inequality holds* $\text{Exp}[w(\text{AlgA}(\text{Sample}(\alpha)))] \geq \frac{1}{4}w(M^*)$.

Proof. Let M_α be a max-weight matching on $\text{Sample}(\alpha)$, for a string α. Then, we have $\text{Exp}[w(\text{AlgA}(\text{Sample}(\alpha)))] \geq \text{Exp}[\frac{1}{2}w(M_\alpha)] \geq \frac{1}{4}w(M^*)$, where the first step follows from Lemma 5 and the second step follows from Lemma 14 and the linearity of expectation. □

Lemma 16. *Let α be any m-bit binary string and $A = \text{Sample}(\alpha)$. We have $M_0(\alpha) = \text{AlgA}(A)$.*

Proof. Follows from the definition of AlgA and M_0. □

Lemma 17. *Let M^* be a max-weight matching. We have $\text{Exp}[w(M_1(\alpha))] \geq \frac{1}{16} w(M^*)$.*

Proof. Follows from Lemmas 13, 15, and 16. □

6 Intermediate Algorithm

In this section we analyze a useful intermediate algorithm between the counting arguments and the final online algorithm. In specific, in the counting argument, we process the non-sampled left-vertices in decreasing order of weight. In this section we use Lemma 6 to argue that we can process the non-sampled left-vertices in an arbitrary order. This is similar to what happens in the original sample-and-price algorithm in the secretary problem. In the secretary problem, we depend on the fraction of time when the second highest bidder is sampled and the highest bidder is not. When this happens, we can process the non-sampled bidders in an arbitrary order, since only one of them meets the required price.

We define an algorithm $\text{AlgB}(\alpha)$ that receives as input m-bit binary string α, and returns a matching. The $\text{AlgB}(\alpha)$ function is as follows:

$M := \text{AlgA}(\text{Sample}(\alpha))$
$A := \{0, \ldots, m\} - \text{Sample}(\alpha)$
$E := \emptyset$
for i in arbitrary order from A:
    $E := E \cup \text{Cands}(i, M)$
return the matching of pairs (i, j) in E where j appears at most once in E

Lemma 18. *For any m-bit binary string α, we have $\text{AlgB}(\alpha) = M_1(\alpha)$.*

Proof. Follows from Lemmas 16 and 6 and the definition of $M_1(\alpha)$. □

7 Online Algorithm

In this section, we define and analyze the final online algorithm, which is closely related to the algorithm in Section 6. The main difference between the two algorithms is that the online algorithm relies on the random permutation of left-vertices for sampling whereas our results discuss a sampling method where each element has an equal chance of being sampled or not. With Lemma 20 we show that the two sampling methods induce the same distribution. The main theorem follows.

Define the online algorithm as follows. Initially, we are given the set of right-vertices, and the total number of left-vertices we will see, m. The algorithm ONLINE proceeds in two phases.

First phase:
$k := \text{Bin}(m, \frac{1}{2})$, where Bin is the binomial distribution.
Reject the first k vertices, not matching them to anything.
Let B be the set of all the rejected vertices.
$M_0 := \text{AlgA}(B)$

Second phase:
We are given M_0 from the first phase.
We build a matching M_1, initialized to $\emptyset$.
On receiving a left-vertex i:
$A := \text{Cands}(i, M_0)$
if $A \neq \emptyset$ and the right-vertex in A is unmatched in M_1
$M_1 := M_1 \cup A$
return M_1

Lemma 19. *Let α be a m-bit binary string, $B = \text{Sample}(\alpha)$ and M_1^B be a matching returned by ONLINE when B is sampled in the first phase. Then, $w(M_1^B) \geq w(\text{AlgB}(\alpha))$.*

Proof. ONLINE and AlgB perform the same operations on the vertices that are not sampled, with the small optimization that ONLINE matches a right vertex j to the first left-vertex i such that $\{(i, j)\} = \text{Cands}(i, M_0)$, while AlgB does not match any right-vertex j that is returned twice by Cands. □

Lemma 20. *Consider a set A of m elements. Let $\mathcal{P}$ be the probability distribution where each $a \in A$ independently has an equal chance of being sampled or not sampled. Let $\mathcal{P}'$ be the probability distribution where we first pick a k from $\text{Bin}(m, \frac{1}{2})$, and then we sample the first k elements from a uniformly random permutation of A. Then the probability distributions $\mathcal{P}$ and $\mathcal{P}'$ are equal.*

Proof. Each subset of A is sampled with the same probability in both cases. □

Theorem 1. *Let M^* be a max-weight matching. Given a uniformly random permutation of the left-vertices, ONLINE returns a matching whose weight is least $\frac{1}{16}w(M^*)$ in expectation.*

Proof. Follows from Lemmas 17, 18, 19, and 20. □

8 Concluding Remarks

In this section we make some remarks on the strong connection between transversal matroids and general matroids. The connection comes from the following characterization of a basis: B is a basis of a matroid M iff B is a minimal set having non-empty intersection with every co-circuit of M [10]. With this characterization, one can think of a general matroid as a bipartite graph in the following way. Let the matroid elements be the left-vertices and co-circuits be the right-vertices. Let there be an edge between an element and a co-circuit if

the element belongs to the co-circuit. An independent set in the general matroid is then a combinatorial structure which is close to a matching, but not the same. Consider taking a particular element into an independent set we are constructing. On taking in the element, we cover all the co-circuits containing that element because they have non-empty intersection with the constructed independent set. After that, to increase the independent set, we can only take elements which cover some uncovered co-circuits. Perhaps it is possible to come up with a sample-and-price scheme for pricing co-circuits to extend the results of this paper to general matroids, solving the online matroid problem?

References

1. Babaioff, M., Immorlica, N., Kleinberg, R.: Matroids, secretary problems, and online mechanisms. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete algorithms, pp. 434–443 (January 2007)
2. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
3. Dimitrov, N.B., Plaxton, C.G.: Competitive weighted matching in transversal matroids. Technical Report TR–08–04, Department of Computer Science, University of Texas at Austin (January 2008)
4. Ferguson, T.: Who solved the secretary problem? Statistical Science 4, 282–289 (1989)
5. Freeman, P.R.: The secretary problem and its extensions: A review. International Statistical Review 51, 189–206 (1983)
6. Karger, D.: Random sampling and greedy sparsification for matroid optimizaiton problems. Mathematical Programming 82, 41–81 (1998)
7. Kleinberg, R.: A multiple-choice secretary algorithm with applications to online auctions. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete algorithms, pp. 630–631 (January 2005)
8. Lawler, E.: Combinatorial Optimization: Networks and Matroids. Dover Publications, Mineola (2001)
9. Lindley, D.V.: Dynamic programming and decision theory. Applied Statistics 10, 39–51 (1961)
10. Oxley, J.: What is a matroid? Cubo. 5, 179–218 (2003)

Scheduling for Speed Bounded Processors

Nikhil Bansal[1], Ho-Leung Chan[2], Tak-Wah Lam[3], and Lap-Kei Lee[3]

[1] IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY
nikhil@us.ibm.com
[2] Computer Science Department, University of Pittsburgh
hlchan@cs.pitt.edu
[3] Department of Computer Science, University of Hong Kong, Hong Kong
{twlam, lklee}@cs.hku.hk

Abstract. We consider online scheduling algorithms in the dynamic speed scaling model, where a processor can scale its speed between 0 and some maximum speed T. The processor uses energy at rate s^α when run at speed s, where $\alpha > 1$ is a constant. Most modern processors use dynamic speed scaling to manage their energy usage. This leads to the problem of designing execution strategies that are both energy efficient, and yet have almost optimum performance.

We consider two problems in this model and give essentially optimum possible algorithms for them. In the first problem, jobs with arbitrary sizes and deadlines arrive online and the goal is to maximize the throughput, i.e. the total size of jobs completed successfully. We give an algorithm that is 4-competitive for throughput and $O(1)$-competitive for the energy used. This improves upon the 14 throughput competitive algorithm of Chan et al. [10]. Our throughput guarantee is optimal as any online algorithm must be at least 4-competitive even if the energy concern is ignored [7]. In the second problem, we consider optimizing the trade-off between the total flow time incurred and the energy consumed by the jobs. We give a 4-competitive algorithm to minimize total flow time plus energy for unweighted unit size jobs, and a $(2 + o(1))\alpha/\ln\alpha$-competitive algorithm to minimize fractional weighted flow time plus energy. Prior to our work, these guarantees were known only when the processor speed was unbounded ($T = \infty$) [4].

1 Introduction

In the last few years, the increasing computing power of processors has caused dramatic increase in their energy consumption. This not only leads to high cooling costs but also to substantially reduced battery life in laptops and other mobile devices. Companies such as IBM, Intel and AMD have made power aware design a key priority and even scrapped the development of faster processors in favor of lower power ones. To be more energy efficient, many modern processors now adopt the technology of *dynamic speed (voltage) scaling* (see, e.g., [13,21,23]), where the processor can adjust its speed dynamically in some range without any overhead. For example, IBM's PowerPC 970FX [1] allows the operating system to dynamically vary the speed (with zero overhead) at various discrete points from 2.5GHz to 625MHz while the power consumption reduces from 100W to less than 10W. In general, the rate of energy usage varies approximately

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 409–420, 2008.

as s^α with speed s, where α is typically 2 or 3 [9,20][1]. Most research in dynamic voltage scaling has focused on how to exploit this capability to reduce energy consumption without an apparent reduction in the functionality offered by the system.

A theoretical investigation of dynamic speed scaling was initiated by the seminal paper of Yao, Demers and Shenker [24]. They considered a model where the processor can run at any speed between 0 and infinity, incurring an energy (cost) of s^α per time unit when run at speed s. We call this the *infinite* speed model. In this model, Yao et al. studied the problem where jobs with arbitrary sizes and deadlines are released over time in an online manner. The goal is to find a schedule that completes all jobs by their deadlines while minimizing the total energy used. They gave an algorithm that is $2^{\alpha-1}\alpha^\alpha$-competitive for energy, and proposed another algorithm OA. Bansal, Kimbrel and Pruhs [3] showed that OA is α^α-competitive and this ratio is tight (OA is discussed in further detail later). They gave another algorithm BKP which is about $2e^{\alpha+1}$-competitive for large α, and moreover showed that any algorithm must be $\Omega((4/3)^\alpha)$-competitive (as a function of α). The deadline scheduling problem has also been considered for other measures such as minimizing the maximum speed and minimizing the maximum temperature [3].

For scheduling jobs without deadlines, a commonly used Quality of Service (QoS) measure is the *flow time* or more generally the *weighted flow time*. The flow time of a job is the time taken to complete a job since it is released. Clearly, minimizing flow time and minimizing energy usage are orthogonal objectives. To understand their tradeoffs, Albers and Fujiwara [2] proposed combining the dual objectives into a single objective of total flow time plus energy.[2] Albers and Fujiwara focused on scheduling jobs of unit size, and they gave an $8.3e((3+\sqrt{5})/2)^\alpha$-competitive algorithm for minimizing unweighted flow time plus energy. Bansal, Pruhs and Stein [4] considered the more general problem of minimizing weighted flow time plus energy. They gave a 4-competitive algorithm for jobs of unit size and weight. They also gave a $\mu_\epsilon\gamma$-competitive algorithm for jobs of arbitrary size and weight, where ϵ can be any positive constant, $\mu_\epsilon = \max\{(1+1/\epsilon),(1+\epsilon)^\alpha\}$ and $\gamma = \max\{2, \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-1/(\alpha-1)}}\}$. There are a large number of other related works for dynamic speed scaling (in the infinite speed model) and more generally for power management. We refer the readers to a survey by Irani and Pruhs [15] for more details.

Even though the infinite speed model is rather unsatisfying to model a real processor, it is a convenient theoretical model to work with. Typically it allows the algorithm to focus only on the speed setting aspect. For example in the deadline scheduling problem described above, as all jobs can always be completed, any algorithm can be assumed to schedule jobs by Earliest Deadline First (EDF) and hence it only needs to specify the speed at any given time. Moreover, the only relevant measures of the quality of the schedule are energy related. Another important reason is that it gives the online algorithm flexibility to recover from past mistakes; for example if the algorithm realizes that it has been working too slowly thus far, it can always try to catch up by speeding up. This often makes the algorithm design easier and more amenable to analysis.

[1] This does not hold at very low speeds due to leakage power effects that do not scale with speed.

[2] Without loss of generality, by changing the units of time or energy if necessary, it can be assumed that the user is willing to spend one unit of energy to improve one unit of flow time.

Recently Chan et al. [10] introduced the *bounded* speed model, where the processor speed can vary between 0 and some maximum T. They considered the deadline scheduling problem in this model. Note that when the maximum speed is bounded, even the optimal offline algorithm may not be able to complete all jobs. A natural objective is to maximize the throughput, defined as the total work of jobs that are successfully completed by their deadlines. In traditional scheduling (where speed is fixed) Koren and Shasha [16] gave an algorithm D^{over} which is 4-competitive on throughput. Moreover Baruah et al. [7] showed that this is the best possible throughput competitive ratio for any online algorithm. Thus running D^{over} at speed T is clearly 4-competitive for throughput; however it can be arbitrarily worse with respect to energy. Chan et al. [10] considered energy efficient algorithms for throughput maximization. They designed an online algorithm that is 14-competitive for throughput, and its energy consumption is at most $O(1)$ times that of any offline solution that maximizes the throughput.

1.1 Our Results

We consider two problems in the bounded speed model: energy efficient algorithms for throughput maximization, and that for minimizing weighted flow time.

Throughput maximization. We first discuss the throughput maximization problem. Our main result is an algorithm Slow-D which matches the optimum throughput guarantee of Koren and Shasha [16] while being $O(1)$-competitive for energy.

Theorem 1. *There is an online algorithm Slow-D that is 4-competitive with respect to throughput and* $(\alpha^\alpha + \alpha^2 4^\alpha)$*-competitive with respect to energy.*

Roughly speaking, Slow-D is a combination of OA and D^{over}. At any time, if all the remaining jobs can be completed using speed T, Slow-D admits all jobs and runs at the same speed as OA; otherwise, i.e., not all jobs can be completed, Slow-D uses the same job selection rule as D^{over} and runs at speed T. Hence, Slow-D uses a more sophisticated job selection method than the 14-competitive algorithm of [10], and differs from D^{over} by running at a slower speed when there is little work remaining. To prove the competitive ratio of Slow-D, the main novelty is a tighter analysis which accounts for the jobs that D^{over} can complete but Slow-D may miss due to the slower speed.

The setting we have considered so far is *overloaded* in that there may be too much work and no algorithm can finish all the jobs even running at the maximum speed at all times. A special case is the *underloaded* setting where all jobs can be completed by running at the maximum speed at all times. In traditional scheduling with a fixed speed processor, running EDF clearly completes all jobs and hence is 1-competitive for throughput. Yet no energy efficient algorithm can be 1-competitive with respect to throughput[3]. However, we can show that near-optimum throughput can be achieved in the underloaded setting. In particular, we have devised an algorithm called TimeSlot(ϵ) that is $(1+\epsilon)$-competitive for throughput and $(1+1/\epsilon)^\alpha \alpha^\alpha$-competitive for energy. Details will be given in the full paper.

[3] To be 1-competitive, an algorithm must always run at maximum speed on whatever little work has arrived thus far, otherwise the adversary can release new jobs to make the algorithm fail to complete all jobs.

To achieve 1-competitiveness in throughput, we consider resource augmentation where the online algorithm is allowed to relax its maximum speed to make up for its lack of future knowledge. In the fixed speed setting without energy concern, Lam and To [17] gave a 1-throughput competitive algorithm using a $2T$-speed processor. Chan et al. [10] gave an energy efficient algorithm that is $(1 + 1/\epsilon)$-competitive for throughput by relaxing the maximum speed to $(1 + \epsilon)T$. We can show that for the overloaded setting, there is an online algorithm that is 1-competitive for throughput and $(1 + 2/\epsilon)^\alpha(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive for energy when using maximum speed $(2 + \epsilon)T$; and for the underloaded setting, there is an online algorithm that is 1-competitive for throughput and $(1 + 1/\epsilon)^\alpha \alpha^\alpha$-competitive for energy when using maximum speed $(1 + \epsilon)T$.

Minimizing weighted flow time. We now discuss minimizing weighted flow time plus energy in the bounded speed model. Our results are obtained by first obtaining a guarantee for weighted *fractional* flow time plus energy (see Section 2 for definition) and then rounding it. We first consider the case of jobs with unit size and weight. We are able to show that there are online algorithms that are respectively 2-competitive for fractional flow time plus energy, and 4-competitive for total flow time plus energy.

In this paper we focus on the case of jobs with arbitrary size and weight. Note that when the maximum speed T is very small (say T^α is less than the smallest job weight), any algorithm must work at speed T whenever there is unfinished work. In this case, the problem reduces to the classic problem of minimizing weighted flow time on a fixed speed processor, without any energy concern [11, 6, 8]. For this problem it has been recently shown that no $O(1)$-competitive algorithm is possible without resource augmentation [5]. Thus we need to relax the maximum speed of the online algorithm to $(1 + \epsilon)T$ in order to be $O(1)$-competitive for total weighted flow time plus energy.

Theorem 2. *For jobs with arbitrary size and weight, there is an online algorithm that is* $((2 + o(1))\alpha/\ln \alpha)$*-competitive for fractional weighted flow time plus energy. Furthermore, there is an online algorithm that given any* $\epsilon > 0$*, uses a processor with maximum speed* $(1 + \epsilon)T$*, and is* $\mu_\epsilon((2 + o(1))\alpha/\ln \alpha)$*-competitive for total weighted flow time plus energy, where* $\mu_\epsilon = \max\{(1 + 1/\epsilon), (1 + \epsilon)^\alpha\}$.

Note that both guarantees mentioned above essentially match those of [4] for the special case of the infinite speed model. Our results here are based on generalizing and extending the analysis in [4]. As we will discuss in Section 2, the algorithms in the infinite speed model set the speed such that at any time the rate of increase of flow time is equal to the rate of increase of energy. However, this is not always possible in the bounded speed model, which makes the analysis substantially harder.

Discrete speed levels. Our results can be easily adapted to discrete speed levels. The idea is to set the maximum speed to the highest speed level and round up the speed function to the next higher level. It maintains the performance on throughput and flow-time, while the energy usage is increased by at most a factor of Δ^α, where Δ is the maximum ratio of two consecutive non-zero speed levels.

2 Preliminaries

Throughout the paper we assume jobs arrive online over time. We use $r(J)$, $p(J)$, $d(J)$ and $w(J)$ (wherever applicable) to denote the release time, size (processing requirement), deadline and weight respectively of a job J. We consider scheduling algorithms for a single processor, and assume that jobs can be preempted arbitrarily without any penalty. The throughput of a schedule is defined as the total size of jobs that are successfully completed by their deadlines (no partial credit is obtained for incomplete jobs). All our results for throughput assume that the jobs are unweighted.

Given a schedule, the *flow time* of a job is the amount of time since this job is released until it completes. Equivalently, flow time of a job is simply its total cost if it pays one unit for each unit of time until it completes. The *fractional* flow time of a job is defined as its total cost if at each time unit it pays an amount equal to its unfinished fraction. The weighted flow time of a job and the fractional weighted flow time are defined analogously. Often fractional weighted flow time is much more convenient to work with. The (online) Highest Density First (HDF) algorithm, which at any time works on the job with the highest weight to size ratio, is optimal for minimizing fractional weighted flow time. On the other hand, no O(1)-competitive online algorithm exists for weighted flow time [5]. Note that at any time the total weighted flow time increases at a rate equal to the total weight of currently unfinished jobs, and the total energy usage increases at a rate s^α where s is the current speed. To trade off energy and flow time, a natural algorithm first proposed by Albers and Fujiwara [2] and analyzed in [4], is to set the speed s such that s^α is equal to the unfinished weight. However, this cannot be done in the bounded speed model as the unfinished weight can be much larger than T^α.

Algorithm OA. We explain the Optimal Available (OA) algorithm proposed by Yao, Demers and Shenker [24] for the infinite speed model. Note that in the infinite speed model it suffices to specify the speed at any time (jobs are always scheduled by EDF). Roughly speaking, the OA algorithm is the "laziest" possible algorithm that at any time works at the average speed just enough to complete all jobs feasibly. Formally, let $p_t(x)$ denote the amount of unfinished work at time t that has deadline within the next x units, then $p_t(x)/x$ is a lower bound on the average rate at which any feasible algorithm must work. At any time t, OA works at speed $s_{OA}(t) = \max_x p_t(x)/x$. For example, consider the instance where a job of size 1 and deadline n arrives at each of the times $0, 1, 2, \ldots, n-1$. The optimum schedule works at speed 1, and incurs total energy of n. On the other hand, OA starts with speed $1/n$ during $[0, 1]$, and has speed $1/n+\ldots+1/(n-i)$ during $[i, i+1]$ for $i = 0, \ldots, n-1$. Note that during $[n-1, n]$, OA consumes energy at rate about $(\ln n)^\alpha$, which is substantially larger than that consumed by the optimum solution at any time. Interestingly however, OA is α^α-competitive with respect to energy [3].

Another useful view of OA is the following. At any time t, OA computes the optimum energy schedule for the unfinished work assuming that no more jobs will arrive, and proceeds accordingly. When more jobs arrive in the future it recomputes this schedule and continues. Let $s^t_{OA}(t')$ denote the speed function (for times $t' > t$) computed by OA at time t, assuming no more jobs arrive after time t. s^t_{OA} is a decreasing piecewise

step function, i.e., it has speed s_j during $I_j = [t_j, t_{j+1}]$ for $j = 0, 1 \ldots$, where $s_0 > s_1 > s_2 > \ldots$, and $t_0 = t$. Moreover, only jobs with deadlines in the interval I_j are executed during I_j. The intervals I_j change when new jobs arrive, but for a fixed t' the computed speed $s^t_{OA}(t')$ can only increase with time t.

3 Energy Efficient Throughput Maximization

Recall that no algorithm can be better than 4-competitive for throughput even without energy concern. Moreover, executing D^{over} at the maximum speed T is 4-competitive for throughput, but it does not guarantee energy efficiency. In this section, we give an energy efficient algorithm Slow-D that is optimally competitive for throughput.

3.1 Algorithm Description

Consider the execution of OA with an unbounded speed processor. Let $s_{OA}(t)$ denote the speed of OA at time t, and without loss of generality we assume that OA follows the EDF policy. We design an algorithm Slow-D that simulates OA and makes decisions based on its own state and that of OA. At any time t, Slow-D works at speed $\tilde{s}(t) = \min\{s_{OA}(t), T\}$. Note that unlike OA (which works in infinite speed model), Slow-D may not complete all the jobs, so we need to specify carefully a job selection and execution strategy.

We first define the notion of down-time(t) that is critically used by Slow-D. At any time t, consider the schedule s^t_{OA} computed by OA assuming no new jobs arrive. We define down-time(t) as the latest time $t' \geq t$ such that the speed $s^t_{OA}(t') \geq T$. If $s_{OA}(t) < T$, and no such t' exists, we set down-time(t) to be the last time before t when the speed was at least T (or 0 if the speed was always below T). By the nature of OA, down-time(t) is a non-decreasing function of t no matter how jobs arrive. At any time t, we label all released jobs (including those OA has completed) based on down-time(t). A job J is called *t-urgent* if $d(J) \leq$ down-time(t), and is called *t-slack* otherwise. Note that a t-slack job may turn into a t'-urgent job at a later time $t' > t$. However, since down-time is non-decreasing, a t-urgent jobs stays urgent until it completes or is discarded. We call a job *slack* if it always remains slack during its entire lifespan; otherwise, it is called *urgent*. We now describe Slow-D.

Slow-D stores all released jobs in two queues $\mathrm{Q}_{\mathrm{work}}$ and $\mathrm{Q}_{\mathrm{wait}}$. At any time t, it processes the job in $\mathrm{Q}_{\mathrm{work}}$ with the earliest deadline at speed $\tilde{s}(t) = \min\{s_{OA}(t), T\}$. As we shall see $\mathrm{Q}_{\mathrm{wait}}$ is empty whenever $\mathrm{Q}_{\mathrm{work}}$ is empty. Slow-D admits jobs as follows:

Job arrival. Consider a job J released at time r_j. J is admitted to $\mathrm{Q}_{\mathrm{work}}$ if either J is r_j-slack, or J and the remaining work of all r_j-urgent jobs in $\mathrm{Q}_{\mathrm{work}}$ can be completed using speed T. Otherwise, J is admitted to $\mathrm{Q}_{\mathrm{wait}}$. Note that jobs admitted to $\mathrm{Q}_{\mathrm{wait}}$ are all urgent.
We say that an *urgent period* begins at r_j if $\mathrm{Q}_{\mathrm{work}}$ contains no urgent job before r_j and J is an urgent job admitted into $\mathrm{Q}_{\mathrm{work}}$.
Latest starting time (*ℓst*) interrupt. Whenever a job J in $\mathrm{Q}_{\mathrm{wait}}$ reaches its latest starting time, i.e., current time $t = d(J) - (p(J)/T)$, it raises an *ℓst interrupt*. At an interrupt we either discard J or else expel all t-urgent jobs in $\mathrm{Q}_{\mathrm{work}}$ to make room for J as follows:

In the current urgent period[4], let J_0 be the last job admitted from $\mathrm{Q}_{\mathrm{wait}}$ to $\mathrm{Q}_{\mathrm{work}}$ (if no jobs have been admitted from $\mathrm{Q}_{\mathrm{wait}}$ so far, let J_0 be a dummy job of size zero admitted just before the current period starts). Consider all the jobs ever admitted to $\mathrm{Q}_{\mathrm{work}}$ that have become urgent after J_0 has been admitted to $\mathrm{Q}_{\mathrm{work}}$, and let W denote the total original size of these jobs. If $p(J) > 2(p(J_0)+W)$, all t-urgent jobs in $\mathrm{Q}_{\mathrm{work}}$ are expelled and J is admitted to $\mathrm{Q}_{\mathrm{work}}$.

Job completion. When a job J completes at time t, remove it from $\mathrm{Q}_{\mathrm{work}}$. If $\mathrm{Q}_{\mathrm{work}}$ contains no more t-urgent job, the current urgent period ends.

Note that the above *urgent period* is defined in such a way that at any time t during an urgent period, only t-urgent job is being processed.

3.2 Analysis

As Slow-D works at speed $\min(s^t_{OA}, T)$, the energy competitiveness follows directly from the result of [10].

Theorem 3. [10] *Any algorithm that works according to the speed function* $\tilde{s}(t) = \min(s^t_{OA}, T)$ *is* $(\alpha^\alpha + \alpha^2 4^\alpha)$*-competitive for energy against any offline algorithm that maximizes the throughput.*

Our goal now is to show that Slow-D is 4-competitive with respect to throughput. We partition the job sequence I into three sets. Let I_ℓ be the set of jobs admitted to $\mathrm{Q}_{\mathrm{wait}}$ upon arrival (these may join $\mathrm{Q}_{\mathrm{work}}$ later after raising *ℓst* interrupts). Among the jobs admitted immediately to $\mathrm{Q}_{\mathrm{work}}$ upon arrival, we let I_s denote those that are slack through their lifespan, and let I_t denote the ones that become urgent at some time before they expire. Note that I_s, I_t and I_ℓ are disjoint. Indeed, I_s and $I_t \cup I_\ell$ are respectively the sets of all slack jobs and all urgent jobs in I.

We first show that all slack jobs are completed by Slow-D.

Lemma 1. *Slow-D completes all jobs in* I_s.

Proof. Consider the execution of slack jobs under the unbounded speed OA. Since these jobs never become urgent, they are always executed at speed strictly less than T by OA. On the other hand, whenever OA runs at speed T or above, OA is working on an urgent job. To ease our discussion, for any time t, we call t a peak time if OA is running at speed T or above; otherwise, t is said to be a leisure time. At any leisure time t, Slow-D as well as OA can only work on some t-slack job (because down-time$(t) < t$ and all t-urgent jobs must have deadline before t). A t-slack job can never been executed at any peak time before t by OA; yet this might be possible for Slow-D. Together with the fact that both Slow-D and OA are using EDF, we conclude that at any leisure time t, Slow-D does not lag behind OA on any t-slack job (including any slack job). Thus all the slack jobs are completed by Slow-D. □

We will show that Slow-D completes enough jobs in I_t and I_ℓ. In particular, we consider each urgent period $P = [S_P, E_P]$ separately and derive a lower bound of the total size

[4] As we shall see *ℓst* interrupts occur only during urgent periods.

of urgent jobs that Slow-D completes in P. The following lemma is key to our lower bound. It requires two notations: join(P) denotes the total size of jobs in I_t that become urgent at some time in P. Let J^* be the latest-deadline job in I_ℓ that is released during P (irrespective of whether it is admitted to $\mathrm{Q}_{\mathrm{work}}$ later or not). We define a *secured interval* $P' = [S'_P, E'_P]$ for P, where $S'_P = S_P$ and $E'_P = \max\{d(J^*), E_P\}$.

Lemma 2. *For any urgent period P, the total size of urgent jobs completed by Slow-D is at least $\frac{1}{4}$ of (join$(P) + |P'| \times T$).*

Before proving Lemma 2, we show how it implies our main result. Let $p(I_t)$ be the total size of all jobs in I_t. Let span(I_ℓ) be the union of the spans of all jobs in I_ℓ, which may consist of a number of disjoint intervals, and let $|\text{span}(I_\ell)|$ be the total length of these intervals.

Lemma 3. *The total size of urgent jobs completed by Slow-D is at least $\frac{1}{4}(p(I_t) + |$span$(I_\ell)| \times T)$.*

Proof. Let C be the collection of all urgent periods. By Lemma 2, the total size of urgent jobs completed by Slow-D over all urgent periods is at least $\frac{1}{4}\sum_{P\in C}(\text{join}(P) + |P'| \times T)$.

For each job $J \in I_t$, J joins $\mathrm{Q}_{\mathrm{work}}$ during some urgent period, so $p(I_t) = \sum_{P\in C} \text{join}(P)$. Similarly, for each job $J \in I_\ell$, J is released during some urgent period, so $|\text{span}(I_\ell)| \le \sum_{P\in C} |P'|$. Summing the above two equality and inequality, we obtain $\sum_{P\in C}(\text{join}(P) + |P'| \times T) \ge p(I_t) + |\text{span}(I_\ell)| \times T$. □

Theorem 4. *Slow-D is 4-competitive on throughput.*

Proof. Let $p(I_s)$ be the total size of all jobs in I_s. By Lemma 1 and 3, the total size of jobs completed by Slow-D is at least $p(I_s) + (1/4) \times (p(I_t) + |\text{span}(I_\ell)| \times T)$. Clearly, any offline algorithm can complete at most $p(I_s)$ work on jobs in I_s, at most $p(I_t)$ work on jobs in I_t and at most $|\text{span}(I_\ell)| \times T$ work on jobs in I_ℓ, which implies the result. □

The rest of this section proves Lemma 2. Consider an arbitrary urgent period $P = [S_P, E_P]$. At S_P, some urgent jobs start to appear in $\mathrm{Q}_{\mathrm{work}}$. These jobs may be released at S_P, or already in $\mathrm{Q}_{\mathrm{work}}$ before S_P but just become urgent as down-time(S_P) becomes greater than their deadlines. As time goes on, more urgent jobs may appear in $\mathrm{Q}_{\mathrm{work}}$, and jobs in $\mathrm{Q}_{\mathrm{wait}}$ may raise *ℓst* interrupts. Assume that k jobs $J_1, J_2, ..., J_k$ in $\mathrm{Q}_{\mathrm{wait}}$ are admitted successfully to $\mathrm{Q}_{\mathrm{work}}$ during P at times $L_1 \le L_2 \le \cdots \le L_k$, respectively. For notational convenience, we let J_0 and J_{k+1} be jobs of size zero, admitted at $L_0 = S_P$ just before any urgent job appears in $\mathrm{Q}_{\mathrm{work}}$ and at $L_{k+1} = E_P$ just after all urgent jobs are removed from $\mathrm{Q}_{\mathrm{work}}$, respectively. Note that during P, Slow-D always runs at speed T and works on urgent jobs. It completes at least J_k and all jobs in $\mathrm{Q}_{\mathrm{work}}$ that are found to be urgent after J_k is admitted and before E_P, as these urgent jobs are not expelled by an interrupt.

We now show a useful property about *ℓst* interrupts of jobs in I_ℓ.

Lemma 4. *Every job $J \in I_\ell$ that is released during an urgent period P must raise an ℓst interrupt in P (irrespective of whether it is admitted to $\mathrm{Q}_{\mathrm{work}}$ later or not).*

Proof. Consider the time $r(J)$ during P when J is released. Since J was placed in $\mathrm{Q}_{\mathrm{wait}}$, there was more than $T(d(J) - r(J) - (p(J)/T))$ urgent work in $\mathrm{Q}_{\mathrm{work}}$ at $r(J)$. At time progresses, more jobs in I_t may join $\mathrm{Q}_{\mathrm{work}}$, or some jobs in I_ℓ may be admitted successfully. In either case, the total amount of admitted urgent work can only increase. Thus, $E_P > r(J) + (d(J) - r(J) - (p(J)/T)) = d(J) - (p(J)/T)$ which is exactly when J raises the interrupt. □

To lower bound the work completed by Slow-D in P, we refine the notation join as follows: For any i, i' such that $0 \leq i < i' \leq k+1$, let $\mathrm{join}(J_i, J_{i'})$ be the total size of jobs in I_t that become urgent after J_i and before $J_{i'}$ are admitted to $\mathrm{Q}_{\mathrm{work}}$. Note that $\mathrm{join}(P) = \mathrm{join}(J_0, J_{k+1})$.

Following the above discussion of P, the total size of urgent jobs that Slow-D completes during P is at least $p(J_k) + \mathrm{join}(J_k, J_{k+1})$. To prove Lemma 2, if suffices to show that $p(J_k) + \mathrm{join}(J_k, J_{k+1})$ is at least $\frac{1}{4}(\mathrm{join}(P) + |P'| \times T)$. We prove this via an inductive argument whose statement is described by Lemma 5.

Lemma 5. *For any urgent period $P = [S_P, E_P]$ and its secured interval P', we have that* (1) $p(J_i) \geq$ *join*$(J_0, J_i) + (L_i - S_P) \times T$, *for* $i = 0, 1, \ldots, k$, *and* (2) *join*$(J_0, J_k) + |P'| \times T \leq 4 \times p(J_k) + 3 \times$ *join*(J_k, J_{k+1}).

Proof. We prove Statement (1) by induction. For $i = 0$, $p(J_0) = 0$, $\mathrm{join}(J_0, J_i)$, and $(L_0 - S_P)$ all equal to zero. Assume the claim is true for i. For $i+1$ (s.t. $i+1 \leq k$),

$$\begin{aligned}
&\mathrm{join}(J_0, J_{i+1}) + (L_{i+1} - S_P) \times T \\
&= \mathrm{join}(J_0, J_i) + \mathrm{join}(J_i, J_{i+1}) + ((L_{i+1} - L_i) + (L_i - S_P)) \times T \\
&\leq p(J_i) + \mathrm{join}(J_i, J_{i+1}) + (L_{i+1} - L_i) \times T \qquad \text{(by induction)} \\
&\leq 2 \times (p(J_i) + \mathrm{join}(J_i, J_{i+1})) < p(J_{i+1}) \ .
\end{aligned}$$

The second last step follows as the maximum work Slow-D can process during $[L_i, L_{i+1}]$ is at most $(p(J_i) + \mathrm{join}(J_i, J_{i+1}))$. The final step follows as J_{i+1} is admitted from $\mathrm{Q}_{\mathrm{wait}}$.

Before proving Statement (2), we observe the following bounds of J^*:

$$(d(J^*) - E_p) \times T \leq p(J^*) \leq 2(p(J_k) + \mathrm{join}(J_k, J_{k+1})) \ .$$

The first inequality follows from Lemma 4. If J^* is admitted to $\mathrm{Q}_{\mathrm{work}}$, then $J^* = J_i$ for some $i \leq k$; otherwise, $p(J^*) \leq 2(p(J_i) + \mathrm{join}(J_i, J_{i+1}))$ for some $i \leq k$. In both cases, $p(J^*) \leq 2(p(J_k) + \mathrm{join}(J_k, J_{k+1}))$. Thus,

$$\begin{aligned}
\mathrm{join}(J_0, J_k) + |P'| \times T &= \mathrm{join}(J_0, J_k) + (E_P - S_P) \times T + \max\{d(J^*) - E_P, 0\} \times T \\
&\leq \mathrm{join}(J_0, J_k) + (L_k - S_P) \times T + (E_P - L_k) \times T + p(J^*) \\
&\leq p(J_k) + (E_P - L_k) \times T + p(J^*) \qquad (\text{by (1)}) \\
&\leq p(J_k) + p(J_k) + \mathrm{join}(J_k, J_{k+1}) + p(J^*) \\
&\leq 4 \times p(J_k) + 3 \times \mathrm{join}(J_k, J_{k+1}) \ .
\end{aligned}$$

□

4 Minimizing Weighted Flow Time Plus Energy

We first consider the problem of minimizing the total fractional weighted flow time plus energy on a variable speed processor, in the bounded speed setting. Let $w_a(t)$ and $w_o(t)$ denote the total fractional weight of jobs in the online algorithm and some fixed optimum schedule at time t. Consider the algorithm that works at speed $s_a(t) = \min\{w_a(t)^{1/\alpha}, T\}$ and schedules jobs using HDF. We prove Theorem 5, which matches the guarantee of [4] up to lower order terms, who showed it for the case of $T = \infty$. Then by the same technique in [4], we can use this result to obtain a competitive algorithm for total (integral) weighted flow time plus energy using a processor with maximum speed $(1+\epsilon)T$ (recall that the speed augmentation is necessary to obtain a constant guarantee for this measure). We begin with a useful lemma relating the total fractional weight of jobs under both online and any offline algorithm.

Lemma 6. *For any offline algorithm, at any time t we have that $w_a(t) - w_o(t) \leq T^\alpha$.*

Proof. Clearly the result holds if $w_a(t) \leq T^\alpha$, and hence we consider the case when $w_a(t) \geq T^\alpha$. Thus at the current time t, the speed $s_a(t) = T$. Let t_0 be the latest time before t such that the speed just before t_0 (we denote this time by t_0^-) is less than T. Consider the set S of jobs that arrive during $[t_0, t]$ and let $w(S)$ denote their total weight. Let $\tilde{w}$ denote the amount of fractional weight completed by the offline algorithm by time t restricted to the jobs in S. As our algorithm always schedules the highest density job, the amount of fractional weight completed by our algorithm during $[t_0, t]$ when considered over all possible jobs (and not just jobs in S) is at least $\tilde{w}$. Thus it follows that $w_a(t) - w_a(t_0^-) \leq w(S) - \tilde{w} \leq w_o(t)$. This implies that $w_a(t) - w_o(t) \leq w_a(t_0^-)$ which is at most T^α since $s_a(t_0^-) < T$. □

Theorem 5. *The algorithm described above is $2\alpha/(\alpha - (\alpha-1)^{1-1/(\alpha-1)}) = ((2 + o(1))\alpha/\ln\alpha)$-competitive with respect to fractional weighted flow time plus energy.*

Proof. For notational ease, we will drop the time t from the notation, since all variables are understood to be functions of t. We will show that there is a potential function Φ, such that the value of the function is 0 at the beginning and at end of the schedule, never increases upon the release of a job, and satisfies the condition $\frac{d\Phi}{dt} \leq c(w_o + s_o^\alpha) - (w_a + s_a^\alpha)$ for some $c \geq 1$. As observed by [4], this proves that the algorithm is c-competitive for total fractional weighted flow plus energy. In fact as $w_a \geq s_a^\alpha$ for our algorithm, it suffices to show that

$$\frac{d\Phi}{dt} \leq c(w_o + s_o^\alpha) - 2w_a \ . \tag{1}$$

Consider the potential function $\Phi = \frac{\eta}{(\beta+1)} \int_0^\infty \left(w_a(h)^{\beta+1} - (\beta+1)w_a(h)^\beta w_o(h)\right) dh$, where $\beta = 1 - 1/\alpha$, and $w_a(h)$ and $w_o(h)$ are the total fractional weight of active jobs that have an inverse density (defined as the size of a job divided by its weight) of at least h under our algorithm and for some fixed optimum schedule respectively. The constant η will be chosen appropriately later.

That the potential function does not increase upon job arrival follows from Lemma 10 in [4] which implies that the content of the integral $w_a(h)^{\beta+1} - (\beta+1)w_a(h)^\beta w_o(h)$ never increases when both $w_a(h)$ and $w_o(h)$ are increased by the same amount. Thus

we analyze the case when a job is being executed. The analysis starts similar to [4]. We let m_a and m_o be the inverse density of the job being executed by our algorithm and the optimum schedule at the current time. Then we have

$$\begin{aligned}
\frac{d\Phi}{dt} &= -\eta\Big(\int_0^{\infty} (w_a(h)^{\beta} - \beta w_a(h)^{\beta-1} w_o(h)) \frac{dw_a(h)}{dt} - w_a(h)^{\beta} \frac{dw_o}{dt} dh\Big) \\
&= -\eta\Big(\int_0^{m_a} (w_a(h)^{\beta} - \beta w_a(h)^{\beta-1} w_o(h)) \frac{s_a}{m_a} dh\Big) + \eta\Big(\int_0^{m_o} w_a(h)^{\beta} \frac{s_o}{m_o} dh\Big) \\
&\leq -\eta(w_a^{\beta} s_a - \beta w_a^{\beta-1} w_o s_a) + \eta \int_0^{m_o} w_a^{\beta} \frac{s_o}{m_o} dh\Big) \\
&= -\eta w_a^{\beta} s_a + \eta\beta w_a^{\beta-1} w_o s_a + \eta w_a^{\beta} s_o \\
&\leq -\eta w_a^{\beta} s_a + \eta\beta w_a^{\beta-1} w_o s_a + \eta(\mu\beta w_a + \beta s_o^{\alpha}) \ . \qquad (2)
\end{aligned}$$

The second step follows from the HDF nature of our algorithm. The final step follows by applying Young's inequality with $a = w_a^{\beta}$, $b = s_0$, $p = 1/\beta$, and $q = \alpha$, and $\mu = (\alpha-1)^{-1/(\alpha-1)}$, which gives $w_a^{\beta} s_o \leq \mu\beta w_a + \left(\frac{1}{\mu}\right)^{\alpha\beta} \left(\frac{s_o^{\alpha}}{\alpha}\right) = \mu\beta w_a + \beta s_o^{\alpha}$.

We now show that (1) holds. We divide the analysis in three different cases. In each case we show that $d\Phi/dt \leq -\eta(1-\mu\beta)w_a + \eta(w_0 + s_o^{\alpha})$. Setting $\eta = 2/(1-\mu\beta)$, this implies a competitive ratio of $2/(1-\mu\beta)$, which by substituting the values of μ and β gives our claimed guarantee. The first case is when $w_o \geq w_a \geq T^{\alpha}$. Observe that in this case $s_a = T$. Starting with (2),

$$\begin{aligned}
\frac{d\Phi}{dt} &\leq \eta(-w_a^{\beta} T + \beta w_a^{\beta-1} w_o T + \mu\beta w_a + \beta s_o^{\alpha}) \\
&< \eta(-w_a^{\beta} T + w_a^{\beta-1} w_o T + \mu\beta w_a + s_o^{\alpha}) \qquad (\text{as } \beta < 1) \\
&= \eta(w_a^{\beta-1}(w_o - w_a)T + \mu\beta w_a + s_o^{\alpha}) \\
&\leq \eta(w_a^{\beta-1}(w_o - w_a) w_a^{1/\alpha} + \mu\beta w_a + s_o^{\alpha}) \qquad (\text{as } w_a > T^{\alpha} \text{ and } w_o > w_a) \\
&= -\eta(1-\mu\beta)w_a + \eta w_o + \eta s_o^{\alpha} \ .
\end{aligned}$$

The second case is when $w_a \geq T^{\alpha}$ and $w_a > w_o$. Again, $s_a = T$. 79

$$\begin{aligned}
\frac{d\Phi}{dt} &\leq \eta(-w_a^{\beta} s_a + \beta w_a^{\beta-1} w_o s_a + \mu\beta w_a + \beta s_o^{\alpha}) \\
&= -\eta w_a^{\beta-1}(w_a - \beta w_o)T + \eta\mu\beta\, w_a + \eta s_o^{\alpha} \qquad (\text{as } \beta < 1) \\
&\leq -\eta w_a^{\beta-1}(w_a - \beta w_o)(w_a - w_o)^{1/\alpha} + \eta\mu\beta w_a + \eta s_o^{\alpha} \quad (\text{Lemma 6 and } (w_a - \beta w_o) > 0) \\
&\leq -\eta w_a^{\beta-1} w_a^{1-\beta}(w_a - w_o)^{\beta}(w_a - w_o)^{1/\alpha} + \eta\mu\beta w_a + \eta s_o^{\alpha} \\
&= -\eta(1-\mu\beta)w_a + \eta w_o + \eta s_o^{\alpha} \ .
\end{aligned}$$

The third step follows as $(w_a - w_o) < T^{\alpha}$ by Lemma 6. The fourth step follows by using that fact that $(1-\beta x) \geq (1-x)^{\beta}$ for any $0 \leq \beta \leq 1$ and $0 \leq x < 1$, which implies that $(w_a - \beta w_o) \geq w_a^{1-\beta}(w_a - w_o)^{\beta}$.

Finally we consider the case that $w_a < T$. Here $s_a = w_a^{1/\alpha}$, and this is exactly the case handled by [4]. We reprove it here for completeness. $\frac{d\Phi}{dt} \leq -\eta w_a^{\beta} s_a + \eta\beta w_a^{\beta-1} w_o s_a + \eta\beta\mu w_a + \eta\beta s_o^{\alpha} = -\eta(1-\beta\mu)w_a + \eta\beta w_o + \eta\beta s_o^{\alpha} \leq -\eta(1-\beta\mu)w_a + \eta w_o + \eta s_o^{\alpha}$. Thus the result follows. □

References

1. http://www-03.ibm.com/chips/power/powerpc/newsletter/sep2004/technical2.html
2. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 621–633. Springer, Heidelberg (2006)
3. Bansal, N., Kimbrel, T., Pruhs, K.: Dynamic speed scaling to manage energy and temperature. Journal of the ACM 51(1) (2007)
4. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: Proc. SODA, pp. 805–813 (2007)
5. Bansal, N., Chan, H.L.: Weighted flow time does not have O(1) competitive algorithms (manuscript)
6. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. In: Proc. SODA, pp. 508–516 (2003)
7. Baruah, S., Koren, G., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D.: On-line scheduling in the presence of overload. In: Proc. FOCS, pp. 100–110 (1991)
8. Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A., Pruhs, K.: Online Weighted Flow Time and Deadline Scheduling. In: Proc. RANDOM-APPROX, pp. 36–47 (2001)
9. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro. 20(6), 26–44 (2000)
10. Chan, H.L., Chan, W.T., Lam, T.W., Lee, L.K., Mak, K.S., Wong, P.: Energy efficient online deadline scheduling. In: Proc. SODA, pp. 795–804 (2007)
11. Chekuri, C., Khanna, S., Zhu, A.: Algorithms for minimizing weighted flow time. In: Proc. STOC, pp. 84–93 (2001)
12. Dertouzos, M.L.: Control robotics: the procedural control of physical processes. In: Proc. IFIP Congress, pp. 807–813 (1974)
13. Grunwald, D., Levis, P., Farkas, K.I., Morrey, C.B., Neufeld, M.: Policies for dynamic clock scheduling. In: Proc. OSDI, pp. 73–86 (2000)
14. Hardy, G.H., Littlewood, J.E., Polya, G.: Inequalities. Cambridge University Press, Cambridge (1952)
15. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News (2005)
16. Koren, G., Shasha, D.: D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM J. Comput. 24(2), 318–339 (1995)
17. Lam, T.W., To, K.K.: Performance Guarantee for Online Deadline Scheduling in the Presence of Overload. In: Proc. SODA, pp. 755–764 (2001)
18. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocations for tree-structured tasks. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 283–296. Springer, Heidelberg (2005)
19. Li, M., Yao, F.: An efficient algorithm for computing optimal discrete voltage schedules. SIAM J. Comput. 35(3), 658–671 (2005)
20. Mudge, T.: Power: A first-class architectural design constraint. Computer 34(4), 52–58 (2001)
21. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proc. SOSP, pp. 89–102 (2001)
22. Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the best response for your erg. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 14–25. Springer, Heidelberg (2004)
23. Weiser, M., Welch, B., Demers, A., Shenker, S.: Scheduling for reduced CPU energy. In: Proc. OSDI, pp. 13–23 (1994)
24. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. FOCS, pp. 374–382 (1995)

Faster Algorithms for Incremental Topological Ordering*

Bernhard Haeupler[1], Telikepalli Kavitha[2], Rogers Mathew[2], Siddhartha Sen[1], and Robert E. Tarjan[1,3]

[1] Princeton University, Princeton NJ 08544
{haeupler, sssix, ret}@cs.princeton.edu
[2] Indian Institute of Science, Bangalore, India.
{kavitha, rogers}@csa.iisc.ernet.in
[3] HP Laboratories, Palo Alto CA 94304

Abstract. We present two online algorithms for maintaining a topological order of a directed acyclic graph as arcs are added, and detecting a cycle when one is created. Our first algorithm takes $O(m^{1/2})$ amortized time per arc and our second algorithm takes $O(n^{2.5}/m)$ amortized time per arc, where n is the number of vertices and m is the total number of arcs. For sparse graphs, our $O(m^{1/2})$ bound improves the best previous bound by a factor of $\log n$ and is tight to within a constant factor for a natural class of algorithms that includes all the existing ones. Our main insight is that the two-way search method of previous algorithms does not require an ordered search, but can be more general, allowing us to avoid the use of heaps (priority queues). Instead, the deterministic version of our algorithm uses (approximate) median-finding; the randomized version of our algorithm uses uniform random sampling. For dense graphs, our $O(n^{2.5}/m)$ bound improves the best previously published bound by a factor of $n^{1/4}$ and a recent bound obtained independently of our work by a factor of $\log n$. Our main insight is that graph search is wasteful when the graph is dense and can be avoided by searching the topological order space instead. Our algorithms extend to the maintenance of strong components, in the same asymptotic time bounds.

1 Introduction

We consider two related problems on dynamic directed graphs: cycle detection and maintaining a topological order. These problems are closely connected, since a directed graph has a topological order if and only if it is acyclic. We present two algorithms for maintaining a topological order that asymptotically improve the best known time bounds for both sparse and dense graphs.

A *topological order* of a directed graph is a total order of the vertices such that for every arc (v, w), $v < w$. A directed graph has a topological order (and generally more than one) if and only if it is acyclic [9]. Given a fixed n-vertex, m-arc graph, a

* This paper combines the best results of [12] and [7]. The former gives two incremental topological ordering algorithms, with amortized time bounds per arc addition of $O(m^{1/2} + (n \log n)/m^{1/2})$ and $O(n^{2.5}/m)$. The latter, which was written with knowledge of the former, gives an algorithm with a bound of $O(m^{1/2})$.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 421–433, 2008.

topological order can be found in $O(n+m)$ time by either of two algorithms: repeated deletion of sources [13,14] or depth-first search [28]. The former method extends to the enumeration of all possible topological orderings.

In some applications, the graph is not fixed but changes over time. The *incremental topological ordering problem* is that of maintaining a topological order of a directed graph as arcs are added, stopping when addition of an arc creates a cycle. This problem arises in incremental circuit evaluation [2], pointer analysis [21], management of compilation dependencies [17,19], and deadlock detection [3].

In considering the incremental topological ordering problem, we shall assume that the vertex set is fixed and specified initially, and that the arc set is initially empty. Our ideas easily extend to support vertex as well as arc additions, with a time of $O(1)$ per vertex addition. Our methods also allow arc deletions, since deletion of an arc cannot create a cycle nor invalidate the current topological order, but our time bounds are only valid if there are no arc deletions. Our algorithms also extend to the maintenance of strong components under arc additions, in the same asymptotic time bounds, but we omit the details here because of space constraints.

One could of course recompute a topological order after each arc addition, but this would require $O(n+m)$ time per arc. Our goal is to do better. The incremental cycle detection and topological ordering problems have received much attention, especially recently. Shmueli [27] applied depth-first search to test for cycles in a directed graph subject to arc additions and deletions. As a heuristic to improve its efficiency, his method maintains a topological ordering, although he did not describe it as such. For the topological ordering problem, Marchetti-Spaccamela et al. [18] gave a one-way search algorithm that takes $O(n)$ amortized time per arc addition. Alpern et al. [2] gave a two-way search algorithm that handles batched arc additions and has a good time bound in an incremental model of computation. Katriel and Bodlaender [11] showed that a variant of the algorithm of Alpern et al. takes $O(\min\{m^{1/2}\log n, m^{1/2}+(n^2\log n)/m\})$ amortized time per arc addition. Liu and Chao [15] tightened this analysis to $\Theta(m^{1/2}+n^{1/2}\log n)$ per arc addition. Pearce and Kelly [20] gave an algorithm that they claimed was fast in practice on sparse graphs, although it has an inferior asymptotic time bound. Kavitha and Mathew [12] improved the results of Liu and Chao by presenting another variant of the algorithm of Alpern et al. with an $O(m^{1/2}+(n\log n)/m^{1/2})$ amortized time bound per arc addition. Ajwani et al. [1] gave an algorithm with an amortized time per arc addition of $O(n^{2.75}/m)$, better for dense graphs than the bound of Kavitha and Mathew. Independent of our work, Liu and Chao [16] modified the algorithm of Ajwani et al. to obtain an $\tilde{O}(n^{2.5}/m)$ amortized time bound per arc addition.

We generalize the two-way search method introduced by Alpern et al. and used in later algorithms by observing that the method does not require an ordered search (used in all previous algorithms) to be correct. This allows us to refine the general method into one that avoids the use of heaps (priority queues), but instead uses either approximate median-finding or random selection, resulting in an $O(m^{1/2})$ amortized time bound per arc addition. The randomized version of our algorithm is especially simple.

For dense graphs, we observe that graph search itself is wasteful because it can do unnecessary arc traversals. By searching the current topological order instead, we obtain

an $O(n^{2.5}/m)$ amortized time bound per arc addition. This algorithm is also simple, although its analysis relies on a linear program bound developed by Ajwani et al.

The body of our paper comprises five sections. Section 2 describes the two-way search method, verifies its correctness, and analyzes its running time. Section 3 refines the method to yield an algorithm fast for sparse graphs. Section 4 describes an alternative approach that avoids graph search and yields an algorithm fast for dense graphs. Section 5 analyzes the running time of this algorithm. Section 6 examines lower bounds and other issues. In particular, we show in this section that on sparse graphs our $O(m^{1/2})$ amortized time bound is tight to within a constant factor for a natural class of algorithms that includes all the existing ones.

2 Topological Ordering Via Two-Way Search

We develop our topological ordering algorithm through refinement of a general method that encompasses many of the older algorithms. By vertex order we mean the current topological order. We maintain the vertex order using a data structure such that testing the predicate "$x < y$" for two vertices x and y takes $O(1)$ time, as does deleting a vertex from the order and reinserting it just before or just after another vertex. The dynamic ordered list implementations of Dietz and Sleator [6] and Bender et al. [4] meet these requirements: their methods take $O(1)$ time worst-case for an order query or a deletion, and $O(1)$ time for an insertion, amortized or worst-case depending on the structure. For us an amortized bound suffices. In addition to a topological order, we maintain the outgoing and incoming arcs of each vertex. This allows two-way search. Initially the vertex order is arbitrary and all sets of outgoing and incoming arcs are empty.

To add an arc (v, w) to the graph, proceed as follows. Add (v, w) to the set of arcs out of v and to the set of arcs into w. If $v > w$, search forward from w and backward from v until finding either a cycle or a set of vertices whose reordering will restore topological order.

A vertex x is *forward* if there is a path from w to x of arcs traversed forward, *backward* if there is a path from x to v of arcs traversed backward. A vertex is *scanned* if it is forward and all its outgoing arcs have been traversed, or it is backward and all its incoming arcs have been traversed. To do the search, traverse arcs forward from forward vertices and backward from backward vertices until either a forward traversal reaches a backward vertex x or a backward traversal reaches a forward vertex x, in which case there is a cycle, or until there is a vertex s such that all forward vertices less than s and all backward vertices greater than s are scanned.

In the former case, stop and report a cycle consisting of a path from w to x traversed forward, followed by a path from x to v traversed backward, followed by (v, w). In the latter case, restore topological order as follows. Let X be the set of forward vertices less than s and Y the set of backward vertices greater than s. Find topological orders $\vec{X}$ and $\vec{Y}$ of the subgraphs induced by X and Y, respectively. Assume s is not forward; the case of s not backward is symmetric. Delete the vertices in $X \cup Y$ from the current vertex order and reinsert them just after s, in order $\vec{Y}$ followed by $\vec{X}$. (See Figure 1.)

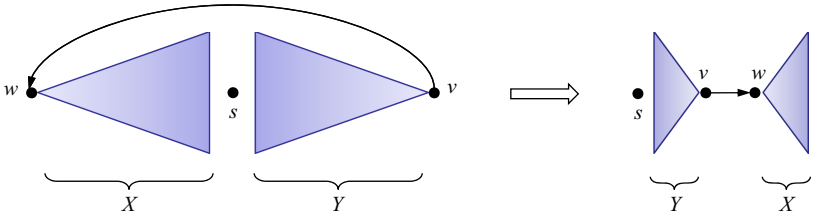


Fig. 1. Restoring topological order after two-way search. Since s is not forward, the vertices in Y are inserted (in topological order) just after s, followed by the vertices in X.

Theorem 1. *The two-way search method is correct.*

Proof. Omitted. See [7]. □

This method requires $O(1)$ time plus $O(1)$ time per arc traversed, plus any overhead needed to guide the search. To obtain a good time bound we need to minimize both the number of arcs traversed and the search overhead. In our discussion we shall assume that no cycle is created. Only one arc addition, the last one, can create a cycle; the last search takes $O(m)$ time plus overhead.

We need a way to charge the search time against graph changes caused by arc additions. To measure such changes, we count pairs of related graph elements, either vertex pairs, vertex-arc pairs, or arc-arc pairs: two such elements are *related* if they are on a common path. The number of related pairs is initially zero and never decreases. There are at most $\binom{n}{2} < n^2/2$ vertex-vertex pairs, nm vertex-arc pairs, and $\binom{m}{2} < m^2/2$ arc-arc pairs. For sparse graphs, the related arc-arc pairs are of most use to us.

We limit the search in three ways to make it more efficient. First, we restrict it to the *affected region*, the set of vertices between w and v. Specifically, only arcs (u, x) with $u < v$ are traversed forward, and only arcs (y, z) with $z > w$ are traversed backward. This suffices to attain an $O(n)$ amortized time bound per arc addition. The bound comes from a count of newly-related vertex-arc pairs: each arc (u, x) traversed forward is newly related to v, each arc (y, z) traversed backward is newly related to w. The algorithm of Marchetti-Spaccamela et al. [18] is the special case that does just a one-way search forward from w using $s = v$, with one refinement and one difference: it does a depth-first search, and it maintains the topological order as an explicit mapping between the vertices and the integers from 1 to n.

One-way search allows a more space-efficient graph representation, since we need only forward incidence sets, not backward ones. But two-way search has a better time bound if it is suitably limited. We make the search *balanced*: each traversal step is of two arcs concurrently, one forward and one backward. There are other balancing strategies [2,11,12], but this simple one suffices for us. Balancing by itself does not improve the time bound; we need a third restriction. We call an arc (u, x) traversed forward and an arc (y, z) traversed backward *compatible* if $u < z$. Compatibility implies that (u, x) and (y, z) are newly related. We make the search *compatible*: each traversal step is of two compatible arcs; the search stops when there is no pair of untraversed compatible arcs.

Theorem 2. *Compatible search is correct.*

Proof. Omitted. □

Lemma 1. *If the searches are compatible, the amortized number of arcs traversed during searches is $O(m^{1/2})$ per arc addition.*

Proof. We count related arc-arc pairs. Consider a compatible search of $2k$ arc traversals, k forward and k backward. Order the arcs (u, x) traversed forward in increasing order on u, breaking ties arbitrarily. Let (u, x) be the $\lceil k/2 \rceil^{th}$ arc in the order. Arc (u, x) and each arc following (u, x) has a compatible arc (y, z) traversed backward. Compatibility and the ordering of forward traversed arcs imply that $u < z$. Thus each such arc (y, z) is newly related to (u, x) and to each arc preceding (u, x), for a total of at least $(k/2)^2$ newly related pairs.

We divide searches into two kinds: those that traverse at most $m^{1/2}$ arcs and those that traverse more. Searches of the first kind satisfy the bound of the lemma. Let $2k_i$ be the number of arcs traversed during the i^{th} search of the second kind. Since $2k_i > m^{1/2}$ and $\sum_i (k_i/2)^2 < m^2/2$, $\sum_i k_i < 2\sum_i {k_i}^2/m^{1/2} = 8\sum_i (k_i/2)^2/m^{1/2} < 4m^{3/2}$. Thus there are $O(m^{1/2})$ arc traversals per arc addition. □

We still need a way to implement compatible search. The most straightforward is to make the search ordered: traverse arcs (u, x) forward in non-decreasing order on u and arcs (y, z) backward in non-increasing order on z. This requires two heaps (priority queues) to store unscanned forward and unscanned backward vertices. In essence this is the algorithm of Alpern et al. [2], although they use a different balancing strategy. The heap overhead is $O(\log n)$ per arc traversal, resulting in an amortized time bound of $O(m^{1/2} \log n)$ per arc addition. More-complicated balancing strategies lead to the improvements [11,12,15] in this bound for non-dense graphs mentioned in Section 1.

3 Compatible Search Via a Soft Threshold

The running time of an ordered search can be reduced further, even for sparse graphs, by using a faster heap implementation, such as those of van Emde Boas [31,30], Thorup [29], and Han and Thorup [8]. But we can do even better, avoiding the use of heaps entirely, by exploiting the flexibility of compatible search. We guess the value of the threshold s and traverse arcs forward only from vertices less than or equal to s and backward only from vertices greater than or equal to s. When we run out of unscanned vertices on one side or the other, we revise our guess of s. Since this method does not know the final value of s until it stops, it can do extra work, but with a careful choice of s this extra work only costs a constant factor.

In addition to the soft threshold s, the algorithm maintains two hard thresholds, l and h, such that all forward vertices less than l and all backward vertices greater than h are scanned. It maintains the invariant $l \leq s \leq h$. It also maintains partitions of the candidate forward vertices into $A \cup B$ and of the candidate backward vertices into $C \cup D$. Set B contains forward vertices temporarily bypassed because they are greater than s (but less than h); set D contains backward vertices temporarily bypassed because

they are less than s (but greater than l). We call the vertices in $B \cup D$ *far*. The remaining candidate vertices, those in $A \cup C$, are *near*. The algorithm is as follows. Initialize l to w, h to v, s to v (or w), A to $\{w\}$, C to $\{v\}$, and B, D, X, and Y to empty. Then repeat an applicable one of the following cases until $A \cup B$ or $C \cup D$ is empty (see Figure 2):

Case 1f. $A = \emptyset$: set $l = s$, $A = B$, $B = D = \emptyset$, and choose $s \in A$.
Case 1b. $C = \emptyset$: set $h = s$, $C = D$, $B = D = \emptyset$, and choose $s \in C$.

In the remaining cases, A and C are non-empty. Choose $u \in A$ and $z \in C$.

Case 2f. $u > s$: delete u from A; if $u < h$, insert u in B.
Case 2b. $z < s$: delete z from C; if $z > l$, insert z in D.

In the remaining cases $l \leq u \leq s \leq z \leq h$.

Case 3f. All arcs from u are traversed: move u from A to X.
Case 3b. All arcs to z are traversed: move z from C to Y.
Case 4. There are untraversed arcs (u, x) and (y, z): choose two such arcs and traverse them. If x is backward or y is forward, stop and report a cycle. If x is unreached and less than h, make it forward and add it to A. If y is unreached and greater than l, make it backward and add it to C.

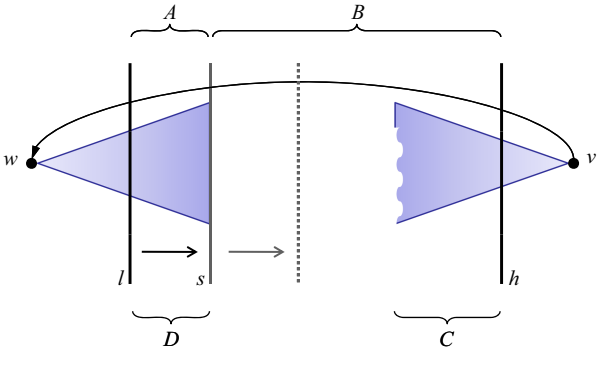


Fig. 2. Compatible search via a soft threshold. In this example A is empty, so Case 1f applies: l is moved to s and a new s is selected from the vertices in B (which becomes the new A).

If the search stops without detecting a cycle, set $s = h$ if $A \cup B$ is empty, $s = l$ otherwise. Delete from X all forward vertices no less than s and from Y all backward vertices no greater than s. Reorder the vertices as described in Section 2.

Theorem 3. *Compatible search with a soft threshold is correct.*

Proof. Omitted. See [7]. □

Lemma 2. *The running time of compatible search via a soft threshold is* $O(1)$ *plus* $O(1)$ *per arc traversed plus* $O(1)$ *for each time a vertex becomes near.*

Proof. Each case either traverses two arcs and adds at most two vertices to $A \cup C$, or permanently deletes a vertex from $A \cup C$, or moves a vertex from $A \cup C$ to $B \cup D$, or moves one or more vertices from $B \cup D$ to $A \cup C$. The number of times vertices are moved from $A \cup C$ to $B \cup D$ is at most the number of times vertices become near. □

The algorithm is correct for any choice of soft threshold, but only a careful choice makes it efficient. Repeated cycling of vertices between near and far is the remaining inefficiency. We choose the soft threshold to limit such cycling no matter how the search proceeds. A good deterministic choice is to let the soft threshold be the median or an approximate median of the appropriate set (A or C); an *ϵ-approximate median* of a totally ordered set of g elements is any element that is less than or equal to at least ϵg elements and greater than or equal to at least ϵg elements, for some constant $\epsilon > 0$. The (exact) median is a $1/2$-approximate median. Finding the median or an approximate median takes $O(g)$ time [5, 26]. An alternative is to choose the soft threshold uniformly at random from the appropriate set. This gives a very simple yet efficient randomized algorithm.

Lemma 3. *If each soft threshold is an ϵ-approximate median of the set from which it is chosen, then the number of times a vertex becomes near is $O(1)$ plus $O(1)$ per arc traversed. If each soft threshold is chosen uniformly at random, then the expected number of times a vertex becomes near is $O(1)$ plus $O(1)$ per arc traversed.*

Proof. The value of l never decreases as the algorithm proceeds; the value of h never increases. Let k be the number of arcs traversed. Suppose each soft threshold is an ϵ-approximate median. The first time a vertex is reached, it becomes near. Each subsequent time it becomes near, it is one of a set of g vertices that become near, as a result of being moved from B to A or from D to C. The two cases are symmetric; consider the former. No matter what happens later, at least ϵg vertices have become near for the last time. Just after s is changed, at least ϵg vertices in A are no less than s, and at least ϵg vertices in A are no greater than s. Just before the next time s changes, $l = s$ or $h = s$. In the former case, all vertices no greater than s can never again become near; in the latter case, all vertices no less than s can never again become near. We charge the group of g newly near vertices to the vertices that become near for the last time. The total number of times vertices can become near is at most $(2 + k)/\epsilon$: there are at most $2 + k$ forward and backward vertices and at most $1/\epsilon$ times a vertex can become near per forward or backward vertex.

Essentially the same argument applies if the soft threshold is chosen uniformly at random. If a set of g vertices becomes near, the expected number that become near for the last time is at least $\sum_{1 \le i \le g/2}(2i)/g = (g/2 + 1)/2 > g/4$ if g is even, at least $\lceil g/2 \rceil + \sum_{1 \le i < \lfloor g/2 \rfloor}(2i)/g = \lceil g/2 \rceil^2/g > g/4$ if g is odd. The total expected number of times vertices can become near is at most $4(2 + k)$. □

Theorem 4. *The amortized time for incremental topological ordering via compatible search is $O(m^{1/2})$ per arc addition, worst-case if each soft threshold is an ϵ-approximate median of the set from which it is chosen, expected if each soft threshold is chosen uniformly at random.*

Proof. Immediate from Lemmas 1–3. □

4 The Dense Case: Topological Search

The compatible search method described in Section 3 is efficient for sparse graphs. In dense graphs, graph search becomes wasteful because it can do unnecessary arc traversals, in particular of arcs that end at vertices beyond the stopping threshold. One way to reduce the overhead of graph search is to sort the incident arc lists by end vertex. Unfortunately, keeping the arc lists sorted seems to require more than $O(m^{3/2})$ time, giving no actual improvement. The $O(n^{2.75})$-time algorithm of Ajwani et al. uses this idea but keeps the arc lists partially sorted, trading off search time against arc list reordering time.

We do better for dense graphs by avoiding graph search and instead searching the topological order. We change the representations of both the graph and the vertex order. For the former we use a matrix $M : M(v, w) = 1$ if (v, w) is an arc, 0 otherwise. For the latter we use an explicit 1-1 mapping $I : V \to \{1, \ldots, n\}$; we also maintain I^{-1}. For a given new arc (v, w) with $I(v) > I(w)$, we visit the vertices in topological order forward from w and backward from v, accessing consecutive entries of I^{-1} until finding either a cycle or a set of vertices whose reordering will restore topological order. This is a form of ordered two-way search in which the ordering comes for free because we search the order, not the graph. The difficulty lies in finding the set of vertices that need to be reordered, because we can no longer depend on arc traversals to find paths.

We explain our algorithm in detail now. We maintain the set of forward vertices F, or those that can be reached from w, and backward vertices R, or those that can reach v, as deques [13] (double-ended queues). To add an arc (v, w) with $I(w) = i$ and $I(v) = j$, proceed as follows. Set $M(v, w) = 1$. If $v > w$, initialize F to $\{w\}$ and R to $\{v\}$ and do an ordered two-way search forward from i and backward from j in $I^{-1}[i..j]$. For each index k visited by the forward search, determine if the vertex $x = I^{-1}[k]$ is forward by querying $M(f, x)$ for each $f \in F$. If any of the queries evaluates to 1, add x to the back of F. The backward search is symmetric. Stop the search when an index t is reached such that all vertices in $I^{-1}[i..t]$ have been visited by the forward search and all vertices in $I^{-1}[t..j]$ have been visited by the backward search. (See Figure 3.)

As in our sparse algorithm, we limit the search in several ways to make it more efficient. First, we restrict it to the affected region $I^{-1}[i..j]$. We also make the search balanced: we alternate searching forward and backward to balance the size of F and R. The forward search runs until it adds a vertex to F, then the backward search runs until it adds a vertex to R, and so on.

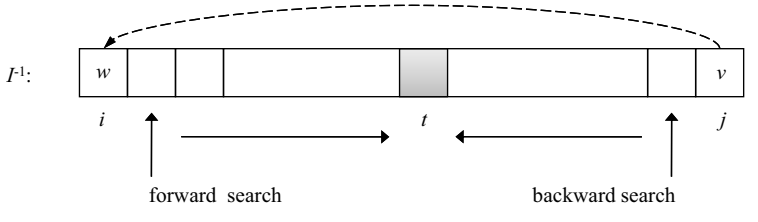


Fig. 3. The forward and backward search meet at index t

At the end of the search phase, we can determine if (v, w) creates a cycle using the following lemma:

Lemma 4. *The new arc (v, w) creates a cycle if and only if $I^{-1}[t] \in F \cap R$ or there is an arc (x, y) such that $x \in F$ and $y \in R$.*

Proof. Omitted. See [12]. □

We still need to reorder the vertices in F and R to restore topological order. We can use the method described in Section 2, but this breaks the 1-1 mapping between the vertices and $\{1, \ldots, n\}$. We take a much simpler approach: we reuse the indices of the vertices in $F \cup R$ (and possibly some other indices we have not seen yet) to reorder the vertices. Begin by deleting all the vertices of $F \cup R$ from their current locations in I^{-1}. The vertices in F must find new indices in $I^{-1}[t..j]$ and the vertices in R must find new indices in $I^{-1}[i..(t-1)]$. Extend the forward search to $I^{-1}[t..j]$; the backward search is extended symmetrically to $I^{-1}[i..(t-1)]$. For each index $k \in \{t, \ldots, j\}$, delete the vertex at the front of F and place it in $I^{-1}[k]$ if $I^{-1}[k]$ is empty. Otherwise if the vertex $x = I^{-1}[k]$ is forward, remove x from its current location and add it to the end of F and place the vertex at the front of F into $I^{-1}[k]$.

At the beginning of the reordering phase the size of F and R are equal to within 1 because they are balanced during the search phase. Consider the forward search; the backward search is symmetric. If the forward search started first during the search phase, then F has at most 1 more vertex than R, the last vertex being at index t itself. It follows that the number of empty indices in $I^{-1}[t..j]$ equals the size of F after the vertices in $F \cup R$ are deleted from I^{-1}. Each time a forward vertex is found during the reordering phase it is added to F and its location in I^{-1} is filled by an existing vertex of F. Each empty index is filled by a vertex from F. Thus the number of empty indices in $I^{-1}[k..j]$ for $k \geq t$ always equals the size of F when the forward search is at k. When $k = j$ there are no empty indices left and all vertices in F have been placed in $I^{-1}[t..j]$.

It is easy to see that the reordering scheme above preserves the relative order of the vertices in F and the relative order of the vertices in R. The reordering is effectively a cyclic permutation of the forward and backward vertices in the affected region. Vertices that are not forward or backward, and all vertices outside the affected region, are unaffected by the algorithm. Combining the above argument with Lemma 4 and Theorem 1, we have:

Theorem 5. *Topological search is correct.*

5 Bounding the Running Time

To bound the running time of the algorithm, observe that its work comes in two forms: searching the topological order between i and j and maintaining the sets F and R, and checking for a cycle prior to the reordering phase. Since the two-way search is ordered, the latter time can be bounded using a count of related vertex pairs (all pairs in $F \times R$ prior to the reordering phase are newly related).

Lemma 5. *The time required to check for cycles over all arc additions is $O(n^2)$.*

It is instructive to observe that balanced search is not required for Lemma 5. Balanced search is used to draw an equivalence between the total movement of vertices during reordering and the work performed by the algorithm to maintain the sets F and R. Let I_e be the topological order before adding arc $e = (v, w)$ and let I'_e be the topological order after adding e. Balanced search allows us to show the following:

Lemma 6. *The time required to maintain F and R for a new arc e is $\sum_{x \in V} |I_e(x) - I'_e(x)|$.*

Proof. Consider the forward search; the backward search is symmetric. Any vertex $x \in F$ belongs to the set F while the forward search moves from $I_e(x)$ to $I'_e(x)$ in I^{-1}. During this time, each non-empty index k causes the algorithm to check if the vertex $I^{-1}[k]$ is adjacent to x. Since x is involved in only one such query per index, the total work attributable to x is $|I_e(x) - I'_e(x)|$. To see why x does not do more than this amount of work, it suffices to observe that F and R are balanced prior to reordering and that $I'_e(x)$ must lie in $\{t, \ldots, j\}$.

Now, we use a result from [1] to bound the total movement of vertices, which in turn bounds the total running time of our algorithm.

Lemma 7. *$\sum_{x \in V} |I_e(x) - I'_e(x)|$ over all arcs e is $O(n^{2.5})$.*

Proof. The reordering phase of our algorithm performs a cyclic permutation of the vertices inserted into $F \cup R$. We can view this permutation as a sequence of swaps between pairs of vertices (x, y) where $x \in R$ and $y \in F$. Prior to a swap $I(x) > I(y)$. If a pair (x, y) is swapped during a permutation, then it is never swapped again in any future permutation because x and y are newly related by the path created from x to v followed by the arc (v, w) followed by the path from w to y. Let $d(x, y) = I(x) - I(y)$, the difference between x and y when they are swapped as a result of a permutation. By the previous argument $d(x, y)$ is uniquely defined. Ajwani et al. [1] used a linear program to show that $\sum d(x, y) = O(n^{2.5})$, where the summation is over all pairs (x, y) that get swapped during some permutation. Thus it suffices to decompose a permutation into a sequence of swaps to prove this lemma.

Let F_e be the set of vertices added to F as a result of processing arc e; define R_e analogously. (F_e and R_e contain vertices inserted during both the search and reordering phases of the algorithm.) Let $F_e = \{w_0, w_1, \ldots, w_p\}$, where $w_0 = w$ and $O(w_0) < \ldots < O(w_p)$, and let $R_e = \{v_0, v_1, \ldots, v_q\}$, where $v_0 = v$ and $O(v_0) > \ldots > O(v_q)$. We decompose the permutation of $F_e \cup R_e$ as follows. For each vertex $x \in R_e$ starting with v_q and in increasing order, swap x with the vertices in F_e in decreasing order, starting with the first vertex $w_r \in F_e$ that is less than x. That is, swap x successively with the vertices in $\{w_r, \ldots, w_0\}$. After the swaps are complete the new index of x is $I'_e(x)$ and all vertices in F_e are higher than x, so we have:

$$I_e(x) - I'_e(x) = \sum_{y \in \{w_r, \ldots, w_0\}} d(x, y) \tag{1}$$

Repeat this process for the next vertex (v_{q-1}) and so on until all vertices in R_e are less than all vertices in F_e. Since the swaps only use existing indices of $F_e \cup R_e$ in I^{-1}

and since the relative order of the vertices in each set is preserved, it follows that the final order of the vertices is: $v_q, v_{q-1}, \ldots, v_0, w_0, \ldots, w_{p-1}, w_p$. We can bound the total movement of the vertices as follows. Let π_e be the permutation due to arc e; $(x, y) \in \pi_e$ means that the pair (x, y) is swapped in π_e.

$$\sum_{x \in V} |I_e(x) - I'_e(x)| = \sum_{x \in R_e} (I_e(x) - I'_e(x)) + \sum_{y \in F_e} (I'_e(y) - I_e(y)) \tag{2}$$

$$= 2 \sum_{x \in R_e} (I_e(x) - I'_e(x)) \tag{3}$$

$$= 2 \sum_{x \in R_e} \sum_{y:(x,y) \in \pi_e} d(x, y) \tag{4}$$

$$= 2 \sum_{(x,y) \in \pi_e} d(x, y).$$

Equation 3 follows from 2 because $\sum_{x \in V} I_e(x) = \sum_{x \in V} I'_e(x)$. Equation 4 follows from Equation 1. Since each pair (x, y) is swapped at most once over all permutations, the above sum is identical to the result of Ajwani et al. within a factor of 2 and we have $\sum_{x \in V} |I_e(x) - I'_e(x)| = 2 \sum_{(x,y) \in \pi_e} d(x, y) = O(n^{2.5})$. □

6 Lower Bounds and Other Issues

A natural question to ask is whether the time bounds we have obtained for our algorithms are tight, and more generally whether there are faster algorithms for either sparse or dense graphs or both. For the sparse case, Katriel and Bodlaender [11] give a class of examples on which our soft threshold algorithm takes $\Omega(m^{1/2})$ time per arc addition; thus our analysis is tight. For the dense case, our topological search algorithm takes $\Omega(n^2/m)$ time per arc addition in the worst case by a general lower bound below. We do not know whether our bound of $O(n^{2.5}/m)$ for this algorithm is tight, although the solution of the LP used in the analysis is $\Theta(n^{2.5})$: there may be additional constraints in the behavior of the algorithm that are not captured by the LP.

More generally, Ramalingam and Reps [23] gave a class of examples in which $n - 1$ arc additions force $\Omega(n \log n)$ vertex reorderings, no matter what topological order is maintained. Katriel [10] gave a class of examples on which any algorithm that (1) only reorders vertices within the affected region and (2) maintains the vertex order as an explicit mapping from the vertices to $\{1, \ldots, n\}$ must do $\Omega(n^2)$ vertex reorderings for n arc additions. Our topological search algorithm is subject to this bound, although our soft threshold algorithm is not. We have obtained the following related result:

Theorem 6. *There is a class of examples on which any algorithm that only reorders vertices within the affected region must do $\Omega(nm^{1/2})$ vertex reorderings for n arc additions.*

Proof. Omitted. See [7]. □

All existing algorithms are subject to this bound. The theorem implies that our soft threshold algorithm is within a constant factor of minimum-time on sparse graphs ($m = O(n)$) among algorithms that reorder only within the affected region.

If both additions and deletions are allowed, there is no known solution for either the topological ordering or cycle detection problem better than running an $O(m)$-time static algorithm after each graph change. There has been quite a bit of work on the harder problem of maintaining full reachability information for a dynamic graph. See [24,25].

We have used amortized running time as our measure of efficiency. An alternative way to measure efficiency is to use an incremental competitive model [22], in which the time spent to handle an arc addition is compared against the minimum work that must be done by any algorithm, given the same current topological order and the same arc addition. The minimum work that must be done is the minimum number of vertices that must be reordered, which is the measure that Ramalingam and Reps used in their lower bound. But no existing algorithm handles an arc addition in time polynomial in the minimum number of vertices that must be reordered. To obtain positive results, some researchers have measured the performance of their algorithms against the minimum sum of degrees of vertices that must be reordered [2] or a more-refined measure that counts out-degrees of forward vertices and in-degrees of backward vertices [20]. For these models, appropriately balanced forms of ordered search are competitive to within a logarithmic factor [2,20]. In such a model, our sparse-efficient algorithm is competitive to within a constant factor.

Alpern et al. and Pearce and Kelly consider batched arc additions as well as single arc additions. Generalizing compatible search and topological search to efficiently handle batched arc additions is a topic for future work.

Acknowledgement

The last author thanks Deepak Ajwani for his presentation at the 2007 Data Structures Workshop at Bertinoro that motivated the work in Sections 2 and 3.

References

1. Ajwani, D., Friedrich, T., Meyer, U.: An $O(n^{2.75})$ algorithm for online topological ordering. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 53–64. Springer, Heidelberg (2006)
2. Alpern, B., Hoover, R., Rosen, B.K., Sweeney, P.F., Zadeck, F.K.: Incremental evaluation of computational circuits. In: SODA 1990, pp. 32–42 (1990)
3. Belik, F.: An efficient deadlock avoidance technique. IEEE Trans. on Comput. 39(7) (1990)
4. 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)
5. Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. J. of Comput. and Syst. Sci. 7(4), 448–461 (1973)
6. Dietz, P.F., Sleator, D.D.: Two algorithms for maintaining order in a list. In: STOC 1987, pp. 365–372 (1987)
7. Haeupler, B., Sen, S., Tarjan, R.E.: Incremental topological ordering and strong component maintenance (2008)
8. Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: FOCS 2002, pp. 135–144 (2002)

9. Harary, F., Norman, R.Z., Cartwright, D.: Structural Models : An Introduction to the Theory of Directed Graphs. John Wiley & Sons, Chichester (1965)
10. Katriel, I.: On algorithms for online topological ordering and sorting. Technical Report MPI-I-2004-1-003, Max-Planck-Institut für Informatik, Saarbrücken, Germany (2004)
11. Katriel, I., Bodlaender, H.L.: Online topological ordering. ACM Trans. on Algor. 2(3), 364–379 (2006)
12. Kavitha, T., Mathew, R.: Faster algorithms for online topological ordering (2007)
13. Knuth, D.E.: The Art of Computer Programming. Fundamental Algorithms, vol. 1. Addison-Wesley, Reading (1973)
14. Knuth, D.E., Szwarcfiter, J.L.: A structured program to generate all topological sorting arrangements. Inf. Proc. Lett. 2(6), 153–157 (1974)
15. Liu, H.-F., Chao, K.-M.: A tight analysis of the Katriel-Bodlaender algorithm for online topological ordering. Theor. Comput. Sci. 389(1-2), 182–189 (2007)
16. Liu, H.-F., Chao, K.-M.: An $\tilde{O}(n^{2.5})$-time algorithm for online topological ordering (2008)
17. Marchetti-Spaccamela, A., Nanni, U., Rohnert, H.: On-line graph algorithms for incremental compilation. In: van Leeuwen, J. (ed.) WG 1993. LNCS, vol. 790, pp. 70–86. Springer, Heidelberg (1994)
18. Marchetti-Spaccamela, A., Nanni, U., Rohnert, H.: Maintaining a topological order under edge insertions. Inf. Proc. Lett. 59(1), 53–58 (1996)
19. Omohundro, S.M., Lim, C.-C., Bilmes, J.: The Sather language compiler/debugger implementation. Technical Report TR-92-017, International Computer Science Institute, Berkeley (1992)
20. Pearce, D.J., Kelly, P.H.J.: A dynamic topological sort algorithm for directed acyclic graphs. J. of Exp. Algorithmics 11, 1–7 (2006)
21. Pearce, D.J., Kelly, P.H.J., Hankin, C.: Online cycle detection and difference propagation for pointer analysis. In: SCAM 2003, pp. 3–12 (2003)
22. Ramalingam, G., Reps, T.W.: On the computational complexity of incremental algorithms. Technical Report CS-TR-1991-1033, University of Wisconsin-Madison (1991)
23. Ramalingam, G., Reps, T.W.: On competitive on-line algorithms for the dynamic priority-ordering problem. Inf. Proc. Lett. 51(3), 155–161 (1994)
24. Roditty, L., Zwick, U.: Improved dynamic reachability algorithms for directed graphs. In: FOCS 2002, pp. 679–688 (2002)
25. Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: STOC 2004, pp. 184–191 (2004)
26. Schönhage, A., Paterson, M., Pippenger, N.: Finding the median. J. of Comput. and Syst. Sci. 13(2), 184–199 (1976)
27. Shmueli, O.: Dynamic cycle detection. Information Processing Letters 17(4), 185–188 (1983)
28. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. on Comput. 1(2), 146–160 (1972)
29. Thorup, M.: Integer priority queues with decrease key in constant time and the single source shortest paths problem. J. of Comput. Syst. Sci. 69(3), 330–353 (2004)
30. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. Inf. Proc. Lett. 6(3), 80–82 (1977)
31. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Mathematical Systems Theory 10, 99–127 (1977)

Dynamic Normal Forms and Dynamic Characteristic Polynomial

Gudmund Skovbjerg Frandsen[1] and Piotr Sankowski[2]

[1] University of Aarhus, Denmark
gudmund@daimi.au.dk
[2] Warsaw University, Poland and University of Rome "La Sapienza", Italy
piotr.sankowski@gmail.com

Abstract. We present the first fully dynamic algorithm for computing the characteristic polynomial of a matrix. In the generic symmetric case our algorithm supports rank-one updates in $O(n^2 \log n)$ randomized time and queries in constant time, whereas in the general case the algorithm works in $O(n^2 k \log n)$ randomized time, where k is the number of invariant factors of the matrix. The algorithm is based on the first dynamic algorithm for computing normal forms of a matrix such as the Frobenius normal form or the tridiagonal symmetric form. The algorithm can be extended to solve the matrix eigenproblem with relative error 2^{-b} in additional $O(n \log^2 n \log b)$ time. Furthermore, it can be used to dynamically maintain the singular value decomposition (SVD) of a generic matrix. Together with the algorithm the hardness of the problem is studied. For the symmetric case we present an $\Omega(n^2)$ lower bound for rank-one updates and an $\Omega(n)$ lower bound for element updates.

Introduction. The computation of the *characteristic polynomial* (CP) of a matrix and the eigenproblem are two important problems in linear algebra and they find an enormous number of applications in mathematics, physics and computer science. Till now almost nothing about the dynamic complexity of these problems has been known. The CP problem is essentially equivalent to the computation of the Frobenius Normal Form (FNF), known also as rational canonical form. All of the efficient algorithms for CP are based on the FNF computation [1–4]. The fastest static algorithms for computing CP are either based on fast matrix multiplication and work in $\tilde{O}(n^\omega)$ time [4, 1] or they are so called black-box approaches working in $\tilde{O}(nm)$ time [2, 3], where m is the number of nonzero entries in the matrix. The latter bound holds only in the generic case, whereas the fastest general algorithm works in $O(\mu nm)$ [3], where μ is the number of distinct invariant factors of the matrix. All of these results have been obtained very recently. In this paper we are trying to understand the dynamic complexity of these fundamental problems by devising efficient algorithms and by proving matching lower bounds. Note, that in this paper and in all of the papers cited above, we study the arithmetic complexity of the problem, i.e., the notion of time is equivalent to the count of arithmetic operations and discrete control operations. More strictly speaking we work in the real RAM model, for details see [5].

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 434–446, 2008.

In the first part of the paper we consider the problem of computing the normal form of a real (complex) $n \times n$ dynamic matrix A. We assume that the matrix can be changed with use of rank-one updates, i.e., for two n dimensional vectors a and b we allow updates of the form $A := A + ab^T$. We want to dynamically compute matrices Q and F such that $A = Q^{-1}FQ$, where Q is the unitary similarity transformation and F is the normal form in question. The algorithm should support queries to F as well as vector queries to Q, i.e., given vector v it should be able to return Qv or $Q^{-1}v$. We present the following fully dynamic randomized algorithms for this problem:

- for generic symmetric matrices — an algorithm for tridiagonal normal form supporting updates in $O(n^2 \log n)$ worst-case time,
- for general matrices — an algorithm for Frobenius normal form (for definition see Section 1.1) supporting updates in $O(kn^2 \log n)$ worst-case time, where k is the number of invariant factors of the matrix.

The queries for F are answered in $O(1)$ time and the queries to Q in $O(n^2 \log n)$ worst-case time. After each update the algorithms can compute the characteristic polynomial explicitly and hence support queries for CP in constant time. These are the first known fully dynamic algorithms for the CP and normal form problems. Our results are based on a general result which can be applied to any normal form, under condition of availability of a static algorithm for computing the normal form of a sparse matrix. For the completeness of the presentation we have included the full algorithm for generic symmetric matrices, whereas the included algorithm for Frobenius normal form is the most universal result.

This algorithm can be extended to solve the dynamic eigenproblem, i.e., we are asked to maintain with relative error 2^{-b} the eigenvalues $\lambda_1, \ldots, \lambda_n$ and a matrix Q composed of eigenvectors. In generic case, our algorithm supports updates in $O(n \log^2 n \log b + n^2 \log n)$ worst-case time, queries to λ_i in constant time and vector queries to Q in $O(n^2 \log n)$ worst-case time. You should note that the relative error 2^{-b} is immanent even in the exact arithmetic model, i.e., the eigenvalues can only be computed approximately (for more details please see [6]).

Let A be a real (complex) $m \times n$ matrix, $m \geq n$. The singular value decomposition (SVD) for A consists in two orthogonal (unitary) matrices U and V and a diagonal matrix $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$ with nonnegative real entries (the singular values) such that $A = U\Sigma V^T$. Usually the entries of Σ are sorted $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ and in that case Σ is unique. We define Σ_k to be $\mathrm{diag}(\sigma_1, \ldots, \sigma_k, 0, \ldots, 0)$. The dynamic SVD problem considers maintaining the SVD under rank one updates and 2 query operations, one operation returns elements of Σ, another returns the k-rank approximation to A, i.e., given r and v return $U\Sigma_r V^T v$. Here, again the results are with a relative error 2^{-b}. Our algorithm for SVD supports updates in $O(n \log^2 n \log b + n^2 \log n)$ worst-case time in the generic case, queries to Σ in constant time and r-rank approximation query in $O(n^2 \log n)$ worst-case time.

Accompanying the above upper bounds, we provide some lower bounds for the problem of computing the characteristic polynomial. The lower bounds are formulated in the model of history dependent algebraic computation trees [7].

One should note that our algorithms for computing the CP fit into this model. We use the technique developed and used by [7] for proving $\Omega(n)$ lower bounds for several dynamic matrix problems. The technique has been used later to prove an $\Omega(n)$ lower bound for the matrix rank problem [8]. Here, we significantly extend the technique to show an $\Omega(n^2)$ lower bound for the problem of computing the characteristic polynomial in the case of column updates. This is the first known result of this type. A column update can be realized with the use of one rank-one update. Additionally, we provide $\Omega(n)$ lower bounds for the CP problem in the case of element updates.

The paper is organized as follows. In the next subsection we motivate our study by reviewing possible applications within the scope of computer science. Nevertheless, note that the eigenproblem is THE method to study physical systems and our result could be applied to speed-up the physical computations in case when system parameters can be changed. In Section 1 we introduce the algorithms mentioned above. Section 2 includes the description of the obtained lower bound.

Applications and Earlier Work. Our result delivers a general framework for solving many problems that are based on the computation of matrix normal forms and on the solution of the matrix eigenproblem. Hence it generalizes a large number of problem specific solutions and can be directly applied to a broad spectrum of problems. Until now, it has been known how to dynamically compute the lowest coefficient of the CP, i.e., the determinant [9] and the rank of the matrix [8]. Our paper generalizes these two results as the CP can be used for both computing the determinant and in a rather simple way for computing the matrix rank [10, 11].

Our algorithms can be used to maintain dynamic information about the spectrum of a graph. Spectral graph theory has a large number of applications (see e.g. [12]) and delivers a way to compute numerous information about the graph. One of the possible applications is a dynamic testing of graph isomorphism, where one of the basic tools is a characteristic polynomial of the adjacency or Laplacian matrix [13].

Our algorithms for computing eigenvalues and eigenvectors can be used for both dynamically approximating the size of the graph partition and for finding good candidates for partition, e.g., the second smallest eigenvalue is related to the minimum partition size [14, 15]. There are several spectral methods for finding partitions and clusters in the graphs which can be used together with our algorithm [16, 17]. Clustering methods find application in image recognition and processing [16], where dynamic algorithms may be useful to process image changes.

Another direct application of our results is the dynamic maintenance of the stationary distribution of the finite Markov chain. For this problem slightly faster algorithms, working in $O(n^2)$ time, based on Sherman-Morison formula has been presented in [18–20]. However, our result is more general and can be used to check if the stationary distribution is unique or to compute the convergence time to stationary distribution by finding second largest eigenvalue [21], etc..

The SVD of a matrix may be directly used for finding the nearest matrix of a rank at most r by zeroing all singular values except the r largest [22]. For such a use only the leftmost r columns of U and V need to be known. Our concrete dynamic algorithm below allows application of the rank r approximation matrix to a vector in time $O(m^2 \log m)$ for any r. Indirectly dynamic SVD has many applications, e.g., for image analysis [23], in databases [24] and for data mining (recommender systems) [25].

In the case when addition of an entire vector or deletion of the last column of the matrix is allowed algorithms that take $O((m+n)\min(m,n))$ time per operation are known [26, 27]. Brand [25] has described an algorithm for rank one updates of the SVD (similar to our model) that takes time $O(mr + r^3)$ for maintaining a rank r approximation. Hence our solution improves these results as well.

1 Dynamic Characteristic Polynomial - Upper Bounds

In this section we show the algorithms for dynamically computing the characteristic polynomial, computing normal form and solving matrix eigenproblem. We show that the CP problem is strongly related to the static problem of computing the CP of a sparse or structured matrix. Standard methods for computing CPs transform the matrix to a normal form from which the characteristic polynomial can be easily computed. If the normal form has $O(n)$ non-zero entries and its CP can be computed in $O(n^2)$ time, it is called a *thin* normal form. For example, the Frobenious or the tridiagonal normal forms are thin. We assume, we are given a static algorithm that computes a thin normal form, a transition matrix and its inverse with use of $O(n)$ matrix-vector multiplications and $O(f(n))$ additional operations. We show how to convert this algorithm into a dynamic algorithm for computing a thin normal form supporting updates in $O(n^2 \log n + f(n))$ amortized time and queries in constant time. This result automatically implies a dynamic algorithm for computing the CP. Next we move to the application of this result and show implementations of this solution in the symmetric and general case. If the algorithm has some additional properties, we can turn it into a dynamic worst-case time algorithm. Finally we show how to extend the result to dynamically solve the matrix eigenproblem and SVD problem.

1.1 Amortized Bound

In the algorithm we keep the $n \times n$ matrix A over the field $\mathcal{F}$ in the following lazy form:

$$A = Q_0^{-1} Q_1^{-1} \dots Q_{k-1}^{-1} Q_k^{-1} T Q_k Q_{k-1} \dots Q_1 Q_0, \tag{1}$$

where $k \leq \lceil \log n \rceil$, the matrices Q_i, for $i = 1, \dots, n$, are some similarity transformations and T is a thin normal form. In each update we recompute the lazy form of the matrix and afterwards we compute its characteristic polynomial with use of the matrix T. We initialize the algorithm with the matrix A_0 and compute its normal form $A_0 = Q_0^{-1} T_0 Q_0$. Moreover, after n updates we reinitialize the

algorithm. Let us consider the sequence of t rank-one updates given by vectors a_i and b_i, for $i = 1, \ldots, t$, and let A_i denote the matrix after the i-th update, i.e.:

$$A_i = A_0 + \sum_{j=1}^{i} a_j b_j^T.$$

Let:

$$t = 2^{j_1} + 2^{j_2} + \ldots + 2^{j_k}, \tag{2}$$

where $j_1 > j_2 > \ldots > j_k$. We require that the lazy form (1) fulfils the following:

$$A_{2^{j_1}+\ldots+2^{j_i}} = Q_0^{-1} Q_1^{-1} \ldots Q_{i-1}^{-1} Q_i^{-1} T_i Q_i Q_{i-1} \ldots Q_1 Q_0, \tag{3}$$

for $i = 1, \ldots, k$ and for T_i in the thin normal form. Now consider a new update numbered $t+1$. We have $t+1 = 2^{j_1} + 2^{j_2} + \ldots + 2^{j_{k'}} + 2^{j'}$, for some $k' \leq k$ and $j' < j_{k'}$. Thus we have to compute a new lazy form fulfilling:

$$A + a_{t+1} b_{t+1}^T = Q_0^{-1} Q_1^{-1} \ldots Q_{k'}^{-1} Q'^{-1}_{k'+1} T Q'_{k'+1} Q_{k'} \ldots Q_1 Q_0. \tag{4}$$

Note that in order to recompute this form we have to discard the matrices $Q_{k'+1}, \ldots, Q_k$ and compute a matrix $Q'_{k'+1}$. Hence applying (3) we have:

$$A + a_{t+1} b_{t+1}^T = A_{2^{j_1}+\ldots+2^{j_{k'}}} + \sum_{j=2^{j_1}+\ldots+2^{j_{k'}}+1}^{2^{j_1}+\ldots+2^{j_{k'}}+2^{j'}} a_j b_j^T =$$

$$= Q_0^{-1} Q_1^{-1} \ldots Q_{k'-1}^{-1} Q_{k'}^{-1} T_{k'} Q_{k'} Q_{k'-1} \ldots Q_1 Q_0 + \sum_{j=2^{j_1}+\ldots+2^{j_{k'}}+1}^{2^{j_1}+\ldots+2^{j_{k'}}+2^{j'}} a_j b_j^T =$$

$$= Q_0^{-1} \ldots Q_{k'}^{-1} \left(T_{k'} + \sum_{j=2^{j_1}+\ldots+2^{j_{k'}}+1}^{2^{j_1}+\ldots+2^{j_{k'}}+2^{j'}} Q_{k'} \ldots Q_0 a_j b_j^T Q_0^{-1} \ldots Q_{k'}^{-1} \right) Q_{k'} \ldots Q_0.$$

Thus we have to compute the normal form of the matrix D_{t+1}:

$$D_{t+1} := T_{k'} + \sum_{j=2^{j_1}+\ldots+2^{j_{k'}}+1}^{2^{j_1}+\ldots+2^{j_{k'}}+2^{j'}} Q_{k'} \ldots Q_0 a_j b_j^T Q_0^{-1} \ldots Q_{k'}^{-1}. \tag{5}$$

Note that the vectors $a_{j,k'} = Q_{k'} \ldots Q_0 a_j$ and $b_{j,k'}^T = b_j^T Q_0^{-1} \ldots Q_{k'}^{-1}$, for $j \leq t$, are computed at the time when the algorithm was recomputing the lazy form after the j-th update. At the time of the j-th update k was greater than k' and so the matrices $Q_0, \ldots, Q_{k'}$ have not changed since then. Thus we only need to compute the vectors $a_{t+1,k'}$ and $b_{t+1,k'}$ when we are performing the $t+1$-th update — this takes $O(n^2 \log n)$ time. The multiplication of a vector by the matrix D_{t+1} takes $O(n 2^{j'})$ time. Hence for the computation of the normal form of D_{t+1} we need $O(n^2 2^{j'} + f(n))$ time.

Let us now compute the total cost of performing n updates:

- for the initialization of the lazy form we need $O(n^3 + f(n))$ time,
- for the computation of vectors $a_{j,i}$ and $b_{j,i}$ for $i = 1, \ldots, k$ we require $O(n \cdot n^2 \log n) = O(n^3 \log n)$ time,
- for the normal forms of D_i we need $O(nf(n) + \sum_{j=1}^{\lceil \log n \rceil} 2^{\lceil \log n \rceil - j} n^2 2^j) = O(n^3 \log n + nf(n))$ time, because we spend $O(n^2 2^j)$ time for computing the normal form $\frac{n}{2^j}$ times, namely every 2^j-th update.

Thus finally we get.

Theorem 1. *If there exists an algorithm for computing a given thin normal form, transition matrix and its inverse by performing $O(n)$ matrix-vector products and $O(f(n))$ additional operations then there exists a dynamic algorithm that maintains the characteristic polynomial and the thin normal form supporting rank one updates in $O(n^2 \log n + f(n))$ amortized time, queries to the normal form in constant time and vector queries to the transition matrix in $O(n^2 \log n)$ time.*

Generic Symmetric Case. Let us move to the implementations of Theorem 1 and let us for the moment assume that the $n \times n$ matrix A remains symmetric during the updates, i.e., we consider only updates of the form $A := A + aa^T$, where a is an arbitrary n dimensional vector. We want to compute a unitary matrix Q and a symmetric tridiagonal matrix T such that $A = QTQ^T$. The following is the result which can be obtained with the use of standard Lanczos method. For the completeness of the presentation the details of the proofs of the following theorems are included in Appendix A[28].

Theorem 2. *There exists an algorithm for computing a tridiagonal form of a symmetric generic matrix (when the characteristic polynomial equals the minimal polynomial) with use of n matrix-vector products and $O(n^2)$ additional operations. The algorithm is randomized and succeeds with high probability.*

Lemma 1. *The symmetric tridiagonal form is thin.*

Combining Theorem 1, Lemma 1 and Theorem 2 we get the $O(n^2 \log n)$ amortized updated time dynamic algorithm for computing the CP and the tridiagonal form of the generic symmetric matrix.

General Case. In the general case we use the following results due to Eberly [2], who showed how the Frobenius normal form of a sparse matrix can be computed. Frobenius normal form F_A of a matrix A is a block diagonal matrix with companion matrices of monic polynomials $f_1, \ldots, f_k$ on the diagonal, where f_i is divisible by f_{i+1}, for $1 \leq i \leq k-1$ and $VAV^{-1} = F_A$. The companion matrix of a monic polynomial $x^d + g_{d-1}x^{d-1} + \ldots + g_1 x + g_0 \in \mathcal{F}[x]$ is a $d \times d$ matrix defined as:

$$C_g = \begin{bmatrix} 0 & \dots & 0 & -g_0 \\ 1 & \dots & 0 & -g_1 \\ & \ddots & \vdots & \vdots \\ 0 & \dots & 1 & -g_{d-1} \end{bmatrix}.$$

The polynomials $f_1, \dots, f_k$ are *the invariant factors* of A and k is the number of invariant factors. We have $\chi_A(\lambda) = \prod_{i=1}^{k} f_k(\lambda)$ and so F_A is thin as it can be used to compute the CP in $O(n^2)$ time.

Theorem 3 (Eberly '00). *There exists an algorithm for computing Frobenius normal form F of the matrix A together with the transition matrix V and its inverse with use of $O(n)$ matrix-vector products and $O(kn^2+n^2 \log^2 n)$ additional operations, where k is the number of invariant factors of A. The algorithm is randomized and may fail with arbitrarily small probability.*

Using the above theorem together with Theorem 1 we obtain the $O(kn^2 \log^2 n)$ amortized updated time dynamic algorithm for computing the CP and the Frobenius form of the matrix.

Remark 1. The time bound in the above theorem can be reduced to $O(kn^2 + n^2 \log n)$ by keeping the inverse of the transition matrix in the lazy form as given in Section 4.4 of [2]. Then the matrix-vector multiplication by the inverse can be carried out in $O(n^2)$ time. We skip the details due to page limitation of this extended abstract. The same holds for all the following results, i.e., with little effort one can always obtain an $O(kn^2 \log n)$ time algorithm instead of the $O(kn^2 \log^2 n)$ time algorithm. In the next theorems we use the $O(kn^2 + n^2 \log n)$ time bound, but postpone the details to the full version of this paper.

1.2 Worst-Case Bound

The algorithm presented in the previous section works in amortized time bound. Here we show how to modify it to work in worst-case time using rebuilding technique. However, as we keep a set of $\log n$ matrices we have to be very careful devising the rebuilding in order to guarantee the same worst-case time. Notice, that the standard technique, so called global rebuilding, in which we use two copies of the structure used alternately for answering queries, does not work here due to the multilevel recomputations. Here, we can only rebuild small parts of the structure, so one may call the used technique *local rebuilding*. Moreover due to non-uniqueness of the standard forms we also have to guarantee that the small recomputed parts remain consistent during the execution of the algorithm. In order to guarantee that the normal forms remain consistent we cannot discard transition matrices, but we have to multiply them. We cannot use standard or even fast matrix multiplication because it is too slow for our purposes. However, we can show that transition matrices can be multiplied faster without using classical matrix multiplication. All the details of the techniques used to prove the following theorem are included in Appendix B[28].

Theorem 4. *There exists an algorithm that:*

- *for the generic matrices maintains tridiagonal normal form and supports updates in $O(n^2 \log n)$ worst-case time,*
- *for the general matrices with k invariant factors maintains Frobenius normal form and supports updates in $O(kn^2 \log n)$ worst-case,*

the queries to CP and the normal form are supported in constant time, whereas vector queries to transition matrix are supported in $O(n^2 \log n)$ time.

1.3 Dynamic Matrix Eigenproblem

Theorem 4 presented in the previous section can be extended to solve the matrix eigenproblem.

Theorem 5. *There exists a dynamic algorithm for the matrix eigenproblem supporting rank one updates:*

- *in $O(n^2 \log n + n \log^2 n \log b)$ worst-case time for symmetric matrices,*
- *in $O(kn^2 \log n + n \log^2 n \log b)$ worst-case time for general matrices with k invariant factors.*

The computations are carried out with relative error 2^{-b}, the queries to the eigenvalues are answered in constant time and vector queries to eigenvector matrix in $O(n^2 \log n)$ worst-case time.

Proof. Note that the eigenvalues and eigenvectors of the maintained tridiagonal or Frobenius normal form F can be computed in $O(n^2 \log n + n \log^2 n \log b)$ time with use of the algorithm given by Pan and Chen [6]. The eigenvalues of F are of course the same as the eigenvalues of the maintained matrix. However, the eigenvectors have to be multiplied by $Q_0^{-1} Q_1^{-1} \dots Q_k^{-1} Q_0$ what takes $O(n^2 \log n)$ time for each eigenvector. □

1.4 Dynamic Singular Value Decomposition

Our algorithm for dynamic SVD is an application of the earlier results for dynamic eigenvalues and eigenvectors of symmetric matrices. The details and the proof of the following theorem are in Appendix C[28].

Theorem 6. *There exists a dynamic algorithm for SVD of a generic matrix supporting rank one updates in $O(n^2 \log n + n \log^2 n \log b)$ worst-case time. The computations are carried out with relative error 2^{-b} and the queries to the singular values are answered in constant time, whereas queries for r-rank approximation are answered in $O(n^2 \log n + nm)$ worst-case time.*

2 Dynamic Characteristic Polynomial - Lower Bounds

Problems considered. Let $s_i(A)$ or simply s_i denote the ith coefficient of the characteristic polynomial of the $n \times n$ matrix A over the field $\mathcal{F}$, i.e.,

$\chi_A(\lambda) = \det(\lambda I - A) = \lambda^n + \sum_{i=1}^{n}(-1)^i s_i \lambda^{n-i}$. We let our basic dynamic algebraic problem D associated with the characteristic polynomial consist in finding an efficient algorithm that after an initial preprocessing of $A = \{a_{ij}\}$ can handle operations $\texttt{change}_{ij}(v)$ that alters a_{ij} to v and operations $\texttt{query}_i$ that returns the current value of $s_i(A)$. To get stronger lower bounds, we also consider the problem D_i where we restrict ourselves to a single $\texttt{query}_i$ that may be automatically appended to all change operations that are thus required to maintain $s_i(A)$. We also consider the very restricted simple problem, D'_i, where we are only required to maintain information about whether $s_i(A)$ is zero or nonzero. All the above problems have variants that consider vector updates, i.e., changing an entire row and/or column of the matrix A instead of changing single entries of A. Similarly, all problems may be restricted to symmetric matrices, so change operations are paired symmetrically.

Model of computation. Our basic model of computation is the history dependent algebraic computation trees from [7]. A standard algebraic computation tree has computation nodes $+, \cdot, -, /$ and branching nodes (zero tests) [29]. For the field of real numbers, all continuous operations including square root used in the Lanczos algorithm, can be supported [5], and we also allow branching based on inequality tests. Each operation is assigned not one tree but many trees, namely one for each history where history means all discrete information obtained so far such as the sequence of operations applied earlier and the results of branching tests in earlier operations. The memory consists of variables holding field values that are preserved between operations. The variables may be written and read by the computation trees. The complexity of a solution is the maximal height of any tree in it. All our algorithms are within this basic model. We state explicitly, when our lower bounds are valid in a weaker model only (such as straightline programs).

Results. The following theorem is immediate from the lower bound for dynamic computation of the determinant [7].

Theorem 7. *Let the field $\mathcal{F}$ be infinite. The problem D has complexity $\Omega(n)$.*

If we allow the more general column updates, then the lower bound can be improved to $\Omega(n^2)$. Actually, this is a corollary to a stronger result we prove, namely a lower bound for maintaining whether a single coefficient is zero.

Theorem 8. *Let the field $\mathcal{F}$ contain the real numbers. Let $1 \leq l \leq n$.*

The problem D'_l has complexity $\Omega(\min(l, n-l))$ for symmetric matrices.

The problem D'_l has complexity $\Omega(\min(l, n-l)^2)$ for symmetric matrices when using vector updates.

The proof of this result is based on a strengthening of the lower bound for matrix rank from [8].

Theorem 9. *Let $\mathcal{F}$ be an algebraically closed field or the real numbers. Consider dynamic computation of* $\operatorname{rank}(M)$ *where $M \in \mathcal{F}^{(3n+1)^2}$, M is symmetric (vector*

updates must be paired into symmetric row-column updates) and $\mathrm{rank}(M)$ *must remain one of* $2n$ *and* $2n+2$*. This problem has complexity at least* $n^2/4$*.*

Proof (of Theorem 9). The proof uses a reduction from matrix vector multiplication verification (MVMV), where the MVMV problem consists in verifying that $Mx = y$ for square matrix M and column vectors x and y. The MVMV problem was introduced in [8], where a lower bound was shown for algebraically closed fields and element updates. Our main contribution is to extend the lower bound for MVMV to be valid for real numbers and vector updates, combined with an observation that the reduction from MVMV need only use the rank of symmetric matrices. For details see Appendix D[28].

Proof (of Theorem 8). It is known that the rank of a symmetric real matrix is precisely the number of nonzero roots of its characteristic polynomial [10]. Hence, for a matrix M as given in the statement of Theorem 9, we may distinguish between the two possible ranks simply by checking whether s_{2n+2} is zero. By embedding M in a larger matrix M_1 that has zeros elsewhere:

$$M_1 = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix},$$

we have the lower bound $\Omega(l^2)$ on deciding whether s_l is zero for $n \times n$ matrix when $l \leq \frac{2}{3}n$. Similarly, by embedding M in the upper left corner of a larger matrix M_2 that has an identity matrix in the lower right corner:

$$M_2 = \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix},$$

we have the lower bound $\Omega((n-l)^2)$ on deciding whether s_l is zero for $n \times n$ matrix when $l > \frac{2}{3}n$. Combining the two bounds, we get $\Omega(\min(l, n-l)^2)$ for all l. When adjusting the arguments towards single element updates rather than vector updates, one may similarly prove the lower bound $\Omega(\min(l, n-l))$ for all l. □

For the problem of maintaining the value of a coefficient rather than simply maintaining whether the coefficient is zero, we can prove slightly stronger lower bounds for element updates than those of Theorem 8 but partly in a more restrictive model (proof in Appendix E[28]).

Theorem 10. *Let the field* $\mathcal{F}$ *be infinite.*

The problem D_l *has a straightline solution of complexity* $O(1)$ *for* $l = 1, 2$*.*

The problem D_l *has complexity* $\Omega(\max(l, n/l))$ *for* $l \geq 3$*.*

The problem D_l *has complexity* $\Omega(n)$ *for* $l \geq 3$*, when the model of computation is restricted to history dependent straightline programs without division, i.e. using operations* $+, -, \cdot$ *only.*

3 Conclusion and Open Problems

In this paper we have proven that several fundamental problems in linear algebra allow to construct fully dynamic algorithms. We were able to show almost square worst-case time randomized dynamic algorithms in the generic case for the problems of computing: characteristic polynomial, tridiagonal symmetric form, Frobenius normal form, eigenvalues, eigenvectors, singular value decomposition and polynomial evaluated at the matrix. What is more important, the algorithms are practically applicable, i.e., work in worst-case time and the constant hidden in big-O is rather small. Moreover, we have been able to prove strong lower bounds for the problems. Hence, we have presented an extensive study of the arithmetic complexity of the problem. Nevertheless, our results rise a question whether similar but numerically correct algorithms can be obtained. We have decided to keep this issue out of the scope of the paper due to its size limitations. The following question are left open as well.

- The computation of the determinant can be carried out in subquadratic time in the case of element updates [9]. Is it possible to get similar algorithms in the case of CP?
- Can the query complexity for the eigenvectors in the above algorithm be reduced from $O(n^2 \log n)$ time to $\tilde{O}(n)$ time? This is possible when we are willing to spend $O(n^{2.5})$ time on updates — details will be included in the full version of the paper.
- It would be consistent with Theorem 10 if $s_{\sqrt{n}}$ could be maintained by computation trees of complexity $\sqrt{n}$, so an open problem is to extend the $\Omega(n)$ lower bound for D_l ($l \geq 3$) to be valid for algebraic computation trees with division.
- It would be consistent with the above results if one could maintain singularity of a matrix with a dynamic algorithm of complexity $O(1)$. In particular Theorem 8 gives $\Omega(n)$ bounds on the complexity of D'_l only for the middle range of l values. This leads to the open problem of proving an $\Omega(n)$ bound for D'_l for general l.
- It is the first time an $\Omega(n^2)$ lower bound for a dynamic matrix problem has been obtained. Can it extended to work for dynamic transitive closure?

Acknowledgements. This work was partially supported by the EU within the 6th Framework Programme under contract 001907 "Dynamically Evolving, Large Scale Information Systems" (DELIS), by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL) and by the Polish Ministry of Science, grant KBN-1P03A01830.

References

1. Giesbrecht, M.: Nearly optimal algorithms for canonical matrix forms. SIAM Journal on Computing 24(5), 948–969 (1995)
2. Eberly, W.: Asymptotically efficient algorithms for the Frobenius form. Paper 723-26, Department of Computer Science, University of Calgary (2003)

3. Villard, G.: Computing the Frobenius normal form of a sparse matrix. In: The Third International Workshop on Computer Algebra in Scientific Computing, pp. 395–407. Springer, Heidelberg (2000)
4. Storjohann, A.: Deterministic computation of the frobenius form. In: FOCS, pp. 368–377 (2001)
5. Ben-Amram, A., Galil, Z.: On pointers versus addresses. J. Assoc. Comput. Mach. 39, 617–648 (1992)
6. Pan, V., Chen, Z.: The complexity of the matrix eigenproblem. In: STOC 1999: Proceedings of the thirty-first annual ACM symposium on Theory of computing, pp. 507–516. ACM Press, New York (1999)
7. Frandsen, G., Hansen, J., Miltersen, P.: Lower bounds for dynamic algebraic problems. Inform. and Comput. 171(2), 333–349 (2001)
8. Frandsen, P., Frandsen, G.: Dynamic matrix rank. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 395–406. Springer, Berlin (2006)
9. Sankowski, P.: Dynamic Transitive Closure via Dynamic Matrix Inverse. In: FOCS, pp. 509–517 (2004)
10. Ibarra, O., Moran, S., Rosier, L.: A note on the parallel complexity of computing the rank of order n matrices. Inform. Process. Lett. 11(4-5), 162 (1980)
11. Mulmuley, K.: A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. In: STOC, pp. 338–339. ACM Press, New York (1986)
12. Chung, F.R.K.: Spectral Graph Theory. CBMS Regional Conference Series in Mathematics, vol. 92. American Mathematical Society (1997)
13. Babai, L., Grigoryev, D., Mount, D.: Isomorphism of graphs with bounded eigenvalue multiplicity. In: STOC 1982: Proceedings of the fourteenth annual ACM symposium on Theory of computing, New York, NY, USA, pp. 310–324 (1982)
14. Fiedler, M.: Algebraic connectivity of graphs. Czechoslovak Mathematical Journal 23(98), 289–305 (1973)
15. Spielman, D., Teng, S.H.: Spectral partitioning works: Planar graphs and finite element meshes. In: FOCS, pp. 96–105 (1996)
16. Weiss, Y.: Segmentation using eigenvectors: A unifying view. In: ICCV (2), pp. 975–982 (1999)
17. Ng, A., Jordan, M., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Proceedings of Advances in Neural Information Processing Systems 14 (2001)
18. Meyer, C.D., J.S.: Updating finite markov chains by using techniques of group matrix inversion. J. Statist. Comput. Simulat. 11, 163–181 (1980)
19. Funderlic, R.E., Plemmons, R.J.: Updating lu factorizations for computing stationary distributions. SIAM J. Algebraic Discrete Methods 7(1), 30–42 (1986)
20. Seneta, E.: Sensivity analysis, ergodicity coefficients, and rank-one updates for finite markov chains. Numerical Solutions of Markov Chains, 121–129 (1991)
21. Diaconis, P., Stroock, D.: Geometric bounds for eigenvalues of Markov chains. Ann. Appl. Probab. 1(1), 36–61 (1991), http://www.ams.org/mathscinet-getitem?mr=92h:60103
22. Golub, G.H., Van Loan, C.F.: Matrix computations, 3rd edn. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore (1996)
23. Chandrasekaran, S., Manjunath, B.S., Wang, Y.F., Winkeler, J., Zhang, H.: An eigenspace update algorithm for image analysis. Graph. Models Image Process 59(5), 321–332 (1997)
24. Kanth, K., Agrawal, D., Singh, A.: Dimensionality reduction for similarity searching in dynamic databases. In: Proceedings of the 1998 ACM SIGMOD international conference on Management of data, NY, USA, pp. 166–176 (1998)

25. Brand, M.: Fast online svd revisions for lightweight recommender systems. In: Barbará, D., Kamath, C. (eds.) SDM. SIAM, Philadelphia (2003)
26. Gu, M., Eisenstat, S.C.: A stable and fast algorithm for updating singular value decomposition. Technical Report YALE/DCS/TR-966, Yale University, New Haven, CT (1993)
27. Gu, M., Eisenstat, S.: Downdating the singular value decomposition. SIAM Journal on Matrix Analysis and Applications 16(3), 793–810 (1995)
28. Frandsen, G.S., Sankowski, P.: Dynamic normal forms and dynamic characteristic polynomial. Research Series RS-08-2, BRICS, Department of Computer Science, University of Aarhus (2008), http://www.brics.dk/RS/08/2/index.html
29. Bürgisser, P., Clausen, M., Shokrollahi, M.: Algebraic complexity theory. Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 315. Springer, Berlin (1997)
30. Bini, D., Pan, V.: Polynomial and Matrix Computations. Birkhäuser (1994)
31. Eberly, W., Kaltofen, E.: On randomized lanczos algorithms. In: ISSAC 1997: Proceedings of the 1997 international symposium on Symbolic and algebraic computation, pp. 176–183. ACM Press, New York (1997)

Algorithms for ε-Approximations of Terrains*

Jeff M. Phillips

Department of Computer Science, Duke University, Durham, NC 27708
jeffp@cs.duke.edu

Abstract. Consider a point set $\mathcal{D}$ with a measure function $\mu : \mathcal{D} \to \mathbb{R}$. Let $\mathcal{A}$ be the set of subsets of $\mathcal{D}$ induced by containment in a shape from some geometric family (e.g. axis-aligned rectangles, half planes, balls, k-oriented polygons). We say a range space $(\mathcal{D}, \mathcal{A})$ has an ε-approximation P if

$$\max_{R \in \mathcal{A}} \left| \frac{\mu(R \cap P)}{\mu(P)} - \frac{\mu(R \cap \mathcal{D})}{\mu(\mathcal{D})} \right| \leq \varepsilon.$$

We describe algorithms for deterministically constructing discrete ε-app- roximations for continuous point sets such as distributions or terrains. Furthermore, for certain families of subsets $\mathcal{A}$, such as those described by axis-aligned rectangles, we reduce the size of the ε-approximations by almost a square root from $O(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ to $O(\frac{1}{\varepsilon} \operatorname{polylog} \frac{1}{\varepsilon})$. This is often the first step in transforming a continuous problem into a discrete one for which combinatorial techniques can be applied. We describe applications of this result in geo-spatial analysis, biosurveillance, and sensor networks.

1 Introduction

Representing complex objects by point sets may require less storage and may make computation on them faster and easier. When properties of the point set approximate those of the original object, then problems over continuous or piecewise-linear domains are now simple combinatorial problems over point sets. For instance, when studying terrains, representing the volume by the cardinality of a discrete point set transforms calculating the difference between two terrains in a region to just counting the number of points in that region. Alternatively, if the data is already a discrete point set, approximating it with a much smaller point set has applications in selecting sentinel nodes in sensor networks. This paper studies algorithms for creating small samples with guarantees in the form of discrepancy and ε-approximations, in particular we construct ε-approximations of size $O(\frac{1}{\varepsilon} \operatorname{polylog} \frac{1}{\varepsilon})$.

* Work on this paper is supported by a James B. Duke Fellowship, by NSF under a Graduate Research Fellowship and grants CNS-05-40347, CFF-06-35000, and DEB-04-25465, by ARO grants W911NF-04-1-0278 and W911NF-07-1-0376, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEGP200A070505, and by a grant from the U.S. Israel Binational Science Foundation.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 447–458, 2008.

ε-approximations. In this paper we study point sets, which we call domains and we label as $\mathcal{D}$, which are either finite sets or are Lebesgue-measureable sets. For a given domain $\mathcal{D}$ let $\mathcal{A}$ be a set of subsets of $\mathcal{D}$ induced by containment in some geometric shape (such as balls or axis-aligned rectangles). The pair $(\mathcal{D}, \mathcal{A})$ is called a *range space*. We say that P is an *ε-approximation* of $(\mathcal{D}, \mathcal{A})$ if

$$\max_{R \in \mathcal{A}} \left| \frac{|R \cap P|}{|P|} - \frac{|R \cap \mathcal{D}|}{|\mathcal{D}|} \right| \leq \varepsilon,$$

where $|\cdot|$ represents the cardinality of a discrete set or the Lebesgue measure for a Lebesgue-measurable set. $\mathcal{A}$ is said to *shatter* a discrete set $X \subseteq \mathcal{D}$ if each subset of X is equal to $R \cap X$ for some $R \in \mathcal{A}$. The cardinality of the largest discrete set X that $\mathcal{A}$ can shatter is known as the *VC-dimension*. A classic result of Vapnik and Chervonenkis [28] states that for any range space $(\mathcal{D}, \mathcal{A})$ with constant VC-dimension v there exists a subset $P \subset \mathcal{D}$ consisting of $O(\frac{v}{\varepsilon^2} \log \frac{v}{\varepsilon})$ points that is an ε-approximation for $(\mathcal{D}, \mathcal{A})$. Furthermore, if each element of P is drawn uniformly at random from $\mathcal{D}$ such that $|P| = O(\frac{v}{\varepsilon^2} \log \frac{v}{\varepsilon\delta})$, then P is an ε-approximation with probability at least $1 - \delta$. Thus, for a large class of range spaces random sampling produces an ε-approximation of size $O(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$.

Deterministic construction of ε-approximations. There exist deterministic constructions for ε-approximations. When $\mathcal{D}$ is the unit cube $[0,1]^d$ there are constructions which can be interpreted as ε-approximations of size $O(\frac{1}{\varepsilon^{2d/(d+1)}})$ for half spaces [16] and $O(\frac{1}{\varepsilon^{2d/(d+1)}} \log^{d/(d+1)} \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$ for balls in d-dimensions [5]. Both have lower bounds of $\Omega(\frac{1}{\varepsilon^{2d/(d+1)}})$ [2]. See Matoušek [17] for more similar results or Chazelle's book [9] for applications. For a domain $\mathcal{D}$, let $\mathcal{R}_d$ describe the subsets induced by axis-parallel rectangles in d dimensions, and let $\mathcal{Q}_k$ describe the subsets induced by k-oriented polygons (or more generally polytopes) with faces described by k predefined normal directions. More precisely, for $\beta = \{\beta_1, \ldots, \beta_k\} \subset \mathbb{S}^{d-1}$, let $\mathcal{Q}_\beta$ describe the set of convex polytopes such that each face has an outward normal $\pm\beta_i$ for $\beta_i \in \beta$. If β is fixed, we will use $\mathcal{Q}_k$ to denote $\mathcal{Q}_\beta$ since it is the size k and not the actual set β that is important. When $\mathcal{D} = [0,1]^d$, then the range space $(\mathcal{D}, \mathcal{R}_d)$ has an ε-approximation of size $O(\frac{1}{\varepsilon} \log^{d-1} \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$ [12]. Also, for all homothets (translations and uniform scalings) of any particular $Q \in \mathcal{Q}_k$, Skriganov constructs an ε-approximation of size $O(\frac{1}{\varepsilon} \log^{d-1} \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$. When $\mathcal{D}$ is a discrete point set of size n, ε-approximations of size $O((\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})^{2 - \frac{2}{v+1}})$ exist for bounded VC-dimension v [19], and can be constructed in time $O(n \cdot \frac{1}{\varepsilon^{2v}} \log^v \frac{1}{\varepsilon})$. In this spirit, for $\mathcal{R}_2$ and a discrete point set of size n, Suri, Toth, and Zhou [26] construct an ε-approximation of size $O(\frac{1}{\varepsilon} \log(\varepsilon n) \log^4(\frac{1}{\varepsilon} \log(\varepsilon n)))$ in the context of a streaming algorithm which can be analyzed to run in time $O(n(\frac{1}{\varepsilon} \log^4 \frac{1}{\varepsilon})^3)$.

Our results. We answer the question, "*for which ranges spaces can we construct ε-approximations of size $O(\frac{1}{\varepsilon} \text{ polylog} \frac{1}{\varepsilon})$*?" by describing how to deterministically construct an ε-approximation of size $O(\frac{1}{\varepsilon} \text{ polylog} \frac{1}{\varepsilon})$ for any domain which can be decomposed into or approximated by a finite set of constant-size polytopes for families $\mathcal{R}_d$ and $\mathcal{Q}_k$. In particular:

- For a discrete point set $\mathcal{D}$ of cardinality n, we give an algorithm for generating an ε-approximation for $(\mathcal{D}, \Omega_k)$ of size $O(\frac{1}{\varepsilon}\log^{2k}\frac{1}{\varepsilon}\text{polylog}(\log\frac{1}{\varepsilon}))$ in $O(n\frac{1}{\varepsilon^3}\text{ polylog}\frac{1}{\varepsilon})$ time. This requires a generalization of the iterative point set thinning algorithm by Chazelle and Matoušek [10] that does not rely on VC-dimension. This implies similar results for $\mathcal{R}_d$ as well.
- For any d-dimensional domain $\mathcal{D}$ that can be decomposed into n k'-oriented polytopes, we give an algorithm for generating an ε-approximation of size $O((k+k')\frac{1}{\varepsilon}\log^{2k}\frac{1}{\varepsilon}\text{polylog}(\log\frac{1}{\varepsilon}))$ for $(\mathcal{D}, \Omega_k)$ in time $O((k+k')n\frac{1}{\varepsilon^4}\text{ polylog}\frac{1}{\varepsilon})$.

We are interested in terrain domains $\mathcal{D}$ defined to have a base B (which may, for instance, be a subset of $\mathbb{R}^2$) and a height function $h : B \to \mathbb{R}$. Any point (p, z) such that $p \in B$ and $0 \leq z \leq h(p)$ (or $0 \geq z \geq h(p)$ when $h(p) < 0$) is in the domain $\mathcal{D}$ of the terrain.

- For a terrain domain $\mathcal{D}$ where B and h are piecewise-linear with n linear pieces, our result implies that there exists an ε-approximation of size $O(k\frac{1}{\varepsilon}\log^4\frac{1}{\varepsilon}\text{polylog}(\log\frac{1}{\varepsilon}))$ for $(\mathcal{D}, \Omega_k)$, and it can be constructed in $O(n\cdot\frac{1}{\varepsilon^4}\text{ polylog}\frac{1}{\varepsilon})$ time.
- For a terrain domain $\mathcal{D}$ where $B \subset \mathbb{R}^2$ is a rectangle with diameter d and h is smooth (C^2-continuous) with minimum height z^- and largest eigenvalue of its Hessian λ, we give an algorithm for creating an ε-approximation for $(\mathcal{D}, \mathcal{R}_2 \times \mathbb{R})$ of size $O(\frac{1}{\varepsilon}\log^4\frac{1}{\varepsilon}\text{polylog}(\log\frac{1}{\varepsilon}))$ in time $O(\frac{\lambda d^2}{z^-}\frac{1}{\varepsilon^5}\text{ polylog}\frac{1}{\varepsilon})$.

These results improve the running time for a spatial anomaly detection problem in biosurveillance [1], and can more efficiently place or choose sentinel nodes in a sensor network, addressing an open problem [21].

Roadmap. We introduce a variety of new techniques, rooted in discrepancy theory, to create ε-approximations of size $O(\frac{1}{\varepsilon}\text{ polylog}\frac{1}{\varepsilon})$ for increasingly difficult domains. First, Section 2 discusses Lebesgue and combinatorial discrepancy. Section 3 generalizes and improves a classic technique to create an ε-approximation for a discrete point set. Section 4 describes how to generate an ε-approximation for a polygonal domain. When a domain can be decomposed into a finite, disjoint set of polygons, then each can be given an ε-approximation and the union of all these point sets can be given a smaller ε-approximation using the techniques in Section 3. Section 5 then handles domains of continuous, non-polygonal point sets by first approximating them by a disjoint set of polygons and then using the earlier described techniques. Section 6 shows some applications of these results.

2 Lebesgue and Combinatorial Discrepancy

Lebesgue discrepancy. The Lebesgue discrepancy is defined for an n-point set $P \subset [0,1]^d$ relative to the volume of a unit cube $[0,1]^d$.[1] Given a range space $([0,1]^d, \mathcal{A})$ and a point set P, the *Lebesgue* discrepancy is defined

[1] Although not common in the literature, this definition can replace $[0,1]^d$ with an hyper-rectangle $[0, w_1] \times [0, w_2] \times \ldots \times [0, w_d]$.

$$D(P, \mathcal{A}) = \sup_{R \in \mathcal{A}} |D(P, R)|, \quad \text{where} \quad D(P, R) = n \cdot |R \cap [0,1]^d| - |R \cap P|.$$

Optimized over all n-point sets, define the *Lebesgue discrepancy of* $([0,1]^d, \mathcal{A})$ as

$$D(n, \mathcal{A}) = \inf_{P \subset [0,1]^d, |P|=n} D(P, \mathcal{A}).$$

The study of Lebesgue discrepancy arguably began with the Van der Corput set C_n [27], which satisfies $D(C_n, \mathcal{R}_2) = O(\log n)$. This was generalized to higher dimensions by Hammersley [13] and Halton [12] so that $D(C_n, \mathcal{R}_d) = O(\log^{d-1} n)$. However, it was shown that many lattices also provide $O(\log n)$ discrepancy in the plane [17]. This is generalized to $O(\log^{d-1} n \log^{1+\tau} \log n)$ for $\tau > 0$ over $\mathcal{R}^d$ [22,23,6]. For a more in-depth history of the progression of these results we refer to the notes in Matoušek's book [17]. For application of these results in numerical integration see Niederreiter's book [20]. The results on lattices extend to homothets of any $Q_k \in \mathcal{Q}_k$ for $O(\log n)$ discrepancy in the plane [22] and $O(\log^{d-1} n \log^{1+\tau} \log n)$ discrepancy, for $\tau > 0$, in $\mathbb{R}^d$ [24], for some constant k. A wider set of geometric families which include half planes, right triangles, rectangles under all rotations, circles, and predefined convex shapes produce $\Omega(n^{1/4})$ discrepancy and are not as interesting from our perspective.

Lebesgue discrepancy describes an ε-approximation of $([0,1]^d, \mathcal{A})$, where $\varepsilon = f(n) = D(n, \mathcal{A})/n$. Thus we can construct an ε-approximation for $([0,1]^d, \mathcal{A})$ of size $g_{\mathcal{D}}(\varepsilon, \mathcal{A})$ as defined below. (Solve for n in $\varepsilon = D(n, \mathcal{A})/n$).)

$$g_D(\varepsilon, \mathcal{A}) = \begin{cases} O(\frac{1}{\varepsilon} \log^{\tau} \frac{1}{\varepsilon} \operatorname{polylog}(\log \frac{1}{\varepsilon})) & \text{for } D(n, \mathcal{A}) = O(\log^{\tau} n) \\ O((1/\varepsilon)^{1/(1-\tau)}) & \text{for } D(n, \mathcal{A}) = O(n^{\tau}) \end{cases} \tag{1}$$

Combinatorial discrepancy. Given a range space $(X, \mathcal{A})$ where X is a finite point set and a coloring function $\chi : X \to \{-1, +1\}$ we say the *combinatorial discrepancy* of $(X, \mathcal{A})$ colored by χ is

$$\operatorname{disc}_{\chi}(X, \mathcal{A}) = \max_{R \in \mathcal{A}} \operatorname{disc}_{\chi}(X \cap R) \quad \text{where}$$

$$\operatorname{disc}_{\chi}(X) = \sum_{x \in X} \chi(x) = |\{x \in X \ : \ \chi(x) = +1\}| - |\{x \in X \ : \ \chi(x) = -1\}|.$$

Taking this over all colorings and all point sets of size n we say

$$\operatorname{disc}(n, \mathcal{A}) = \max_{\{X : |X| = n\}} \min_{\chi : X \to \{-1,+1\}} \operatorname{disc}_{\chi}(X, \mathcal{A}).$$

Results about combinatorial discrepancy are usually proved using the partial coloring method [4] or the Beck-Fiala theorem [8]. The partial coloring method usually yields lower discrepancy by some logarithmic factors, but is nonconstructive. Alternatively, the Beck-Fiala theorem actually constructs a low discrepancy coloring, but with a slightly weaker bound. The Beck-Fiala theorem states that for a family of ranges $\mathcal{A}$ and a point set X such that

$\max_{x \in X} |\{A \in \mathcal{A} \ : \ x \in A\}| \leq t$, $\text{disc}(X, \mathcal{A}) \leq 2t - 1$. So the discrepancy is only a constant factor larger than the largest number of sets any point is in.

Srinivasan [25] shows that $\text{disc}(n, \mathcal{R}_2) = O(\log^{2.5} n)$, using the partial coloring method. An earlier result of Beck [3] showed $\text{disc}(n, \mathcal{R}_2) = O(\log^4 n)$ using the Beck-Fiala theorem [8]. The construction in this approach reduces to $O(n)$ Gaussian eliminations on a matrix of constraints that is $O(n) \times O(n)$. Each Gaussian elimination step requires $O(n^3)$ time. Thus the coloring χ in the construction for $\text{disc}(n, \mathcal{R}_2) = O(\log^4 n)$ can be found in $O(n^4)$ time.We now generalize this result.

Lemma 1. $\text{disc}(n, \mathcal{Q}_k) = O(\log^{2k} n)$ *for points in* $\mathbb{R}^d$ *and the coloring that generates this discrepancy can be constructed in* $O(n^4)$ *time, for* k *constant.*

The proof combines techniques from Beck [3] and Matoušek [18].

Proof. Given a class $\mathcal{Q}_k$, each potential face is defined by a normal vector from $\{\beta_1, \ldots, \beta_k\}$. For $j \in [1, k]$ project all points along β_j. Let a *canonical interval* be of the form $\left[\frac{t}{2^q}, \frac{t+1}{2^q}\right)$ for integers $q \in [1, \log n]$ and $t \in [0, 2^q)$. For each direction β_j choose a value $q \in [1, \log n]$ creating 2^q canonical intervals induced by the ordering along β_j. Let the intersection of any k of these canonical intervals along a fixed β_j be a *canonical subset*. Since there are $\log n$ choices for the values of q for each of the k directions, it follows that each point is in at most $(\log n)^k$ canonical subsets. Using the Beck-Fiala theorem, we can create a coloring for X so that no canonical subset has discrepancy more than $O(\log^k n)$.

Each range $R \in \mathcal{Q}_k$ is formed by at most $O(\log^k n)$ canonical subsets. For each ordering by β_i, the interval in this ordering induced by R can be described by $O(\log n)$ canonical intervals. Thus the entire range R can be decomposed into $O(\log^k n)$ canonical subsets, each with at most $O(\log^k n)$ discrepancy.

Applying the Beck-Fiala construction of size n, this coloring requires $O(n^4)$ time to construct.

Corollary 1. *disc*$(n, \mathcal{R}_d) = O(\log^{2d} n)$ *and the coloring that generates this discrepancy can be constructed in* $O(n^4)$ *time, for* d *constant.*

A better nonconstructive bound exists due to Matoušek [18], using the partial coloring method. For polygons in $\mathbb{R}^2$ $\text{disc}(n, Q_k) = O(k \log^{2.5} n \sqrt{\log(k + \log n)})$, and for polytopes in $\mathbb{R}^d$ $\text{disc}(n, Q_k) = O(k^{1.5\lfloor d/2 \rfloor} \log^{d+1/2} n \sqrt{\log(k + \log n)})$. For more results on discrepancy see Beck and Chen's book [7].

Similar to Lebesgue discrepancy, the set $P = \{p \in X \mid \chi(p) = +1\}$ generated from the coloring χ for combinatorial discrepancy $\text{disc}(n, \mathcal{A})$ describes an ε-approximation of $(X, \mathcal{A})$ where $\varepsilon = f(n) = \text{disc}(n, \mathcal{A})/n$. Thus, given this value of ε, we can say that P is an ε-approximation for $(X, \mathcal{A})$ of size

$$g(\varepsilon, \mathcal{A}) = \begin{cases} O(\frac{1}{\varepsilon} \log^\tau \frac{1}{\varepsilon} \text{ polylog}(\log \frac{1}{\varepsilon})) & \text{for } \text{disc}(n, \mathcal{A}) = O(\log^\tau n) \\ O((1/\varepsilon)^{1/(1-\tau)}) & \text{for } \text{disc}(n, \mathcal{A}) = O(n^\tau). \end{cases} \quad (2)$$

The next section will describe how to iteratively apply this process efficiently to achieve these bounds for any value of ε.

3 Deterministic Construction of ε-Approximations for Discrete Point Sets

We generalize the framework of Chazelle and Matoušek [10] describing an algorithm for creating an ε-approximation of a range space $(X, \mathcal{A})$. Consider any range space $(X, \mathcal{A})$, with $|X| = n$, for which there is an algorithm to generate a coloring χ that yields the combinatorial discrepancy $\text{disc}_\chi(X, \mathcal{A})$ and can be constructed in time $O(n^w \cdot l(n))$ where $l(n) = o(n)$. For simplicity, we refer to the combinatorial discrepancy we can construct $\text{disc}_\chi(X, \mathcal{A})$ as $\text{disc}(n, \mathcal{A})$ to emphasize the size of the domain, and we use equation (2) to describe $g(\varepsilon, \mathcal{A})$, the size of the ε-approximation it corresponds to. The values $\text{disc}(n, \mathcal{A})$, w, and $l(n)$ are dependent on the family $\mathcal{A}$ (e.g. see Lemma 1), but not necessarily its VC-dimension as in [10]. As used above, let $f(n) = \text{disc}(n, \mathcal{A})/n$ be the value of ε in the ε-approximation generated by a single coloring of a set of size n — the relative error. We require that, $f(2n) \leq (1 - \delta)f(n)$, for constant $0 < \delta \leq 1$; thus it is a geometrically decreasing function.

The algorithm will compress a set X of size n to a set P of size $O(g(\varepsilon, \mathcal{A}))$ such that P is an ε-approximation of $(X, \mathcal{A})$ by recursively creating a low discrepancy coloring. We note that an ε-approximation of an ε'-approximation is an $(\varepsilon + \varepsilon')$-approximation of the original set.

We start by dividing X into sets of size $O(g(\varepsilon, \mathcal{A}))$,[2] here ε is a parameter. The algorithm proceeds in two stages. The first stage alternates between merging pairs of sets and halving sets by discarding points colored $\chi(p) = -1$ by the combinatorial discrepancy method described above. The exception is after every $w + 2$ halving steps, we then skip one halving step. The second stage takes the one remaining set and repeatedly halves it until the error $f(|P|)$ incurred in the remaining set P exceeds $\frac{\varepsilon}{2+2\delta}$. This results in a set of size $O(g(\varepsilon, \mathcal{A}))$.

Algorithm 3.1. Creates an ε-approximation for $(X, \mathcal{A})$ of size $O(g(\varepsilon, \mathcal{A}))$.

1: Divide X into sets $\{X_0, X_1, X_2, \ldots\}$ each of size $4(w + 2)g(\varepsilon, \mathcal{A})$. [2]
2: **repeat** {*Stage 1*}
3: **for** $w + 2$ steps **do** {or stop if only one set is left}
4: MERGE: Pair sets arbitrarily (i.e. X_i and X_j) and merge them into a single set (i.e. $X_i := X_i \cup X_j$).
5: HALVE: Halve each set X_i using the coloring χ from $\text{disc}(X_i, \mathcal{A})$ (i.e. $X_i = \{x \in X_i \mid \chi(x) = +1\}$).
6: MERGE: Pair sets arbitrarily and merge each pair into a single set.
7: **until** only one set, P, is left
8: **repeat** {*Stage 2*}
9: HALVE: Halve P using the coloring χ from $\text{disc}(P, \mathcal{A})$.
10: **until** $f(|P|) \geq \varepsilon/(2 + 2\delta)$

[2] If the sets do not divide equally, artificially increase the size of the sets when necessary. These points can be removed later.

Theorem 1. *For a finite range space $(X, \mathcal{A})$ with $|X| = n$ and an algorithm to construct a coloring $\chi : X \to \{-1, +1\}$ such that*

- *the set $\{x \in X : \chi(x) = +1\}$ is an α-approximation of $(X, \mathcal{A})$ of size $g(\alpha, \mathcal{A})$ with $\alpha = disc_\chi(X, \mathcal{A})/n$ (see equation (2)).*
- *χ can be constructed in $O(n^w \cdot l(n))$ time where $l(n) = o(n)$.*

then Algorithm 3.1 constructs an ε-approximation for $(X, \mathcal{A})$ of size $O(g(\varepsilon, \mathcal{A}))$ in time $O(w^{w-1} n \cdot g(\varepsilon, \mathcal{A})^{w-1} \cdot l(g(\varepsilon, \mathcal{A})) + g(\varepsilon, \mathcal{A}))$.

Proof. Let $2^j = 4(w+2)g(\varepsilon, \mathcal{A})$, for an integer j, be the size of each set in the initial dividing stage (adjusting by a constant if $\delta \leq \frac{1}{4}$). Each round of Stage 1 performs $w+3$ MERGE steps and $w+2$ HALVE steps on sets of the same size and each subsequent round deals with sets twice as large. The union of all the sets is an α-approximation of $(X, \mathcal{A})$ (to start $\alpha = 0$) and α only increases in the HALVE steps. The ith round increases α by $f(2^{j-1+i})$ per HALVE step. Since $f(n)$ decrease geometrically as n increases, the size of α at the end of the first stage is asymptotically bounded by the increase in the first round. Hence, after Stage 1 $\alpha \leq 2(w+2)f(4(w+2)g(\varepsilon, \mathcal{A})) \leq \frac{\varepsilon}{2}$. Stage 2 culminates the step before $f(|P|) \geq \frac{\varepsilon}{2+2\delta}$. Thus the final HALVE step creates an $\frac{\varepsilon\delta}{2+2\delta}$-approximation and the entire second stage creates an $\frac{\varepsilon}{2}$-approximation, hence overall Algorithm 3.1 creates an ε-approximation. The relative error caused by each HALVE step in stage 2 is equivalent to a HALVE step in a single round of stage 1.

The running time is also dominated by Stage 1. Each HALVE step of a set of size 2^j takes $O((2^j)^w l(2^j))$ time and runs on $n/2^j$ sets. In between each HALVE step within a round, the number of sets is divided by two, so the running time is asymptotically dominated by the first HALVE step of each round. The next round has sets of size 2^{j+1}, but only $n/2^{j+w+2}$ of them, so the runtime is at most $\frac{1}{2}$ that of the first HALVE step. Thus the running time of a round is less than half of that of the previous one. Since $2^j = O(wg(\varepsilon, \mathcal{A}))$ the running time of the HALVE step, and hence the first stage is bounded by $O(n \cdot (w \cdot g(\varepsilon, \mathcal{A}))^{w-1} \cdot l(g(\varepsilon, \mathcal{A})) + g(\varepsilon, \mathcal{A}))$. Each HALVE step in the second stage corresponds to a single HALVE step per round in the first stage, and does not affect the asymptotics.

We can invoke Theorem 1 along with Lemma 1 and Corollary 1 to compute χ in $O(n^4)$ time (notice that $w = 4$ and $l(\cdot)$ is constant), so $g(\varepsilon, \Omega_k) = O(\frac{1}{\varepsilon} \log^{2k} \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$ and $g(\varepsilon, \mathcal{R}_d) = O(\frac{1}{\varepsilon} \log^{2d} \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$. We obtain the following important corollaries.

Corollary 2. *For a set of size n and over the ranges Ω_k an ε-approximation of size $O(\frac{1}{\varepsilon} \log^{2k} \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$ can be constructed in time $O(n \frac{1}{\varepsilon^3} \text{polylog} \frac{1}{\varepsilon})$.*

Corollary 3. *For a set of size n and over the ranges $\mathcal{R}_d$ an ε-approximation of size $O(\frac{1}{\varepsilon} \log^{2d} \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$ can be constructed in time $O(n \frac{1}{\varepsilon^3} \text{polylog} \frac{1}{\varepsilon})$.*

Weighted case. These results can be extended to the case where each point $x \in X$ is given a weight $\mu(x)$. Now an ε-approximation is a set $P \subset X$ and a weighting $\mu : X \to \mathbb{R}$ such that

$$\max_{R \in \mathcal{A}} \left| \frac{\mu(P \cap R)}{\mu(P)} - \frac{\mu(X \cap R)}{\mu(X)} \right| \leq \varepsilon,$$

where $\mu(P) = \sum_{p \in P} \mu(p)$. The weights on P may differ from those on X. A result from Matoušek [15], invoking the unweighted algorithm several times at a geometrically decreasing cost, creates a weighted ε-approximation of the same asymptotic size and with the same asymptotic runtime as for an unweighted algorithm. This extension is important when we combine ε-approximations representing regions of different total measure. For this case we weight each point relative to the measure it represents.

4 Sampling from Polygonal Domains

We will prove a general theorem for deterministically constructing small ε-approximations for polygonal domains which will have direct consequences on polygonal terrains. A key observation of Matoušek [15] is that the union of ε-approximations of disjoint domains forms an ε-approximation of the union of the domains. Thus for any geometric domain $\mathcal{D}$ we first divide it into pieces for which we can create ε-approximations. Then we merge all of these point sets into an ε-approximation for the entire domain. Finally, we use Theorem 1 to reduce the sample size.

Instead of restricting ourselves to domains which we can divide into cubes of the form $[0,1]^d$, thus allowing the use of Lebesgue discrepancy results, we first expand on a result about lattices and polygons.

Lattices and polygons. For $x \in \mathbb{R}$, let $\lfloor x \rfloor$ represent the fractional part of x, and for $\alpha \in \mathbb{R}^{d-1}$ let $\alpha = (\alpha_1, \ldots, \alpha_{d-1})$. Now given α and m let $P_{\alpha,m} = \{p_0, \ldots, p_{m-1}\}$ be a set of m lattice points in $[0,1]^d$ defined $p_i = (\frac{i}{m}, \lfloor \alpha_1 i \rfloor, \ldots, \lfloor \alpha_{d-1} i \rfloor)$. $P_{\alpha,m}$ is *irrational* with respect to any polytope in $\mathcal{Q}_\beta$ if for all $\beta_i \in \beta$, for all $j \leq d$, and for all $h \leq d-1$, the fraction $\beta_{i,j}/\alpha_h$ is irrational. (Note that $\beta_{i,j}$ represents the jth element of the vector β_i.) Lattices with α irrational (relative to the face normals) generate low discrepancy sets.

Theorem 2. *Let $Q \in \mathcal{Q}_{\beta'}$ be a fixed convex polytope. Let $\beta, \beta' \subset \mathbb{S}^{d-1}$ be sets of k and k' directions, respectively. There is an ε-approximation of $(Q, \mathcal{Q}_\beta)$ of size $O((k+k')\frac{1}{\varepsilon}\log^{d-1}\frac{1}{\varepsilon}\text{polylog}(\log\frac{1}{\varepsilon}))$.*

This ε-approximation is realized by a set of lattice points $P_{\alpha,m} \cap Q$ such that $P_{\alpha,m}$ is irrational with respect to any polytope in $\mathcal{Q}_{\beta \cup \beta'}$.

Proof. Consider polytope tQ_h and lattice $P_{\alpha,m}$, where the uniform scaling factor t is treated as an asymptotic quantity. Skriganov's Theorem 6.1 in [24] claims

$$\max_{v \in \mathbb{R}^d} D(P_{\alpha,m}, tQ_h + v) = O\left(t^{d-1}\rho^{-\theta} + \sum_f S_f(P_{\alpha,m}, \rho) \right)$$

where

$$S_f(P_{\alpha,m}, \rho) = O(\log^{d-1} \rho \log^{1+\tau} \log \rho)$$

for $\tau > 0$, as long as $P_{\alpha,m}$ is irrational with respect to the normal of the face f of Q_h and infinite otherwise, where $\theta \in (0,1)$ and ρ can be arbitrarily large. Note that this is a simplified form yielded by invoking Theorem 3.2 and Theorem 4.5 from [24]. By setting $\rho^\theta = t^{d-1}$,

$$\max_{v \in \mathbb{R}^d} D(P_{\alpha,m}, tQ_h + v) = O(h \log^{d-1} t \log^{1+\tau} \log t). \tag{3}$$

Now by noting that as t grows, the number of lattice points in tQ_h grows by a factor of t^d, and we can set $t = n^{1/d}$ so (3) implies that $D(P_{\alpha,m}, tQ_h) = O(h \log^{d-1} n \log^{1+\tau} \log n)$ for $|P_{\alpha,m}| = m = n$ and $tQ_h \subset [0,1]^d$.

The discrepancy is a sum over the set of h terms, one for each face f, each of which is small as long as $P_{\alpha,m}$ is irrational with respect to f's normal β_f. Hence this lattice gives low discrepancy for any polytope in the analogous family $\mathcal{Q}_\beta$ such that $P_{\alpha,m}$ is irrational with respect to $\mathcal{Q}_\beta$. Finally we realize that any subset $Q \cap Q_k$ for $Q \in \mathcal{Q}_{\beta'}$ and $Q_k \in \mathcal{Q}_\beta$ is a polytope defined by normals from $\beta' \cup \beta$ and we then refer to $g_D(\varepsilon, \mathcal{Q}_{\beta \cup \beta'})$ in (1) to bound the size of the ε-approximation from the given Lebesgue discrepancy.

Remark 1. Skriganov's result [24] is proved under the *whole space* model where the lattice is infinite (tQ_h is not confined to $[0,1]^d$), and the relevant error is the difference between the measure of tQ_h versus the cardinality $|tQ_h \cap P_{\alpha,m}|$, where each $p \in P_{\alpha,m}$ represents 1 unit of measure. Skriganov's main results in this model is summarized in equation (3) and only pertains to a fixed polytope Q_h instead of, more generally, a family of polytopes $\mathcal{Q}_\beta$, as shown in Theorem 2.

Samples for polygonal terrains. Combining the above results and weighted extension of Theorem 1 implies the following results.

Theorem 3. *We can create a weighted ε-approximation of size $O((k+k')\frac{1}{\varepsilon} \cdot \log^{2k} \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$ of $(\mathcal{D}, \mathcal{Q}_k)$ in time $O((k+k')n\frac{1}{\varepsilon^4} \text{ polylog}\frac{1}{\varepsilon})$ for any d-dimensional domain $\mathcal{D}$ which can be decomposed into n d-dimensional convex k'-oriented polytopes.*

Proof. We divide the domain into n k'-oriented polytopes and then approximate each polytope $Q_{k'}$ with a point set $P_{\alpha,m} \cap Q_{k'}$ using Theorem 2. We observe that the union of these point sets is a weighted ε-approximation of $(\mathcal{D}, \mathcal{Q}_k)$, but is quite large. Using the weighted extension of Theorem 1 we can reduce the point sets to the size and in the time stated.

This has applications to terrain domains $\mathcal{D}$ defined with a piecewise-linear base B and height function $h : B \to \mathbb{R}$. We decompose the terrain so that each linear piece of h describes one 3-dimensional polytope, then apply Theorem 3 to get the following result.

Corollary 4. *For terrain domain $\mathcal{D}$ with piecewise-linear base B and height function $h : B \to \mathbb{R}$ with n linear pieces, we construct a weighted ε-approximation of $(\mathcal{D}, \mathcal{Q}_k)$ of size $O(k\frac{1}{\varepsilon} \log^4 \frac{1}{\varepsilon} \text{polylog}(\log \frac{1}{\varepsilon}))$ in time $O(kn\frac{1}{\varepsilon^4} \text{ polylog}\frac{1}{\varepsilon})$.*

5 Sampling from Smooth Terrains

We can create an ε-approximation for a smooth domain (one which cannot be decomposed into polytopes) in a three stage process. The first stage approximates any domain with a set of polytopes. The second approximates each polytope with a point set. The third merges all point sets and uses Theorem 1 to reduce their size.

This section mainly focuses on the first stage. More formally, we can approximate a non-polygonal domain $\mathcal{D}$ with a set of disjoint polygons P such that P has properties of an ε-approximation.

Lemma 2. *If* $|\mathcal{D} \setminus P| \leq \frac{\varepsilon}{2}|\mathcal{D}|$ *and* $P \subseteq \mathcal{D}$ *then* $\max_{R \in \mathcal{A}} \left| \frac{|R \cap P|}{|P|} - \frac{|R \cap \mathcal{D}|}{|\mathcal{D}|} \right| \leq \varepsilon.$

Proof. No range $R \in \mathcal{A}$ can have $\left| \frac{|R \cap P|}{|P|} - \frac{|R \cap \mathcal{D}|}{|\mathcal{D}|} \right| > \varepsilon$ because if $|\mathcal{D}| \geq |P|$ (w.l.o.g.), then $|R \cap \mathcal{D}| - \frac{|\mathcal{D}|}{|P|}|R \cap P| \leq \varepsilon|\mathcal{D}|$ and $|R \cap P|\frac{|\mathcal{D}|}{|P|} - |R \cap \mathcal{D}| \leq \varepsilon|\mathcal{D}|$. The first part follows from $\frac{|\mathcal{D}|}{|P|} \geq 1$ and is loose by a factor of 2. For the second part we can argue

$$|R \cap P|\frac{|\mathcal{D}|}{|P|} - |R \cap \mathcal{D}| \leq |R \cap P|\frac{1}{1-\frac{\varepsilon}{2}} - |R \cap \mathcal{D}| \leq |R \cap \mathcal{D}|\frac{1}{1-\frac{\varepsilon}{2}} - |R \cap \mathcal{D}|$$
$$= \frac{\frac{\varepsilon}{2}}{1-\frac{\varepsilon}{2}}|R \cap \mathcal{D}| \leq \varepsilon|R \cap \mathcal{D}| \leq \varepsilon|\mathcal{D}|.$$

For terrain domains $\mathcal{D}$ defined with a base B and a height function $h : B \to \mathbb{R}$, if B is polygonal we can decompose it into polygonal pieces, otherwise we can approximate it with constant-size polygonal pieces according to Lemma 2. Then, similarly, if h is polygonal we can approximate the components invoking Corollary 4; however, if it is smooth, then we can approximate each piece according to Lemma 2.

We can improve further upon this approach using a stretched version of the Van der Corput Set and dependent on specific properties of the terrain. Consider the case where B is a rectangle with diameter $d_{\mathcal{D}}$ and h is C^2 continuous with minimum value $z_{\mathcal{D}}^-$ and where the largest eigenvalue of its Hessian is $\lambda_{\mathcal{D}}$. For such a terrain $\mathcal{D}$, interesting ranges $\mathcal{R}_2 \times \mathbb{R}$ are generalized cylinders where the first 2 dimensions are an axis-parallel rectangle and the third dimension is unbounded. We can state the following result (proved in the full version).

Theorem 4. *For a domain* $\mathcal{D}$ *with rectangular base* $B \subset \mathbb{R}^2$ *and with a* C^2*-continuous height function* $h : B \to \mathbb{R}$ *we can deterministically create a weighted* ε*-approximation of* $(\mathcal{D}, \mathcal{R}_2 \times \mathbb{R})$ *of size* $O\left(\left(\frac{\lambda_{\mathcal{D}} d_{\mathcal{D}}^2}{z_{\mathcal{D}}^- \varepsilon}\right)\left(\frac{1}{\varepsilon}\log^4\frac{1}{\varepsilon}\text{polylog}(\log\frac{1}{\varepsilon})\right)\right)$*. We reduce the size to* $O(\frac{1}{\varepsilon}\log^4\frac{1}{\varepsilon}\text{polylog}(\log\frac{1}{\varepsilon}))$ *in time* $O\left(\left(\frac{\lambda_{\mathcal{D}} d_{\mathcal{D}}^2}{z_{\mathcal{D}}^-}\right)\frac{1}{\varepsilon^5}\text{ polylog}\frac{1}{\varepsilon}\right)$.

This generalizes in a straightforward way for $B \in \mathbb{R}^d$. Similar results are possible when B is not rectangular or when B is not even piecewise-linear. The techniques of Section 4 are necessary if Q_k is used instead of $\mathcal{R}_2$, and are slower by a factor $O(\frac{1}{\varepsilon})$.

6 Applications

Creating smaller ε-approximations improves several existing algorithms.

Biosurveillance. Let M and B be two points sets in $\mathbb{R}^2$. An important anomaly detection problem for biosurveillance [14,1] reduces to finding a range (from some family of ranges such as $\mathcal{R}_2$) that maximizes a statistical discrepancy function on M and B, such as $d_P(m_R, b_R) = m_R \ln \frac{m_R}{b_R} + (1 - m_R) \ln \frac{1-m_R}{1-b_R}$, where $m_R = |R \cap M|/|M|$ and $b_R = |R \cap B|/|B|$. Using the results in this paper we can prove the following:

Theorem 5. *Let $|M \cup B| = n$. A range $R \in \mathcal{R}^2$ such that $|d_P(m_R, b_R) - \max_{r \in \mathcal{R}_2} d_P(m_r, b_r)| \leq \varepsilon$ can be deterministically found in $O(n\frac{1}{\varepsilon^3}\text{polylog}(\log \frac{1}{\varepsilon}) + \frac{1}{\varepsilon^4}\text{polylog}(\log \frac{1}{\varepsilon}))$ time.*

A range $R \in \mathcal{R}^2$ such that $|d_P(m_R, b_R) - \max_{r \in \mathcal{R}_2} d_P(m_r, b_r)| \leq \varepsilon + \delta$ can be deterministically found in $O(n\frac{1}{\varepsilon^3}\text{polylog}(\log \frac{1}{\varepsilon}) + \frac{1}{\delta}\frac{1}{\varepsilon^2}\text{polylog}(\log \frac{1}{\varepsilon}))$ time.

This can be generalized to when M and B are terrain domains. This case arises, for example, when each point is replaced with a probability distribution.

Sensor Networks. Let $\mathcal{D}$ be a set of points describing the location of sensors. If $P \subseteq \mathcal{D}$ is an *ε-sentinel* of $(\mathcal{D}, \mathcal{A})$, then for all $R \in \mathcal{A}$ (1) if $|R \cap \mathcal{D}| \geq \varepsilon|\mathcal{D}|$ then $|R \cap P| \geq \varepsilon\frac{3}{4}|P|$, and (2) if $|R \cap P| \geq \varepsilon\frac{3}{4}|P|$ then $|R \cap \mathcal{D}| \geq \frac{\varepsilon|\mathcal{D}|}{2}$. Previous work constructs ε-sentinels for half spaces [21] of size $O(\frac{1}{\varepsilon})$ and in expected time $O(\frac{n}{\varepsilon}\log n)$ or for any $\mathcal{A}$ with bounded VC-dimension v [11] of size $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ and in time $O(n\frac{1}{\varepsilon^{2v}}\log^v\frac{1}{\varepsilon})$. Noting that an $\frac{\varepsilon}{4}$-approximation can be used as an ε-sentinel, we can state the following.

Theorem 6. *For a discrete point set $\mathcal{D}$ of size n, we can compute ε-sentinels for $(\mathcal{D}, \mathcal{Q}_k)$ of size $O(\frac{1}{\varepsilon}\log^{2k}\frac{1}{\varepsilon}\text{polylog}(\log\frac{1}{\varepsilon}))$ in time $O(n\frac{1}{\varepsilon^3}\text{polylog}(\log\frac{1}{\varepsilon}))$.*

Furthermore, we can create $O(n\varepsilon/\log^{2k}\frac{1}{\varepsilon})$ disjoint sets of ε-sentinels in $O(n\frac{1}{\varepsilon^3}\log(n\varepsilon)\text{polylog}(\log\frac{1}{\varepsilon}))$ total time.

We can extend this result to place an ε-sentinel to cover a polygonal domain $\mathcal{D}$ as well. Details and further results are in the full version.

Acknowledgments. I would like to thank Pankaj Agarwal for many helpful discussions including finding a bug in an earlier version of the proof of Lemma 1, Shashidhara Ganjugunte, Hai Yu, Yuriy Mileyko, and Esther Ezra for a careful proofreading, Jirka Matoušek for useful pointers, Subhash Suri for posing a related problem, and Don Rose for discussions on improving the Beck-Fiala Theorem.

References

1. Agarwal, D., McGregor, A., Phillips, J.M., Venkatasubramanian, S., Zhu, Z.: Spatial scan statistics: Approximations and performance study. In: Proceedings 12th ACM SIGKDD Knowledge Discovery & Data Mining, pp. 24–33 (2006)

2. Alexander, R.: Principles of a new method in the study of irregularities of distribution. Inventiones Mathematicae 103, 279–296 (1991)
3. Beck, J.: Balanced two-coloring of finite sets in the square I. Combinatorica 1, 327–335 (1981)
4. Beck, J.: Roth's estimate on the discrepancy of integer sequences is nearly sharp. Combinatorica 1, 319–325 (1981)
5. Beck, J.: Irregularities of distribution I. Acta Mathematics 159, 1–49 (1987)
6. Beck, J.: Probabilistic diophantine approximation, I Kronecker sequences. Annals of Mathematics 140, 451–502 (1994)
7. Beck, J., Chen, W.: Irregularities of Distribution. Cambridge University Press, Cambridge (1987)
8. Beck, J., Fiala, T.: "integer-making" theorems. Disc. App. Math. 3, 1–8 (1981)
9. Chazelle, B.: The Discrepancy Method. Cambridge University Press, Cambridge (2000)
10. Chazelle, B., Matousek, J.: On linear-time deterministic algorithms for optimization problems in fixed dimensions. Journal of Algorithms 21, 579–597 (1996)
11. Gandhi, S., Suri, S., Welzl, E.: Catching elephants with mice: Sparse sampling for monitoring sensor networks. In: Proceedings 5th Embedded Networked Sensor Systems, pp. 261–274 (2007)
12. Halton, J.H.: On the efficiency of certain quasi-random sequences of points in evaluating multidimensional integrals. Numerical Mathematics 2, 84–90 (1960)
13. Hammersly, J.M.: Monte Carlo methods for solving multivariable problems. Annals of New York Acadamy of Science 86, 844–874 (1960)
14. Kulldorff, M.: A spatial scan statistic. Comm. in Stat.: T&M 26, 1481–1496 (1997)
15. Matoušek, J.: Approximations and optimal geometric divide-and-conquer. In: Proceedings 23rd Symposium on Theory of Computing, pp. 505–511 (1991)
16. Matoušek, J.: Tight upper bounds for the discrepancy of halfspaces. Discrete and Computational Geometry 13, 593–601 (1995)
17. Matoušek, J.: Geometric Discrepancy. Springer, Heidelberg (1999)
18. Matoušek, J.: On the discrepancy for boxes and polytopes. Monatsh. Math. 127, 325–336 (1999)
19. Matoušek, J., Welzl, E., Wernisch, L.: Discrepancy and approximations for bounded VC-dimension. Combinatorica 13, 455–466 (1993)
20. Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods. SIAM, Philadelphia (1992)
21. Shrivastava, N., Suri, S., Tóth, C.D.: Detecting cuts in sensor networks. ACM Transactions on Sensor Networks 4(10) (2008)
22. Skriganov, M.: Lattices in algebraic number fields and uniform distributions modulo 1. Leningrad Mathematics Journal 1, 535–558 (1990)
23. Skriganov, M.: Constructions of uniform distributions in terms of geometry of numbers. St. Petersburg Mathematics Journal 6, 635–664 (1995)
24. Skriganov, M.: Ergodic theory on $SL(n)$, diophantine approximations and anomalies in the lattice point problem. Inventiones Mathematicae 132, 1–72 (1998)
25. Srinivasan, A.: Improving the discrepancy bound for sparse matrices: Better approximations for sparse lattice approximation problems. In: Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 692–701 (1997)
26. Suri, S., Tóth, C.D., Zhou, Y.: Range counting over multidimensional data streams. In: Proceedings 20th Symposium on Computational Geometry, pp. 160–169 (2004)
27. van der Corput, J.G.: Verteilungsfunktionen I. Aka. Wet. Ams. 38, 813–821 (1935)
28. Vapnik, V., Chervonenkis, A.: On the uniform convergence of relative frequencies of events to their probabilities. Theory of Prob. and its Applic. 16, 264–280 (1971)

An Approximation Algorithm for Binary Searching in Trees

Eduardo Laber and Marco Molinaro

Informatics Department of PUC-Rio, Brazil
Address: Rua Marquês de São Vicente 225, RDC, 4º andar, CEP 22453-900, Rio de Janeiro - RJ, Brazil
{laber,mmolinaro}inf.puc-rio.br

Abstract. In this work we consider the problem of computing efficient strategies for searching in trees. As a generalization of the classical binary search for ordered lists, suppose one wishes to find a (unknown) specific node of a tree by asking queries to its arcs, where each query indicates the endpoint closer to the desired node. Given the likelihood of each node being the one searched, the objective is to compute a search strategy that minimizes the expected number of queries. Practical applications of this problem include file system synchronization and software testing. Here we present a linear time algorithm which is the first constant factor approximation for this problem. This represents a significant improvement over previous $O(\log n)$-approximation.

1 Introduction

Searching in ordered structures is a fundamental problem in theoretical computer science. In one of its most basic variants, the objective is to find a special element of a totally ordered set by making queries which iteratively narrow the possible locations of the desired element. This can be generalized to searching in more general structures which have only a partial order for their elements instead of a total order [1,2,3,4,5].

In this work, we focus on searching in structures that lay between totally ordered sets and the most general posets: we wish to efficiently locate a particular node in a tree. More formally, as input we are given a tree $T = (V, E)$ which has a 'hidden' *marked* node and a function $w : V \to \mathbb{R}$ that gives the likelihood of a node being the one marked. For example, T could be modeling a network with one defective unit. In order to discover which node of T is marked, we can perform *edge queries*: after querying the arc (i, j) of T (j being a child of i)[1], we receive an answer stating that either the marked node is a descendant[2] of j (called a **yes** answer) or that the marked node is not a descendant of j (called a **no** answer).

A search strategy is a procedure that decides the next query to be posed based on the outcome of previous queries. As an example, consider the strategy

[1] Henceforth, when we refer to the arc (i, j), j is a child of i.
[2] We consider that a node is a descendant of itself.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 459–471, 2008.

for searching the tree T of Figure 1.a represented by the decision tree D of Figure 1.b. A decision tree can be interpreted as a strategy in the following way: at each step we query the arc indicated by the node of D that we are currently located. In case of a **yes** answer, we move to the right child of the current node and we move to its left child otherwise. We proceed with these operations until the marked node is found. Let us assume that 4 is the marked node in Figure 1.a. We start at the root of D and query the arc $(3,4)$ of T, asking if the marked node is a descendant of node 4 in T. Since the answer is **yes**, we move to the right child of $(3,4)$ in D and we query the arc $(4,6)$ in T. In this case, the outcome of the query $(4,6)$ is **no** and then we move to node $(4,5)$ of D. By querying this node we conclude that the marked node of T is in indeed 4.

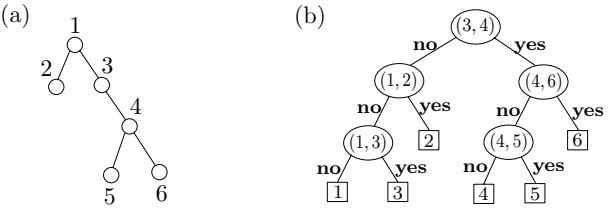


Fig. 1. (a) Tree T. (b) Example of a decision tree for T; Internal nodes correspond to arcs of T and leaves to nodes of T.

We define the average number of queries of a strategy $\mathcal{S}$ as $\sum_{v \in V} s_v w(v)$, where s_v is the number of queries needed to find the marked node when v is the marked node. Therefore, our optimization problem is to find the strategy with minimum expected number of queries. We make a more formal definition of a strategy by means of decision trees in Section 2.

Besides generalizing a fundamental problem in theoretical computer science, searching in posets (and in particular in trees) also has practical applications such as in file system synchronization and software testing [5]. We remark that although these applications were considered in the 'worst case' version of this problem, taking into account the likelihood that the elements are marked (for instance via code complexity measures in the former example) may lead to improved searches.

Statement of the results. Our main result is a linear time algorithm that provides the first constant factor approximation for the problem of binary searching in trees. The algorithm is based on the decomposition of the input tree into special paths. A search strategy is computed for each of these paths and then combined to form a strategy for searching the original tree. This decomposition is motivated by the fact that the problem of binary searching in paths is easily reduced to the well-solved problem of searching in ordered lists with access probabilities.

We shall notice that the complexity of this problem remains open, which contrasts with its 'worst case' version that is polynomially solvable [3,4,5]. In fact, the 'average case' version studied here is one more example of problems related to

searching and coding whose complexities are unknown [6,7]. Another interesting example is the Huffman coding problem with unequal cost letters [8,9].

Related work. Searching in totally ordered sets is a very well studied problem [10]. In addition, many variants have also been considered, such as when there is information about the likelihood of each element being the one marked [11], or where each query has a different fixed cost and the objective is to find a strategy with least total cost [12,13,14]. As a generalization of the latter, [15,16] considered the variant when the cost of each query depends on the queries posed previously.

The most general version of our problem when the input is a poset instead of a tree was first considered by Lipman and Abrahams [2]. Apart from introducing the problem, they present an optimized exponential time algorithm for solving it. In [17], Kosaraju et. al. present a greedy $O(\log n)$-approximation algorithm. In fact, their algorithm handles more general searches, see [18,19] for other more general results. To the best of our knowledge, this $O(\log n)$-approximation algorithm is the only available result, with theoretical approximation guarantee, for the average case version of searching in trees. Therefore, our constant approximation represents a significant advance for this problem.

The variant of our problem where the goal is to minimize the number of queries in the worst case for searching in trees, instead of minimizing the average case, was first considered by Ben-Asher et. al. [3]. They have shown that it can be solved in $O(n^4 \log^3 n)$ via dynamic programming. Recent advances [4,5] have reduced the time complexity to $O(n^3)$ and then $O(n)$. In contrast, the more general version where the input is a poset instead of a tree is NP-hard [1].

2 Preliminaries

First we need to fix some notation. For any tree T and node $j \in T$, we denote by T_j the subtree of T composed by all descendants of j. In addition, we denote the root of a tree T by $r(T)$. We also extend the set difference operation to trees: given trees $T^1 = (V^1, E^1)$ and $T^2 = (V^2, E^2)$, $T^1 - T^2$ is the forest of T^1 induced by the nodes $V^1 - V^2$. Furthermore, we extend the weight function to a subtree T' in a standard way $w(T') = \sum_{v \in T'} w(v)$. Finally, for any multiset W of non-negative real values, we define its entropy as $H(W) = -\sum_{w \in W} w \log \frac{w}{\sum_{w' \in W} w'}$.

Every query strategy for a tree T can be represented by a binary decision tree D such that a path of D indicates what queries should be made at each step. On a decision tree D, each internal node of D corresponds to a query for a different arc of T and each leaf of D corresponds to a different node of T (these correspondences shall become clear later). In addition, each internal node u of D satisfies a *search property* that can be described as follows. If u corresponds to a query for the arc (i, j), then: (i) u has exactly one right subtree and one left subtree; (ii) all nodes of the right subtree of u correspond to either a query for an arc in T_j or to a node in T_j; (iii) all nodes of the left subtree of u correspond to either a query for an arc in $T - T_j$ or to a node in $T - T_j$.

For any decision tree D, we use $u_{(i,j)}$ to denote the internal node of D which corresponds to a query for the arc (i,j) of T and u_v to denote the leaf of D which corresponds to the node v of T.

From the example presented in the introduction, we can infer an important property of the decision trees. Consider a tree T and a search strategy given by a decision tree D for T. If v is the marked node of T, the number of queries posed to find the marked node is the distance (in arcs) from the root of D to u_v.

For any decision tree D, we define $d(u,v,D)$ as the distance between nodes u and v in D (when the decision tree is clear from the context, we omit the last parameter of this function). Thus, the expected (with respect to w) number of queries it takes to find the marked node using the strategy given by the decision tree D, or simply the cost of D, is given by:

$$\text{cost}(D,w) = \sum_{v \in T} d(r(D), u_v, D)w(v)$$

Therefore, the problem of computing a search strategy for (T,w) which minimizes the expected number of queries can be recast as the problem of finding a decision tree for T with minimum cost, that is, that minimizes $\text{cost}(D,w)$ among all decision trees D for T. The cost of such minimum cost decision tree is denoted by $\text{OPT}(T,w)$.

Now we present properties of decision trees which are crucial for the analysis of the proposed algorithm. Consider a subtree T' of T; we say that a node u is a *representative* of T' in a decision tree D if the following conditions hold: (i) u is a node of D that corresponds to either an arc or a node of T' (ii) u is an ancestor of all other nodes of D which correspond to arcs or nodes of T'. The next lemma, whose proof is deferred to the full version of this paper, asserts the existence of a representative for each subtree of T.

Lemma 1. *Consider a tree T and a decision tree D for T. For each subtree T' of T, there is a unique node $u \in D$ which is the representative of T' in D.*

We denote the representative of T' (with respect to some decision tree) by $u(T')$.

The second property is given by the following lemma:

Lemma 2. *Consider a tree T, a weight function w and a decision tree D for T. Then for every subtree T' of T, $\sum_{v \in T'} d(u(T'), u_v, D)w(v) \geq OPT(T', w)$.*

The idea of the proof is to construct a decision tree D' for T' based on D in the following way: the nodes of D' are the nodes of D which correspond to the arcs and nodes of T'; there is an arc from u to v in D' iff u is the closest ancestor of v in D, among the nodes of D' (Figure 2).

By construction, the distance between two nodes u and v in D' is not greater than their distance in D. In addition, $u(T')$ is the root of D', so we have:

$$\text{cost}(D',w) = \sum_{v \in T'} d(u(T'), u_v, D')w(v) \leq \sum_{v \in T'} d(u(T'), u_v, D)w(v)$$

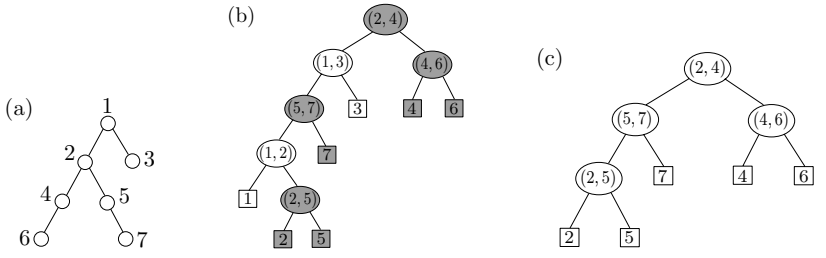


Fig. 2. (a) Tree T. (b) A decision tree D for T, with nodes corresponding to nodes and arcs of T_2 in gray. (c) Decision tree D' for T_2 constructed by connecting the nodes of D corresponding to nodes and arcs of T_2.

As one can prove that D' is a valid decision tree for T', we have that $\mathrm{OPT}(T', w) \leq \mathrm{cost}(D', w)$ and consequently $\sum_{v \in T'} d(u(T'), u_v, D)w(v) \geq \mathrm{OPT}(T', w)$.

3 An Algorithm for Binary Searching in Trees

In this section we present our algorithm for binary searching in trees. A crucial observation employed by the algorithm is that our problem of searching in trees can be efficiently solved when the input tree is a path. This is true because it can be easily reduced to the well-solved problem of searching a hidden marked element from a total order set U in a sorted list $L \subseteq U$ of elements where each element U has a given probability of being the marked one [11]. Due to this correspondence, an approximate strategy for searching in lists gives an approximation (with the same guarantee) for searching in path-like trees.

Motivated by this observation, the algorithm decomposes the input tree into special paths, finds decision trees for each of these paths (with modified weight functions) and combine them into a decision tree for the original tree. In our analysis, we obtain a lower bound on the optimal solution and an upper bound on the returned solution in terms of the costs of the decision trees for the paths. Thus, the approximation guarantee of the algorithm is basically a constant times the guarantee of the approximation used to compute the decision trees for the paths. Throughout the text, we present the execution and analysis of the algorithm over an instance (T, w), where T is rooted at node r.

For every node $u \in T$, we define the *cumulative weight* of u as the sum of the weights of its descendants, namely $w(T_u)$. A *heavy path* Q of T is defined recursively as follows: r belongs to Q; for every node u in Q, the non-leaf children of u with greatest cumulative weight also belongs to Q.

Let $Q = (q_1 \to \ldots \to q_{|Q|})$ be a heavy path of T. We define $\overline{T}_{q_i} = T_{q_i} - T_{q_{i+1}}$, for $i < |Q|$ and $\overline{T}_{q_{|Q|}} = T_{q_{|Q|}}$. In addition, we define $T^j_{q_i}$ as the jth heaviest maximal subtree rooted at a child of q_i not in Q (Figure 3). Finally, let n_i denote the number of children of q_i which do not belong to Q and define e^j_i as the arc of T connecting the node q_i to the subtree $T^j_{q_i}$.

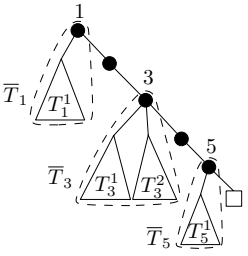


Fig. 3. Example of structures $\overline{T}_{q_i}$ and $T_{q_i}^j$. (a) Tree T with nodes of the heavy path in black.

Now we explain the high level structure of the solution returned by the algorithm. The main observation is that we can break the task of finding the marked node in three stages: first finding the node q_i of Q such that $\overline{T}_{q_i}$ contains the marked node, then querying the arcs $\{e_i^j\}_j$ to discover which tree $T_{q_i}^{j'}$ contains the marked node or that the marked node is q_i, and finally (if needed) locate the marked node in $T_{q_i}^{j'}$. The algorithm follows this reasoning: it computes a decision tree for the heavy path Q, then a decision tree for querying the arcs $\{e_i^j\}$ and recursively computes a decision tree for each tree $T_{q_i}^j$.

Now we present the algorithm itself, which consists of the following five steps:

(i) Find a heavy path Q of T and then for each $q_i \in Q$ define $w'(q_i) = w(\overline{T}_{q_i})/w(T)$.
(ii) Calculate a decision tree D_Q for the instance (Q, w') using the approximation algorithm presented in [11].
(iii) Calculate recursively a decision tree D_i^j for each instance $(T_{q_i}^j, w)$.
(iv) Build a decision tree D_i for each $\overline{T}_{q_i}$ as follows. The leftmost path of D_i consists of nodes corresponding to the arcs $e_i^1, \ldots, e_i^{n_i}$, with a node u_{q_i} appended at the end. In addition, for every j, D_i^j is the right child of the node corresponding to e_i^j in D_i (Figure 4).
(v) Construct the decision tree D for T by replacing the leaf u_{q_i} of D by D_i, for each $q_i \in Q$ (Figure 5).

It is not difficult to see that that the decision tree D computed by the algorithm is a valid decision tree for T.

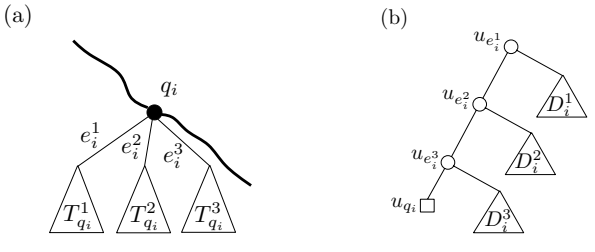


Fig. 4. Illustration of Step (iv) of the algorithm. (a) Tree T with heavy path in bold. (b) Decision tree D_i for $\overline{T}_i$.

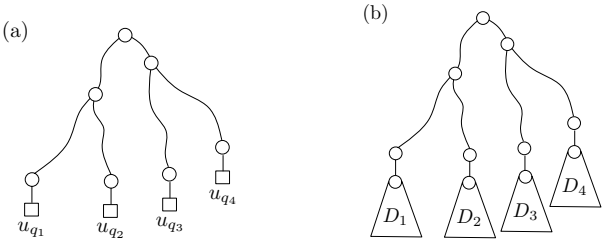


Fig. 5. Illustration of Step (v) of the algorithm. (a) Decision tree D_Q built at Step (ii). (b) Decision tree D constructed by replacing the leaves $\{u_{q_i}\}$ by the decision trees $\{D_i\}$.

3.1 Upper Bound

As the trees $\{T^j_{q_i}\}$ and Q form a partition of the nodes of T, we analyze the distance of the root of D to the nodes in each of these structures separately in order to upper bound the cost of D.

First, consider a tree $T^j_{q_i}$ and let x be a node in $T^j_{q_i}$. Noticing that u_x is a leaf of the tree D^j_i, the path from $r(D)$ to u_x in D must contain the node $r(D^j_i)$. Then, by the construction made in Step (iv), the path from $r(D)$ to u_x in D has the form $(r(D) \rightsquigarrow u_{e^1_i} \rightarrow u_{e^2_i} \rightarrow \ldots \rightarrow u_{e^j_i} \rightsquigarrow r(D^j_i) \rightsquigarrow u_x)$. Notice that the path $(r(D) \rightsquigarrow u_{e^1_i})$ in D is the same as the path $(r(D) \rightsquigarrow u_{q_i})$ in D_Q. In addition, the path from $r(D^j_i)$ to u_x is the same in D and in D^j_i. Employing the previous observations, we have that the length of the path $(r(D) \rightsquigarrow u_{e^1_i} \rightarrow u_{e^2_i} \rightarrow \ldots \rightarrow u_{e^j_i} \rightsquigarrow r(D^j_i) \rightsquigarrow u_x)$ is:

$$d(r(D), u_x, D) = d(r(D), u_{q_i}, D_Q) + j + d(r(D^j_i), u_x, D^j_i)$$

Now we consider a node $q_i \in Q$. Again due to the construction made in Step (iv) of the algorithm, it is not difficult to see that the path from $r(D)$ to u_{q_i} in D traverses the leftmost path of D_i, that is, this path has the form $(r(D) \rightsquigarrow u_{e^1_i} \rightarrow u_{e^2_i} \rightarrow \ldots \rightarrow u_{e^{n_i}_i} \rightarrow u_{q_i})$. Because the path $(r(D) \rightsquigarrow u_{e^1_i})$ in D is the same as the path $(r(D) \rightsquigarrow u_{q_i})$ in D_Q, it follows that the length of the path from the root of D to u_{q_i} is $d(r(D), u_{q_i}, D_Q) + n_i$.

Weighting distance to reach nodes in $\{T^j_{q_i}\}$ and in Q, we find the cost of D:

$$\begin{aligned}
\text{cost}(D, w) &= \sum_{q_i \in Q} d(r(D), u_{q_i}, D_Q) w(\overline{T}_{q_i}) + \sum_{q_i \in Q} \sum_j j \cdot w(T^j_{q_i}) \\
&\quad + \sum_{q_i \in Q} \sum_j \sum_{x \in T^j_{q_i}} d(r(D^j_i), u_x, D^j_i) w(x) + \sum_{q_i \in Q} n_i w(q_i) \\
&= w(T) \cdot \text{cost}(D_Q, w') + \sum_{q_i \in Q} \sum_j j \cdot w(T^j_{q_i}) \\
&\quad + \sum_{q_i \in Q} \sum_j \text{cost}(D^j_i, w) + \sum_{q_i \in Q} n_i w(q_i)
\end{aligned}$$

Now all we need is to upper bound the first term of the previous equality. Notice that $\text{cost}(D_Q, w')$ is exactly the cost of the approximation computed at Step (ii) of the algorithm. As mentioned previously, in this step we use the result from [11] which guarantees an upper bound of $H(\{w'(q_i)\}) + 2$ for $\text{cost}(D_Q, w')$. Substituting this bound on the last displayed equality and observing that $w(T) \cdot H(\{w'(q_i)\}) = H(\{w(\overline{T}_{q_i})\})$, we have:

$$\text{cost}(D, w) \leq H(\{w(\overline{T}_{q_i})\}) + 2 \cdot w(T) + \sum_{q_i \in Q} \sum_j j \cdot w(T^j_{q_i}) + \sum_{q_i \in Q} \sum_j \text{cost}(D^j_i, w) + \sum_{q_i \in Q} n_i w(q_i) \quad (1)$$

3.2 Entropy Lower Bound

In this section we present a lower bound on the cost of an optimal decision tree for (T, w). Hence, let D^* be a minimum cost decision tree for (T, w), and let r^* be the root of D^*.

Consider a tree $T^j_{q_i}$ and let x be a node in $T^j_{q_i}$. By definition, the representative of $T^j_{q_i}$ in D^* (the node $u(T^j_{q_i})$) is an ancestor of the node u_x in D^*. Notice that the representative of $T^j_{q_i}$ is a node in D^* that corresponds to some arc (i', j') of $T^j_{q_i}$ (that is $u(T^j_{q_i}) = u_{(i',j')}$) and that (i', j') is also an arc of $\overline{T}_{q_i}$. Therefore, the definition of representative again implies that $u(T^j_{q_i})$ is a descendant of $u(\overline{T}^j_{q_i})$. Combining the previous observations, we have that the path in D^* from r^* to u_x has the form $(r^* \rightsquigarrow u(\overline{T}_{q_i}) \rightsquigarrow u(T^j_{q_i}) \rightsquigarrow u_x)$.

Now consider a node $q_i \in Q$; again by the definition of representative, the path from r^* to u_{q_i} can be written as $(r^* \rightsquigarrow u(\overline{T}_{q_i}) \rightsquigarrow u_{q_i})$. Adding the weighted paths for nodes in $\{T^j_{q_i}\}$ and in Q, we can write the cost of D^* as:

$$\text{OPT}(T, w) = \sum_{q_i} d(r^*, u(\overline{T}_{q_i}))w(\overline{T}_{q_i}) + \sum_{q_i} \sum_j d(u(\overline{T}_{q_i}), u(T^j_{q_i}))w(T^j_{q_i}) + \sum_{q_i} \sum_j \sum_{x \in T^j_{q_i}} d(u(T^j_{q_i}), u_x)w(x) + \sum_{q_i} d(u(\overline{T}_{q_i}), u_{q_i})w(q_i) \quad (2)$$

Now we lower bound each term of the last equation. The idea to analyze the first term is the following: it can be seen as the cost of the decision tree D^* under a cost function where the representative of $\overline{T}_{q_i}$, for every i, has weight $w(\overline{T}_{q_i})$ and all other nodes have weight zero. Since D^* is a binary tree, we can use Shannon's Coding Theorem to guarantee that the first term of (2) is lower bounded by $H(\{w(\overline{T}_{q_i})\})/\log 3 - w(T)$(formal proof in the full paper).

Now we bound the second term of (2). Fix a node $q_i \in Q$; consider two different trees $T^j_{q_i}$ and $T^{j'}_{q_i}$ such that $d(r^*, u(T^j_{q_i})) = d(r^*, u(T^{j'}_{q_i}))$. We claim that $u(T^j_{q_i})$ and $u(T^{j'}_{q_i})$ are siblings in D^*. Because their distances to r^* are the same, $u(T^j_{q_i})$ cannot be a neither a descendant nor an ancestor of $u(T^{j'}_{q_i})$. Therefore, they have a common ancestor, say $u_{(v,z)}$, and one of these node is in the right

subtree of $u_{(v,z)}$ and the other in the left subtree of $u_{(v,z)}$. It is not difficult to see that $u_{(v,z)}$ can only correspond to either (q_i, j) or (q_i, j'). Without loss of generality suppose the latter; then the right child of $u_{(v,z)}$ corresponds to an arc/node of $T^{j'}_{q_i}$, and therefore $u(T^{j'}_{q_i})$ must be this child. Due to their distance to r^*, it follows that $u(T^{j}_{q_i})$ must be the other child of $u_{(v,z)}$.

As a consequence, we have that for any level ℓ of D^*, there are at most two representatives of the trees $T^j_{q_i}$ located at ℓ. This together with the fact that $T^j_{q_i}$ is the jth heaviest tree rooted at a child of q_i guarantees that

$$\sum_{j=1}^{n_i} d(u(\overline{T}_{q_i}), u(T^j_{q_i}))w(T^j_{q_i}) \geq \sum_{j=1}^{n_i} \lfloor (j-1)/2 \rfloor w(T^j_{q_i}) \geq \sum_{j=1}^{n_i} \left(\frac{j}{2} - \frac{3}{2}\right) w(T^j_{q_i}) \quad (3)$$

and the last inequality gives a bound for the second term of (2).

Directly employing Lemma 2, we can lower bound the third term of (2) by $\sum_{q_i,j} \mathrm{OPT}(T^j_{q_i}, w)$.

Finally, for the fourth term we fix q_i and note that the path in D^* connecting $u(\overline{T}_{q_i})$ to u_{q_i} must contain the nodes corresponding to arcs $e^1_i, \ldots, e^{n_i}_i$ (otherwise when traversing D^* and reaching u_{q_i} we would not have enough knowledge to infer that q_i is the marked node, contradicting the validity of D^*). Applying this reasoning for each q_i, we conclude that last term of (2) is lower bounded by $\sum_{q_i} n_i \cdot w(q_i)$.

Therefore, applying the previous discussion to lower bound the terms of (2) we obtain that:

$$\mathrm{OPT}(T, w) \geq \frac{H(\{w(\overline{T}_{q_i})\})}{c} - \frac{5w(T)}{2} + \sum_{q_i,j} \frac{j \cdot w(T^j_{q_i})}{2} + \sum_{q_i,j} \mathrm{OPT}(T^j_{q_i}, w) + \sum_{q_i} n_i w(q_i) \quad (4)$$

3.3 Alternative Lower Bounds

When the value of the entropy $H(\{w(\overline{T}_{q_i})\})$ is large enough, it dominates the term $-5w(T)/2$ in inequality (4) and leads to a sufficiently strong lower bound. However, when this entropy assumes a small value we need to adopt a different strategy to devise an effective bound.

First alternative lower bound. In order to reduce the additive that appears in (4), we use almost the same derivation that leads from (2) to (4); however, we simply lower bounded the first summation of (2) by zero. This gives:

$$\mathrm{OPT}(T, w) \geq -\frac{3w(T)}{2} + \sum_{q_i,j} \frac{j \cdot w(T^j_{q_i})}{2} + \sum_{q_i,j} \mathrm{OPT}(T^j_{q_i}, w) + \sum_{q_i \in Q} n_i w(q_i) \quad (5)$$

Second alternative lower bound. Now we devise a lower bound without an additive factor; however it also does not contain the important term $\sum_{q_i \in Q} n_i \cdot w(q_i)$.

Consider a tree $T^j_{q_i}$ and a node $v \in T^j_{q_i}$. By the definition of representative, $u(T^j_{q_i})$ is an ancestor of u_v in D^*, thus $d(r^*, u_v, D^*) = d(r^*, u(T^j_{q_i}), D^*) + d(u(T^j_{q_i}), u_v, D^*)$. Because $\{T^j_{q_i}\}$ and Q form a partition of nodes of T, the cost $\mathrm{OPT}(T, w)$ can be written as:

$$\mathrm{OPT}(T,w) = \sum_{q_i,j} d(r^*, u(T^j_{q_i}))w(T^j_{q_i}) + \sum_{q_i \in Q} d(r^*, u_{q_i})w(q_i) + \sum_{q_i,j} \sum_{v \in T^j_{q_i}} d(u(T^j_{q_i}), u_v)w(v)$$

First, as the trees $\{T^j_{q_{|Q|}}\}$ do not contain any arcs, $\{u(T^j_{q_{|Q|}})\}$ and $\{u_{q_i}\}$ cannot be the root of D^*. Therefore, at most one distance $d(r^*, u(T^j_{q_i}))$ (for $q_i \neq q_{|Q|}$) of the first two summations of the previous inequality can be equal to zero, and all others must have value of at least one. But the construction of the heavy path guarantees that for $q_i \neq q_{|Q|}$ the weight of each of the trees $\{T^j_{q_i}\}$ is at most $w(T)/2$. As a consequence, the first two summations of the inequality can be lower bounded by $w(T)/2$. Combining this fact with a lower bound for the last summation provided by Lemma 2 (with $T' = T^j_{q_i}$) we have:

$$\mathrm{OPT}(T,w) \geq \frac{w(T)}{2} + \sum_{q_i,j} \mathrm{OPT}(T^j_{q_i}, w) \tag{6}$$

3.4 Approximation Guarantee

We proceed by induction over the number of nodes of T, where the base case is the trivial one when T has only one node. Notice that because each tree $T^j_{q_i}$ is properly contained in T, the inductive hypothesis asserts that D^j_i is an approximate decision tree for $T^j_{q_i}$, namely $\mathrm{cost}(D^j_i, w) \leq \alpha \mathrm{OPT}(T^j_{q_i}, w)$ for some constant α. The analysis needs to be carried out in two different cases depending on the value of the entropy (henceforth we use H as a shorthand for $H(\{w(\overline{T}_{q_i})\})$.

Case 1: $H/\log 3 \geq 3w(T)$. Applying this bound to inequality (4) we have:

$$\mathrm{OPT}(T,w) \geq \frac{H}{6\log 3} + \sum_{q_i,j} \frac{j \cdot w(T^j_{q_i})}{2} + \sum_{q_i,j} \mathrm{OPT}(T^j_{q_i}, w) + \sum_{q_i} n_i w(q_i) \tag{7}$$

Employing the entropy hypothesis to the first term of inequality (1) and using the inductive hypothesis we have:

$$\mathrm{cost}(D,w) \leq \frac{H(3\log 3 + 2)}{3\log 3} + \sum_{q_i,j} \left(j \cdot w(T^j_{q_i})\right) + \sum_{q_i,j} \alpha \mathrm{OPT}(T^j_{q_i}, w) + \sum_{q_i} n_i w(q_i)$$

Setting $\alpha \geq 2(3\log 3 + 2)$ it follows from the previous inequality and from inequality (7) that $\mathrm{cost}(D, w) \leq \alpha \mathrm{OPT}(T, w)$.

Case 2: $H/\log 3 \leq 3w(T)$. Applying the entropy hypothesis and the inductive hypothesis to inequality (1) we have:

$$\text{cost}(D,w) \leq (3\log 3+2)w(T) + \sum_{q_i,j}\left(j \cdot w(T^j_{q_i})\right) + \sum_{q_i,j} \alpha \text{OPT}(T^j_{q_i},w) + \sum_{q_i} n_i w(q_i)$$

Adding $(\alpha-1)$ times inequality (6) to inequality (5) we have:

$$\alpha\text{OPT}(T,w) \geq \frac{(\alpha-4)w(T)}{2} + \sum_{q_i,j}\frac{j\cdot w(T^j_{q_i})}{2} + \sum_{q_i,j}\alpha\text{OPT}(D^j_i,w) + \sum_{q_i} n_i w(q_i)$$

Setting $\alpha \geq 2(3\log 3+2)+4$ we have that $\text{cost}(T,w) \leq \alpha\text{OPT}(T,w)$. Therefore, the inductive step holds for both cases when $\alpha \geq 2(3\log 3+2)+4$.

3.5 Efficient Implementation

Notice that in order to implement the Step (iv) of the algorithm, we need to sort the trees $\{T^j_{q_i}\}$. As a heavy path decomposition can be computed in linear time [6], it is easy to see that all steps of the steps of the algorithm, besides the sorting at Step (iv), can be implemented in linear time. Hence we resort to an 'approximate sorting' to provide a linear time implementation of algorithm.

Fix a node q_i in Q. Without loss of generality, we assume there are more than three trees $\{T^j_{q_i}\}$ (that is $n_i > 3$), otherwise we could sort them in constant time. In order to simplify the analysis, we use $w_j = w(T^j_{q_i})$ and let $W = \max_j\{w_j\}$. Then we have the sequence $WQ = (w_1, w_2, \ldots, w_{n_i})$, which is nondecreasing by the definition of the trees $\{T^j_{q_i}\}$. Now we partition WQ into blocks with similar weights: for $0 \leq j < n_i$, the j-block contains all elements of WQ with weights in the interval $[W/2^{j-1}, W/2^j)$ and the n_i-block will contain all elements with weights in $[W/2^{n_i-1}, 0]$. Notice that because WQ is ordered, it is the concatenation of these blocks. Defining s_j as the index of the first element of the j-block[3], we have that the for $j < n_i$ the j-block is the sequence $(w_{s_j}, w_{s_j+1}, \ldots, w_{s_{j+1}-1})$ and the n_i-block is the sequence $(w_{s_{n_i}}, w_{s_{n_i}+1}, \ldots, w_{n_i})$.

Now we say that WQ' is an *approximate sorting* of the weights $\{w_j\}$ if WQ' contains first all elements 1-block of WQ, then all elements of the 2-block of WQ and so on. The sequence WQ' can be thought as a permutation of WQ where only elements *on the same j-block* can be permuted. Defining $WQ' = (w_{\sigma(1)}, w_{\sigma(2)}, \ldots, w_{\sigma(n_i)})$, it follows that the subsequence $(w_{\sigma(s_j)}, w_{\sigma(s_j+1)}, \ldots, w_{\sigma(s_{j+1})})$ contains the same elements of the j-block of WQ. As there are only n_i blocks, we can find an approximate sorting in linear time using a bucketing strategy.

We claim that $\sum_{j=1}^{n_i} j \cdot w_{\sigma(j)} \leq 3\sum_{j=1}^{n_i} j \cdot w_j$. For every $0 \leq j < n_i$, it follows from the definition of j-block and the relationship between the elements of WQ' and the blocks of WQ that for every $0 \leq j < n_i$:

$$\sum_{k=s_j}^{s_{j+1}-1} k \cdot w_k \geq \frac{W}{2^j}\sum_{k=s_j}^{s_{j+1}-1} k = \frac{1}{2}\left(\frac{W}{2^{j-1}}\sum_{k=s_j}^{s_{j+1}-1} k\right) \geq \frac{1}{2}\sum_{k=s_j}^{s_{j+1}-1} k \cdot w_{\sigma(k)} \qquad (8)$$

[3] In order to avoid a heavier notation, we assume that each j-block is nonempty.

In addition, because for every $k \geq s_{n_i}$ we have $w_{\sigma(k)} \leq W/2^{n_i}$, the sum $\sum_{k=s_{n_i}}^{n_i} k \cdot w_{\sigma(k)}$ can be upper bounded by $n_i^2(W/2^{n_i})$. Therefore, for $n_i > 3$ this term can be upper bounded by W. Combining with the fact that $\sum_{k=1}^{n_i} k \cdot w_k \geq W$ we have that $\sum_{k=s_{n_i}}^{n_i} k \cdot w_{\sigma(k)} \leq \sum_{k=1}^{n_i} k \cdot w_k$.

Using the previous inequality together with inequality (8), we have:

$$\sum_{k=1}^{n_i} k \cdot w_{\sigma(k)} = \sum_{j=1}^{n_i-1} \sum_{k=s_j}^{s_{j+1}-1} k \cdot w_{\sigma(k)} + \sum_{k=s_{n_i}}^{n_i} k \cdot w_{\sigma(j)} \leq 2 \sum_{j=1}^{n_i-1} \sum_{k=s_j}^{s_{j+1}-1} k \cdot w_k + \sum_{k=1}^{n_i} k \cdot w_k \leq 3 \sum_{k=1}^{n_i} k \cdot w_k$$

Now suppose that our algorithm uses the approximate permutation σ to sort the trees $\{T_{q_i}^j\}$. It is easy to see this only impacts the second term of the right hand side of equation (1), with $\sum_j j \cdot w(T_{q_i}^j)$ being replaced by $\sum_j j \cdot w(T_{q_i}^{\sigma(j)})$. Recalling that $w_j = w(T_{q_i}^j)$, the previous argument implies that this approximate sorting introduces a multiplicative factor of three in second term of (1) and it is straight forward to check this does not alter the guarantee of the algorithm.

Theorem 1. *There is a linear time algorithm which provides a constant factor approximation for the problem of binary searching in trees.*

References

1. Carmo, R., Donadelli, J., Kohayakawa, Y., Laber, E.: Searching in random partially ordered sets. Theor. Comput. Sci. 321, 41–57 (2004)
2. Lipman, M., Abrahams, J.: Minimum average cost testing for partially ordered components. IEEE Transactions on Information Theory 41, 287–291 (1995)
3. Ben-Asher, Y., Farchi, E., Newman, I.: Optimal search in trees. SIAM Journal on Computing 28 (1999)
4. Onak, K., Parys, P.: Generalization of binary search: Searching in trees and forest-like partial orders. In: FOCS, pp. 379–388 (2006)
5. Mozes, S., Onak, K., Weimann, O.: Finding an Optimal Tree Searching Strategy in Linear Time. In: SODA (2008)
6. Douïeb, K., Langerman, S.: Near-entropy hotlink assignments. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 292–303. Springer, Heidelberg (2006)
7. Barkan, A., Kaplan, H.: Partial alphabetic trees. J. Algorithms 58, 81–103 (2006)
8. Karp, R.: Minimum-redundancy coding for the discrete noiseless channel. IRE Trans. Inform. Theory 7, 27–39 (1961)
9. Golin, M., Kenyon, C., Young, N.: Huffman coding with unequal letter costs. In: STOC (2002)
10. Knuth, D.: The art of computer programming, volume 3: sorting and searching. Addison Wesley Longman Publishing Co., Inc., Redwood City (1998)
11. de Prisco, R., de Santis, A.: On binary search trees. Inf. Process. Lett. 45, 249–253 (1993)
12. Knight, W.: Search in an ordered array having variable probe cost. SIAM J. Comput. 17, 1203–1214 (1988)

13. Laber, E., Milidiú, R., Pessoa, A.: Strategies for searching with different access costs. In: Nešetřil, J. (ed.) ESA 1999. LNCS, vol. 1643, pp. 236–247. Springer, Heidelberg (1999)
14. Laber, E., Milidiú, R., Pessoa, A.: On binary searching with non-uniform costs. In: SODA, pp. 855–864 (2001)
15. Navarro, G., Baeza-Yates, R., Barbosa, E., Ziviani, N., Cunto, W.: Binary searching with nonuniform costs and its application to text retrieval. Algorithmica 27, 145–169 (2000)
16. Szwarcfiter, J., Navarro, G., Baeza-Yates, R., de, S., Oliveira, J., Cunto, W., Ziviani, N.: Optimal binary search trees with costs depending on the access paths. Theor. Comput. Sci. 290, 1799–1814 (2003)
17. Kosaraju, R., Przytycka, T., Borgstrom, R.: On an optimal split tree problem. In: Dehne, F., Gupta, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663. Springer, Heidelberg (1999)
18. Laber, E., Nogueira, L.: On the hardness of the minimum height decision tree problem. Discrete Applied Mathematics 144, 209–212 (2004)
19. Chakaravarthy, V., Pandit, V., Roy, S., Awasthi, P., Mohania, M.: Decision trees for entity identification: approximation algorithms and hardness results. In: PODS, pp. 53–62 (2007)

Algorithms for 2-Route Cut Problems

Chandra Chekuri[1,⋆] and Sanjeev Khanna[2,⋆⋆]

[1] Dept. of Computer Science, University of Illinois, Urbana, IL 61801
chekuri@cs.uiuc.edu
[2] Dept. of CIS, Univ. of Pennsylvania, Philadelphia, PA 19104
sanjeev@cis.upenn.edu

Abstract. In this paper we study approximation algorithms for *multi-route cut* problems in undirected graphs. In these problems the goal is to find a minimum cost set of edges to be removed from a given graph such that the edge-connectivity (or node-connectivity) between certain pairs of nodes is reduced below a given threshold K. In the usual cut problems the edge connectivity is required to be reduced below 1 (i.e. disconnected). We consider the case of $K = 2$ and obtain poly-logarithmic approximation algorithms for fundamental cut problems including single-source, multiway-cut, multicut, and sparsest cut. These cut problems are dual to multi-route flows that are of interest in fault-tolerant networks flows. Our results show that the flow-cut gap between 2-route cuts and 2-route flows is poly-logarithmic in undirected graphs with arbitrary capacities. 2-route cuts are also closely related to well-studied feedback problems and we obtain results on some new variants. Multi-route cuts pose interesting algorithmic challenges. The new techniques developed here are of independent technical interest, and may have applications to other cut and partitioning problems.

1 Introduction

We study *multi-route cut* problems in undirected graphs which generalize well-known and standard (1-route) cut problems. Consider a single pair of nodes s, t in an edge-weighted $G = (V, E)$ with c_e denoting the weight/cost of edge $e \in E$. Then a K-route cut for s, t is a subset $E' \subseteq E$ such that removing E' would leave at most $K - 1$ edge-disjoint paths between s and t. In other words s and t are not K-edge-connected in $G[E \setminus E']$; we say that s, t are K-*separated* by E' or that E' is a K-route cut. A regular s-t cut is a 1-route cut. We are interested in finding a minimum weight K-route cuts for $K > 1$. In this paper we focus on $K = 2$ as the first non-trivial case in this class of problems. We consider several natural and well-studied cut problems such as the single-source multiple-sink cut, multiway cut, and multicut in this new setting. We also consider two

⋆ Supported in part by NSF grants CCF-0728782 and CNS-0721899, and US-Israel BSF grant 2002276.
⋆⋆ Supported in part by a Guggenheim Fellowship, by NSF Award CCF-0635084, and by US-Israel BSF grant 2002276.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 472–484, 2008.

orthogonal generalizations in the definition of the cut: the first is to find a set of edges that reduces the *node*-connectivity, and the second is to find node-weighted cuts instead of edge-cuts. We start by motivating the study of the multi-route cut problems.

Our primary inspiration comes from the fact that multi-route cuts are the natural dual problems to multi-route *flows*. To describe multi-route flows it is useful to consider the standard (1-route) flow between s and t in a graph G as an assignment of non-negative numbers to the set of all the *paths* $\mathcal{P}$ from s to t. Let $\mathcal{P}_K$ denote the set of all tuples $(p_1, p_2, \ldots, p_K)$ where $p_i \in \mathcal{P}$ and the paths $p_1, p_2, \ldots, p_K$ are *edge-disjoint*. A tuple $(p_1, p_2, \ldots, p_K)$ of this form is called an *elementary K-flow*. A K-route flow between s, t is simply an assignment of non-negative numbers to elementary K-flows. Multi-route flows arise in several applications where fault-tolerance to edge (or node) failures is relevant [20,1,5,6,2], and are implicitly used in LP-relaxations for connectivity problems such as the survivable network design problem (SNDP). Kishimoto [20], building on some earlier work, introduced multi-route flows. For any K, the maximum K-route flow between s and t can be computed via the ellipsoid method for linear programming. Kishimoto [20] gave an algorithm that requires solving at most K regular maximum flows. Aggarwal and Orlin [1] simplified the ideas and analysis in [20] (see also [3]), and showed several interesting applications of multi-route flows to combinatorial problems. Just as a regular s-t minimum-cut upper bounds the s-t maximum flow, a minimum K-route cut bounds the maximum K-route flow. However, for $K > 1$, the flow-cut equivalence no longer holds even for a single pair s, t. Thus it is of interest to study the two following questions. What is the gap between the maximum K-route flow and the minimum K-route cut for a single pair as well as more general multi-pair settings? What is the complexity of finding minimum K-route cuts? Some of these questions have been raised in the recent work Bruhn *et al.* [7] who studied multi-route flows and cuts in the single source setting for unit-capacity graphs and suggested several open problems, including some of which we study in this paper.

Another motivation for multi-route cuts, and in particular for 2-route-cuts, comes from feedback problems in undirected graphs. The input to a feedback problem is a graph with either edge or node weights, and a set of (simple) cycles $\mathcal{C}$ that is usually specified implicitly. The goal is to remove edges (or nodes) of minimum-cost such that the remaining graph has no cycle from $\mathcal{C}$. These problems have received considerable attention in the past and have several applications. In the subset feedback problem, a set of terminals $S \subseteq V$ defines $\mathcal{C}$; a cycle $C \in \mathcal{C}$ iff C contains a terminal. An $O(1)$-approximation is known for this problem even in the node-weighted setting [14]. We observe that a 2-route cut corresponds to removing edges (or nodes) such that the remaining graph does not have 2-edge-disjoint (or 2-node-disjoint) paths between specified terminal pairs. In undirected graphs, a 2-route cuts for node-disjoint paths give rise to new and interesting feedback problems, e.g., the 2-route multiway-cut problem corresponds to the feedback problem where $\mathcal{C}$ is the set of cycles that contain at

least *two* terminals from a given terminal set $S \subseteq V$. Note that the edge-disjoint 2-route cuts require more than simple cycles to be removed (see Fig 2).

Finally, we remark that 2-route cuts, and more generally K-route cuts, are algorithmically challenging. Many of the techniques that have been developed for the regular cut problems cannot be applied directly. We develop some new techniques in this paper and we believe that a proper and full understanding of these problems would require new algorithmic ideas of broader applicability. We discuss the technical challenges in more detail after we describe our results.

2-route Cut Problems: In this paper we restrict our attention to $K = 2$ and study several natural 2-route cut problems in undirected graphs. In particular we consider the following problems in undirected graph $G = (V, E)$.

- Single-source multiple-sink. Given a source s and multiple sinks $t_1, t_2, \ldots, t_h$, find a minimum weight cut to 2-separate s from t_i for $1 \leq i \leq h$.
- Multiway-cut. Given h terminals $s_1, s_2, \ldots, s_h$, find a minimum weight cut that 2-separates s_i from s_j for each $i \neq j$.
- Multicut. Given h node pairs $s_1t_1, s_2t_2, \ldots, s_ht_h$, find a minimum weight cut that 2-separates s_i from t_i for $1 \leq i \leq h$.
- Sparsest cut. Given h node pairs $s_1t_1, s_2t_2, \ldots, s_ht_h$, find a cut that minimizes the ratio of the cut cost to the number of pairs that are 2-separated.

Results: We first describe our results for the case of edge-disjoint 2-route cut problems in edge-weighted graphs. We obtain poly-logarithmic approximations for the four 2-route cut problems that we mentioned above. In particular, we achieve approximation ratios of $O(\log n)$, $O(\log n \log h)$ and $O(\log^2 n \log h)$ for the single-source, multiway-cut and multicut problems respectively. The multicut result can be used to obtain an $O(\log^2 n \log^2 h)$ approximation for the sparsest cut problem using standard ideas. Our results are obtained via a natural LP relaxation whose dual corresponds to a maximum 2-route flow problem. Thus we obtain poly-logarithmic upper bounds on 2-route flow-cut gaps.

Our techniques can be adapted to get approximation guarantees similar to the corresponding edge-disjoint version for the single-source multiple-sink and the multiway cut problem in the *node-disjoint* setting; however, the multicut problem introduces some additional technical difficulties and we leave this as a direction for future work. All our results, including those for node-disjoint paths version, extend to *node-weighted* problems with identical performance guarantees.

It is easy to show that the 2-route cut problems considered here are at least as hard to approximate as their 1-route counterparts (modulo constant factors). Moreover, it is easy to show that the LP integrality gap is $\Omega(\log n)$ for the 2-route multiway-cut problem even when the terminals span the vertex set. In the rest of the paper, we focus on the 2-route edge-disjoint cut problems in edge-weighted graphs. Most proofs are omitted due to space limitations; please see the author web pages for a longer version of the paper.

Algorithmic ideas: We illustrate the new algorithmic ideas needed to address 2-route problems by considering the simplest setting, namely the s, t 2-route cut

problem. An optimal solution to this problem may be obtained as follows: *guess* an edge e, and output the edges in a minimum s,t cut in $G-e$. We leave the proof of the optimality of this procedure as a useful exercise to the reader. For general K, one needs to guess $K-1$ edges. Thus, the single pair problem for any fixed K can be solved optimally even in directed graphs. Now consider the single-source multiple-sink problem where we wish to 2-separate s from $t_1, t_2, \ldots, t_h$. When $K = 1$ this problem can be reduced to the single pair case by simply connecting $t_1, t_2, \ldots, t_k$ to a super-sink t with infinite cost edges. However, for $K \geq 2$ this reduction does not work. In fact, the 2-route single-source multiple-sink problem is at least as hard as the regular multiway-cut problem which is known to be APX-hard. We therefore resort to approximation algorithms and consider the natural LP relaxation for these problems. The relaxation assigns lengths to the edges such that for each pair $s_i t_i$ that needs to be separated, the minimum length of 2-edge disjoint paths between them is at least 1. The main challenge is to round a fractional solution to this relaxation.

One of the difficulties with rounding for K-route-cuts when $K \geq 2$ is that the cut does not disconnect the graph into connected components with the source and sink of a terminal pair in different components. In fact, for $K \geq 2$, if the original graph G is connected, then deleting the edges in any *minimal* solution, would yield a residual graph that is still connected. Thus the standard technique of using a "ball-growing" procedure to identify the set of edges to be deleted, is not directly applicable in this setting. To concretely illustrate the difficulty of adapting classical ball-growing techniques to our setting, consider the simple example shown in Figure 1. It shows a feasible fractional solution for a 2-route cut separating s from t. The shortest distance from s to t in this example is 0, while the two edge-disjoint path distance from s to t is 1. Thus any ball-growing procedure that uses shortest path distances, will place both s and t inside the same ball, no matter how small the radius of the ball. On the other hand, in this example, any ball grown from s w.r.t. 2 edge-disjoint path distance contains only vertex s if the radius is less than 1, and the entire graph if the radius is 1. In case of former, the only edges leaving the ball have a fractional length of 0. Thus these edges cannot be deleted in any solution with a finite performance guarantee. In case of latter, we do not get to separate s from t. This simple example also highlights that the two-edge-disjoint-path distance measure behaves quite differently from the usual shortest path distance metric. It appears that the standard region growing algorithms [21,15] and embedding methods are difficulty to adapt to the 2-route setting.

To overcome these difficulties, we introduce a novel randomized rounding technique that allows us to reduce 2-route cut problems to 1-route problems (not in a one-to-one correspondence). An interesting aspect of our reduction is that it maps feasible fractional solutions to 2-route cut problems to feasible fractional solutions to appropriate 1-route cut problems. We can then use standard rounding algorithms for the 1-route cut problems to output a feasible solution for the initial 2-route cut problem. The indirect nature of this rounding process creates technical difficulties in arguing feasibility of the final solution. We note that the

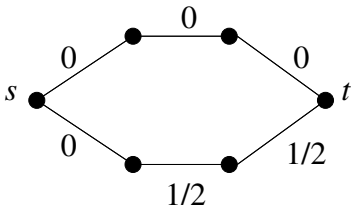


Fig. 1. Ball-growing w.r.t. 2 edge-disjoint paths distance

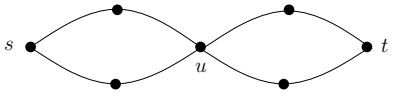


Fig. 2. There is no simple cycle between s and t but an edge has to be removed to 2-separate s from t in the edge-disjoint case

random scaling we use is non-standard and there does not appear to be a natural deterministic analogue (yet) of the procedure. We believe these new techniques are of independent technical interest, and may have applications to other cut and partitioning problems.

Related Work: Cut and flow problems are ubiquitous in combinatorial optimization and hence we do not discuss this well known area. As mentioned earlier, multi-route flows have been of interest since the work of Kishimoto and others [20]. Recently, Bruhn *et al.* [7] considered the gap between a maximum K-route flow and a maximum 1-route flow for the single-source multiple-sink problem. They showed that for *unit capacity* undirected graphs, the maximum 1-route flow is not more than $2(1 - 1/K)$ times the maximum K-route flow. In particular this implies that for any K, in unit-capacity graphs one can obtain a simple $2(K - 1)$ approximation for the single-source multi-sink K-route cut problem. We note that the unit-capacity problem is very different in nature from the problem with general capacities for $K \geq 2$. In a regular 1-route problem one can replace an edge with integer capacity c_e by c_e edges of unit capacity without changing the problem. However, for $K \geq 2$ this transformation does not preserve the flow! It is easy to construct examples where this transformation would increase the K-route flow by an unbounded amount. For instance, consider a network with two nodes s and t and two parallel edges, one with capacity 1 and other with capacity M for some integer $M \geq 1$. Then the maximum 2-route flow between s and t is 1. However, if we replace the capacity M edge by M parallel copies of unit capacity, the 2-route flow value increases to $\Omega(M)$.

2 Preliminaries

Given a (multi-)graph G, we will use n and m to denote the number of vertices and edges in G respectively. For convenience, we will assume that m is polynomially bounded in n. We let $c(e)$ denote the cost of an edge e. We say that a vertex s is *K-separated* from a vertex t if the maximum K-route flow from s to t is 0.

For any two nodes $s, t \in V$, let $\mathcal{P}_\ell^G(s,t)$ denote the set of all tuples $(p_1, p_2, \ldots, p_\ell)$ where each p_i is a path from s to t and p_i and p_j are edge-disjoint for $i \neq j$. Let $x : E \to \mathbb{R}^+$ be an assignment of non-negative weights to

the edges and for a path p let $x(p) = \sum_{e \in p} x(e)$. For two nodes $s, t \in V(G)$ we let $d_1^G(s,t;x)$ denote the length of a shortest path between s and t with respect to edge weights x in the graph G. In other words $d_1^G(s,t;x) = \min_{p \in \mathcal{P}_1^G(s,t)} x(p)$. We let $d_2^G(s,t;x) = \min_{(p,q) \in \mathcal{P}_2^G(s,t)} x(p) + x(q)$ denote the minimum length of two edge disjoint paths from s to t with respect to edge weights x. We define $d_\ell^G(s,t;x)$ for any integer ℓ in a similar fashion. Finally, for any vertex s we define $B_\ell^G(s,\theta;x) = \{v | d_\ell^G(s,v;x) \leq \theta\}$. We drop the superscript G when the graph is clear from the context.

LP Relaxation: We now describe an LP relaxation for our problems - this is a natural generalization of the $K = 1$ case and has been considered earlier. Since the multicut problem captures the other two problems in our study as special cases, it suffices to only give a formulation for multicut.

$$\begin{aligned} \min \sum_{e \in E} c(e)x(e) & \\ x(p) + x(q) \geq 1 & \qquad (p,q) \in \mathcal{P}_2^G(s_i,t_i), 1 \leq i \leq h \\ x_e \geq 0 & \qquad e \in E. \end{aligned}$$

Recall that we restrict attention to the edge-weighted case of the 2-route edge-disjoint cut problems. For each $e \in E$ there is a variable $x(e)$ that in the binary case models whether e is in the cut ($x(e) = 1$) or not ($x(e) = 0$). In the LP relaxation, we let $x(e)$ be any non-negative number. If x is a feasible solution to the LP then $d_2(s_i,t_i;x) \geq 1$ for $1 \leq i \leq h$.

One can solve the LP in polynomial time using the ellipsoid method: the separation oracle is a simple min-cost flow problem (can be solved by the successive shortest path algorithm). We remark that the LP can be generalized in straight forward fashion for K-route cuts where K is any integer. The dual of the above LP can be seen to be a maximum 2-route multicommodity flow LP; there is a scaling factor of K involved depending on whether one counts the total flow or the total elementary K-flow and we ignore this issue for now.

For the single-source multiple-sink problem the pairs to be separated are st_i, $1 \leq i \leq h$ and for the multiway-cut problem the pairs are $s_i s_j$, $i \neq j$.

In the sequel we work with a feasible fractional solution to the LP and all our approximation bounds will be with respect to the lower bound provided by an optimum solution to the LP.

A Useful Lemma: The simple lemma below will be useful in our analysis.

Lemma 1. *Let $G = (V,E)$ be a graph and let $x : E \to [0, 1/3)$ be an edge weight function. Let $s \in V$ such that for all $u \in V \setminus \{s\}$, $d_2^G(s,u;x) \geq 1$. Then for any cycle C containing s there is a node v in C such that $d_1^G(s,v;x) > 1/3$.*

Corollary 1. *Let $G = (V,E)$ be a graph and let $x : E \to [0, 1/3)$ be an edge weight function. Let $s \in V$ and $B = B_2(s,0;x)$ such that for all $u \in V \setminus B$, $d_2^G(s,u;x) \geq 1$. Then for any cycle C containing s and a node $u \in V \setminus B$, there is a node $v \in C$ such that $d_1^G(s,v;x) > 1/3$.*

3 Single-Source 2-Route Cuts

We are given a graph G, a source s, and a set $T = \{t_1, t_2, \ldots, t_h\}$ of terminals that need to be 2-route separated from the source s. A reduction similar to the one given in [13] for the subset feedback edge set problem can be used to show that even the restricted case of the single-source 2-route cut problem where $T = V \setminus \{s\}$ is APX-hard.

We now give a rounding algorithm for single-source multi-sink problem. Let x be a feasible fractional solution for the 2-route cut instance on G. We will assume that in the solution x, each variable $x(e)$ is assigned a value that is an integral multiple of $1/n$. This can be ensured by replacing each $x(e)$ with $\min\{1, (\lfloor 4nx(e) \rfloor)(\frac{1}{n})\}$. Since no minimal edge-disjoint collection of paths contains more than $2n$ edges, it is easy to see that the resulting solution is feasible; the new solution's cost is at most four times that of the original. For a number $\alpha \in [0,1]$ we let $n(\alpha)$ denote the number $\lfloor n\alpha \rfloor \frac{1}{n}$.

Now pick a radius $\theta \in (0,1)$ uniformly at random. Let $B = B_2(s, \theta; x) = \{v \mid d_2^G(s, v; x) \leq \theta\}$, and let $A = V \setminus B$. Note that all terminals lie in A and that the induced graph $G[B]$ is 2-edge-connected. Our goal now is to alter the original LP solution x into another solution x' so that $d_2^G(s, v; x') \geq 1$ for all $v \in A$ and $d_2^G(s, v; x') = 0$ for all $v \in B$.

For an edge $e = (u, v)$, let $r(e)$ denote the least radius r such that $d_2^G(s, u; x) \leq r$ *and* $d_2^G(s, v; x) \leq r$. Note that $r(e)$ is the same as the length of the shortest (not-necessarily simple) cycle containing s and e with edge lengths given by x. We set $x'(e)$ as follows. If $r(e) \leq \theta$ we set $x'(e) = 0$, otherwise we scale up $x(e)$ by a factor of $1/(r(e) - n(\theta))$, that is, we set $x'(e) = x(e)/(r(e) - n(\theta))$. We observe that $e \in G[B]$ implies $x'(e) = 0$. An equivalent scaling process is to pick i uniformly at random from $0, 1, \ldots, n-1$ and set $\theta = i/n$.

Lemma 2. *In the solution* $\mathbf{x}'$, $d_2^G(s, u; x') \geq 1$ *for all* $u \in A$. *Moreover, for all* $v \in B, u \in A$, $d_2^G(v, u; x') \geq 1$.

Proof. For clarity of exposition, let $\alpha(v) = d_2^G(s, v; x)$ and $\beta(v) = d_2^G(s, v; x')$. Assume by way of contradiction that there is some $v \in A$ such that $\beta(v) < 1$. Among all such vertices choose w such that $\alpha(w)$ is largest. Let P and Q be two edge-disjoint paths from s to w such that $x'(P) + x'(Q) < 1$. By the choice of w, for any edge $e \in P \cup Q$, $r(e) \leq \alpha(w)$. Walk from w to s along P to find the first node a such that $a \in B$ (a exists since $s \in B$) and let P' be the sub-path of P from w to a. Using Q, define b and Q' as above. We claim that $x(P') + x(Q') \geq \alpha(w) - \theta$ for otherwise we can use P' and Q' to find two disjoint paths between s and w of total x length strictly less than $\alpha(w)$. We prove this claim after we use it to finish the proof of the lemma. Note that for any edge e in $P' \cup Q'$, $x'(e) = x(e)/(r(e) - n(\theta)) \geq x(e)/(r(e) - \theta) \geq x(e)/(\alpha(w) - \theta)$. Thus, after scaling $x'(P' \cup Q') \geq 1$ which implies that $x'(P \cup Q) \geq 1$ contradicting our assumption. For the second part of the lemma, we observe that $d_2^G(s, v; x') = 0$ for any $v \in B$. Further, by the triangle inequality $d_2^G(s, u; x') \leq d_2^G(s, v; x') + d_2^G(v, u; x')$ and hence $d_2^G(v, u; x') < 1$ implies $d_2^G(s, u; x') < 1$.

Now we prove the claim. Since $a \in B$, there are two edge disjoint paths P_a and Q_a from a to s such that $x(P_a) + x(Q_a) \leq \theta$. Further, P_a and Q_a have all their edges in $G[B]$. Similarly let P_b and Q_b be the paths for b. We claim that $P' \cup Q' \cup P_a \cup Q_a \cup P_b \cup Q_b$ contain two edge disjoint paths from w to s of total length at most

$$x(P') + x(Q') + (x(P_a) + x(Q_a))/2 + (x(P_b) + x(Q_b))/2 < \alpha(w) - \theta + \theta/2 + \theta/2 < \alpha(w).$$

It is easy to see that $P' \cup Q' \cup P_a \cup Q_a \cup P_b \cup Q_b$ contains two edge disjoint paths from w to s. To bound the total x-length of these paths, we create a *fractional* flow of two units from w to s of the desired length such that no edge has more than one unit of flow. Then the claim follows by using the fact that there exists an integer flow of no higher cost than the fractional flow. Send one unit of flow from w along P' to a which then splits the flow into a half unit along P_a and another half along Q_a. The other unit of flow is sent along Q' and split at b for P_b and Q_b. It can be checked that no edge has more than one unit of flow and that the x-length of this flow is equal to $x(P') + x(Q') + (x(P_a) + x(Q_a))/2 + (x(P_b) + x(Q_b))/2$. □

We next show that the expected cost of the resulting solution is only $O(\log n)$ times larger than the cost of the original solution.

Lemma 3. $\mathrm{E}_\theta[x'(e)] = O(\log n) \cdot x(e)$.

Proof. Let $r(e) = i/n$ for some integer i. Note that $x'(e) = 0$ if $\theta \geq r(e)$ and otherwise $x'(e) = x(e)/(r(e) - n(\theta))$. Therefore

$$E_\theta[x'(e)] = \int_0^{r(e)} \frac{x(e)}{r(e) - n(\theta)} \cdot d\theta = x(e) \sum_{j<i} \frac{1}{n} \cdot \frac{1}{i/n - j/n} = O(\log n) x(e).$$

□

Remark 1. Lemmas 2 and 3 hold for the following modified scaling procedure as well: $x'(e) = x(e)$ if $\theta \geq r(e)$ and $x'(e) = x(e)/(r(e) - n(\theta))$ otherwise.

The rounding procedures for 2-route multiway cut and 2-route multicut implicitly need the modifed analysis mentioned in the above remark. The analysis given in Lemma 3 is tight even when there is a single terminal to be separated from the source, and all edge weights are 1. Consider a graph G with vertices v_0 through v_n. Let $s = v_0$ and $t = v_n$. For each $0 \leq i < n$, there are two parallel edges between v_i and v_{i+1}. Consider an LP solution that for each pair of parallel edges assigns $x_e = 1/n$ on one of the edges and 0 on the other. Then $d_2(s, v_i) = i/n$. Now consider the edge e between $v_{n/2-1}$ and $v_{n/2}$. With probability $1/n$ the initial radius r is between i/n and $(i+1)/n$. When $i < n/2$, we scale e by a factor roughly $1/(1/2 - i/n)$. Thus the expected scaling factor is $\Omega(\log n)$; a similar argument shows that this holds for $\Omega(n)$ edges.

We now complete the description of the algorithm by showing how we can use the solution x' to find a feasible cut. We remove any edges e such that $x'(e) \geq 1/3$. In the remaining graph let $T' = \{u \mid d_1^G(s, u; x') \geq 1/3\}$. We solve

a single source min-cut problem to disconnect s from T'. Note that $3x'$ is sufficient to pay for both the above steps since the single source min-cut problem has an integrality gap of 1. By Corollary 1, any cycle involving s and a node from A contains a node from T'. Therefore, separating T' from s ensures that there is no cycle involving s and a node from A. Since all terminals are in A, the solution is feasible. The expected cost is $3\sum_e c(e)x'(e)$ which by Lemma 3 is $O(\log)\sum_e c(e)x(e)$. We can easily derandomize the procedure by using standard ideas; the proof of Lemma 3 shows the existence of a $\theta \in [0,1)$ such that $\sum_e c(e)x'(e) = O(\log n)\sum_e c(e)x(e)$. We observe that there are only n distinct values in $\{d_2(s,v;x) \mid v \in V\}$ that are relevant in choosing θ, hence we can try all these values and pick the one which results in the least cost. This gives the following theorem.

Theorem 1. *Single-source 2-route cut problem has an $O(\log n)$-approximation.*

4 2-Route Multiway Cut

Let $S = \{s_1, s_2, \ldots, s_h\} \subseteq V$ be a set of terminals. In the 2-route Multiway Cut problem the goal is to find a minimum cost set of edges whose removal 2-separates s_i and s_j for all $1 \leq i < j \leq h$. It is easy to show that the standard isolating cut heuristic [11], which gives a $2(1-1/h)$-approximation for the standard (1-route) multiway cut problem, yields an $\Omega(h)$-approximation for the 2-route variant.

Lemma 4. *The integrality gap of the LP for 2-route multiway-cut is $\Omega(\log n)$ even when $S = V$.*

Note that when $S = V$ the problem is equivalent to the feedback edge set problem and the LP solution for the 2-route problem is equivalent to the LP solution for the feedback edge problem for which the $\Omega(\log n)$ gap was observed in [13] using high-girth expanders.

We give an LP rounding approach that gives an $O(\log n \log h)$ approximation. Let x be a feasible solution to the LP. As before we will assume that $x(e)$ is an integer multiple of $1/n$. Let $e = (u,v)$. We let $r_i(e)$ to be the smallest r such that $B_2(s_i, r; x)$ contains both end points of e. We set $r(e) = \min_i r_i(e)$. The algorithm consists of the following steps.

1. Pick θ uniformly at random from $[0, 1/4)$.
2. For each e, set $x'(e) = \max\{2x(e), x(e)/(r(e) - n(\theta))\}$.
3. Remove edges e such that $x'(e) \geq 1/3$.
4. Separate all pairs (s_i, v) with $d_1^G(s_i, v; x') \geq 1/3$ by solving a multi-cut prob.
5. Output edges removed in Steps 3 and 4.

Theorem 2. *The 2-route multiway cut problem has an $O(\log n \log h)$-approximation.*

5 2-Route Multicut

We now consider the 2-route multicut problem. We are given G and h pairs $s_1t_1, s_2t_2, \ldots, s_ht_h$ and the goal is to 2-separate s_i from t_i for $1 \leq i \leq h$.

We give an LP rounding algorithm that essentially reduces it to the standard multicut problem. Our algorithm is inspired by the algorithms of Calinescu, Karloff and Rabani for multiway-cut [8] and 0-extension [9]; the underlying idea has seen several applications subsequently. Let x be a feasible solution to the LP. For an edge $e = uv$, we let $r_i(e) = \max\{d_2(s_i, u; x), d_2(s_i, v; x)\}$.

1. For each $u \in V(G)$, set $\rho(v) = 0$.
2. Pick θ uniformly at random from $[0, 1/2)$ and pick a random permutation σ of $\{1, 2, \ldots, h\}$.
3. For $i = 1$ to h do
   - For $v \in B_2(s_{\sigma(i)}, \theta; x)$, if $\rho(v) = 0$ then $\rho(v) = i$.
4. For each edge e:
   - Find the least index j such that $r_{\sigma(j)}(e) \leq \theta$; if no such j exists then set $j = h + 1$.
   - If $j = 1$, set $x'(e) = x(e)$, else set $x'(e) = \max_{i<j} x(e)/(r_{\sigma(i)}(e) - n(\theta))$.
5. Remove edges e such that $x'(e) \geq 1/10$.
6. In G separate all pairs (p, q) with $d_1^G(p, q; x') \geq 1/6$ via a multicut algorithm.
7. Output edges removed in Steps 5 and 6.

We observe that the first three steps of the above algorithm are similar to the adaptation of the CKR procedure and analysis from [9] for the (1-route) multicut problem (see lecture notes [17,10] for details of this). The only difference is that step 3 is performed with respect to 1-route distance. At the end of step 3, the 1-route multicut algorithm outputs as solution all edges (u, v) such that $\rho(u) \neq \rho(v)$. The feasibility of this solution is immediate since the radius of each ball is less than $1/2$, and hence no ball can contain both a source and its corresponding sink. An elegant argument from [9] can then be used to show that the expected cost of this solution is within an $O(\log n)$ factor of the optimal. We note that the classical region growing algorithm of [15] may be viewed as a deterministic version of this randomized ball-growing process.

In contrast, for 2-route multicut, a critical step is the randomized scaling (step 4) which allows us in effect to reduce our problem to an instance of 1-route multicut. The cost analysis of the resulting solution combines the scaling analysis from Lemma 3 with the argument from [9] followed by the integrality gap for the standard 1-route multicut [15]; this is not too difficult. The main difficulty, however, is in proving the feasibility of the resulting solution. In the setting of 2-route distance, the sets $\{v \mid \rho(v) = i\}$ are difficult to visualize, and the intuitive distance based arguments are no longer applicable. We rely on a careful inductive proof to argue for the feasibility of the cut.

5.1 Feasibility

We will show that the solution obtained in the step 7 above is indeed a feasible solution. For clarity of exposition, assume without loss of generality that the permutation σ is an identity permutation. For $i \in [1..h]$ let $V_i = \{w \mid \rho(w) = i\}$.

Lemma 5. *For any node $w \in V_i$ and $u \in V \setminus V_i$, we have $d_2(w, u; x') \geq 1$.*

The main technical lemma needed to establish feasibility is stated below.

Lemma 6. *For any $i \in [1..h]$, let C be any cycle (possibly non-simple) that involves a node $w \in V_i$ and a node $u \in V \setminus V_i$. Then after scaling in the step 4 of the algorithm, either the cycle C has a pair of nodes p, q such that $d_1(p, q; x') \geq 1/6$ or there is an edge e on C such that $x'(e) \geq 1/10$.*

We now finish the proof of the feasibility of the solution output by the algorithm. For any pair $s_i t_i$, $1 \leq i \leq h$, we claim that $\rho(s_i) \neq \rho(t_i)$. Suppose not. Let $\rho(s_i) = \rho(t_i) = q$. Then $s_i \in B_2(s_{\sigma(q)}, \theta; x)$ and $t_i \in B_2(s_{\sigma(q)}, \theta; x)$ which implies that $d_2(s_i, t_i; x) \leq 2\theta$. Since $\theta < 1/2$ this would imply that $d_2(s_i, t_i; x) < 1$ which contradicts the feasibility of x.

From above and Lemma 6, for any cycle C that contains both s_i and t_i, either there is any edge e in C such that $x'(e) \geq 1/10$ or there are nodes p, q in C such that $d_1(p, q; x') \geq 1/6$. Since the algorithm removes all edges f with $x'(f) \geq 1/10$ (in Step 5) and ensures that there is no path between nodes p, q with $d_1(p, q; x') \geq 1/6$ (in Step 6), every cycle C between s_i and t_i is removed.

5.2 Cost Analysis

We will first analyze the cost of the solution x'. To do so, it suffices to consider the expected scaling factor for any edge in G. Fix $\theta \in (0, 1/2)$ and an edge $e = (u, v)$. Recall that $r_i(e) = \max\{d_2(s_i, u; x), d_2(s_i, v; x)\}$. By renumbering pairs, assume that $r_1(e) \leq r_2(e) \leq ... \leq r_h(e)$. We will denote by $f(e)$ the scaling factor for edge e. Define $f_i(e, \theta) = 1$ if $r_i(e) \leq \theta$ and $\frac{1}{(r_i(e) - n(\theta))}$ otherwise. The scaling factor $f(e)$ for edge e is determined to be $f_i(e, \theta)$ only if in the random permutation σ, the source s_i occurs before each one of $s_1, s_2, ..., s_{i-1}$. The probability of this event is at most $1/i$. Thus for a fixed choice of θ, the expected scaling factor for an edge e can be bounded by $E_\sigma[f(e)] \leq \sum_{i=1}^{h} \frac{1}{i} f_i(e, \theta)$.

Taking the expectation over θ, which is independent of σ, we get the expected scaling factor for the edge e is at most

$$\begin{aligned} E_{\theta,\sigma}[f(e)] &\leq \int_0^{1/2} \sum_{i=1}^{h} \frac{1}{i} f_i(e, \theta) \cdot d\theta \\ &= \sum_{i=1}^{h} \frac{1}{i} \left(\int_0^{r_i(e)} \frac{1}{r_i(e) - n(\theta)} \cdot d\theta + \int_{r_i(e)}^{1/2} 1 \cdot d\theta \right) \\ &= O(\log h \log n). \end{aligned}$$

Thus the expected cost of the solution x' is $O(\log h \log n)$ times the cost of the solution x. Finally, we lose another factor of $O(\log n)$ in solving the multicut instance on x'. We thus get the following theorem.

Theorem 3. *The 2-route multicut problem has an $O(\log h \log^2 n)$-approximation.*

References

1. Aggarwal, C., Orlin, J.: On Multi-route Maximum Flows in Networks. Networks 39, 43–52 (2002)
2. Bagchi, A., Chaudhary, A., Kolman, P.: Short length Menger's theorem and reliable optical networking. Theoretical Computer Science 339, 315–332 (2005)
3. Bagchi, A., Chaudhary, A., Kolman, P., Sgall, J.: A simple combinatorial proof for the duality of multiroute flows and cuts. TR 2004-662, Charles Univ. (2004)
4. Bar-Yehuda, R., Geiger, D., Naor, J., Roth, R.M.: Approximation Algorithms for the Feedback Vertex Set Problem with Applications to Constraint Satisfaction and Bayesian Inference. SIAM J. Comput. 27(4), 942–959 (1998)
5. Brightwell, G., Oriolo, G., Shepherd, F.B.: Some strategies for reserving resilient capacity. SIAM J. on Discrete Math. 14(4), 524–539 (2001)
6. Brightwell, G., Oriolo, G., Shepherd, F.B.: Reserving Resilient Capacity for a Single Commodity with Upper Bound Constraints. Networks 41(2), 87–96 (2003)
7. Bruhn, H., Cerny, J., Hall, A., Kolman, P.: Single Source Multiroute Flows and Cuts on Uniform Capacity Networks. In: Proc. of ACM-SIAM SODA (2007)
8. Călinescu, G., Karloff, H., Rabani, Y.: An improved approximation algorithm for multiway cut. Journal of Computer and System Sciences 60, 564–574 (2000)
9. Călinescu, G., Karloff, H., Rabani, Y.: Approximation algorithms for the 0-extension problem. SIAM J. on Computing 34(2), 358–372 (2004)
10. Chekuri, C.: Lecture notes on Multicut rounding via CKR method, http://www.cs.uiuc.edu/homes/chekuri/teaching/fall2006/lect15.pdf
11. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM J. on Computing 23, 864–894 (1994)
12. Even, G., Naor, J., Rao, S., Schieber, B.: Divide-and-conquer approximation algorithms via spreading metrics. J. ACM 47, 585–616 (2000)
13. Even, G., Naor, J., Schieber, B., Zosin, L.: Approximating minimum subset feedback sets in undirected graphs with applications. SIAM J. Disc. Math. 13(2), 255–267 (2000)
14. Even, G., Naor, J., Zosin, L.: An 8-approximation for the subset feedback vertex set problem. SIAM J. on Computing 30(4), 1231–1252 (2000)
15. Garg, N., Vazirani, V., Yannakakis, M.: Approximate Max-Flow Min-(Multi)Cut Theorems and Their Applications. SIAM J. Comput. 25(2), 235–251 (1996)
16. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1987)
17. Gupta, A., Ravi, R.: Lecture notes on LP solutions as Metrics: MultiCut, and Region Growing, http://www.cs.cmu.edu/afs/cs/academic/class/15854-f05/www/scribe/lec20.pdf
18. Günlük, O.: A new min-cut max-flow ratio for multicommodity flows. SIAM J. on Discrete Math. 21(1), 1–15 (2007); Preliminary version In: Proc.of IPCO (2002)

19. Karger, D., Klein, P., Stein, C., Thorup, M., Young, N.: Rounding algorithms for a geometric embedding of minimum multiway cut. In: Proceedings of the 29th ACM Symposium on Theory of Computing, pp. 668–678 (1999)
20. Kishimoto, W.: A method for obtaining the maximum multi-route flow in a network. Networks 27(4), 279–291 (1996)
21. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM 46(6), 787–832 (1999); Prelim. version In: Proc. of IEEE FOCS (1988)
22. Vazirani, V.: Approximation Algorithms. Springer, Heidelberg (2001)

The Two-Edge Connectivity Survivable Network Problem in Planar Graphs

Glencora Borradaile[1,⋆] and Philip Klein[2,⋆⋆]

[1] Department of Combinatorics and Optimization, University of Waterloo
glencora@uwaterloo.ca
[2] Computer Science Department, Brown University
klein@cs.brown.edu

Abstract. Consider the following problem: given a graph with edge-weights and a subset Q of vertices, find a minimum-weight subgraph in which there are two edge-disjoint paths connecting every pair of vertices in Q. The problem is a failure-resilient analog of the Steiner tree problem, and arises in telecommunications applications. A more general formulation, also employed in telecommunications optimization, assigns a number (or *requirement*) $r_v \in \{0, 1, 2\}$ to each vertex v in the graph; for each pair u, v of vertices, the solution network is required to contain $\min\{r_u, r_v\}$ edge-disjoint u-to-v paths.

We address the problem in planar graphs, considering a popular relaxation in which the solution is allowed to use multiple copies of the input-graph edges (paying separately for each copy). The problem is SNP-hard in general graphs and NP-hard in planar graphs. We give the first polynomial-time approximation scheme in planar graphs. The running time is $O(n \log n)$.

Under the additional restriction that the requirements are in $\{0, 2\}$ for vertices on the boundary of a single face of a planar graph, we give a linear-time algorithm to find the optimal solution.

1 Introduction

In the field of telecommunications network design, an important requirement of networks is resilience to link failures [19]. The goal of the survivable network problem is to find a graph that provides multiple routes between pairs of terminals. In this work we focus on edge-disjoint paths, though vertex-disjoint paths have also been the subject of research. More formally for Z a set of non-negative integers, the input to the Z-edge connectivity problem is a weighted, undirected graph G and an assignment of *connectivity requirements* $r_v \in Z$ to vertices v. The goal is to find a minimum-weight subgraph such that, for each pair u, v of vertices, the subgraph contains at least $\min\{r_u, r_v\}$ edge-disjoint u-to-v paths. Because it is considered unlikely that two links would fail simultaneously, some research has focused on requiring at most two paths between vertices that need

⋆ Work done while at Brown University.

⋆⋆ Supported by NSF grant CCF-0635089. Work done while visiting MIT.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 485–501, 2008.

to be connected. There is a wealth of literature on such *low-connectivity network design problems*. Resende and Pardalos [19] survey the literature, which includes heuristics, structural results, polyhedral results, computational results using cutting planes, and approximation algorithms.

This work focuses on $\{0,1,2\}$-edge connectivity. We consider the well-studied relaxation wherein the solution subgraph is allowed to contain multiple copies of each edge of the input graph. We call such a subgraph a *sub-multigraph* and the weight of the edges appearing twice in the solution is counted according to multiplicity. For two-connectivity, at most two copies of an edge are needed. This version of the problem, like the other variants, is SNP-hard in general graphs [6]. In [3], Berger and Grigni gave a polynomial-time approximation scheme (PTAS) for $\{1,2\}$-edge connectivity (ie. the spanning case) in planar multigraphs. A year later, a PTAS was given for $\{0,1\}$-edge connectivity (that is, the Steiner tree problem) in planar graphs. Here we give a PTAS for the $\{0,1,2\}$-edge connectivity (ie. the subset case) for planar multigraphs. The running time is significantly lower than that of [3]. In the following, OPT denotes the weight of the optimal solution to the problem at hand.

Theorem 1. *Let G be a planar graph with nonnegative edge-weights and integer requirements $r_v \in \{0,1,2\}$ for each vertex v. For any $0 < \epsilon < 1$, there is an $O(n \log n)$ algorithm that finds a sub-multigraph H of G such that for every pair u, v of vertices, there are at least $\min\{r_u, r_v\}$ edge-disjoint u-to-v paths in H. Further, the total weight of the edges in H is at most $(1+\epsilon)$OPT.*

An important special case involves finding a sub-multigraph that achieves two-edge connectivity between a given set Q of vertices. Our approximation scheme addresses this problem (i.e. $r_v = 2$ for all $v \in Q$ and $r_v = 0$ for all $v \notin Q$). In addition, for the special case where the vertices of Q are on the boundary of a common face, we give a linear-time algorithm to find the optimal solution:

Theorem 2. *There is a linear-time algorithm that, given a planar embedded graph with edge-weights and a subset Q of the vertices on the boundary of a single face, finds a minimum-weight two-edge-connected sub-multigraph of G spanning Q.*

For ease of exposition, we will take the the face on which the vertices Q lie to be the outermost or infinite face of the planar embedded graph. That is, the vertices of Q lie on the boundary of the planar graph.

Both results rely on a common observation (Theorem 3, Section 2) concerning the structure of two-edge connectivity between boundary vertices of planar graphs.

1.1 Related Work

Two-edge-connected spanning subgraph. A special case that has received much attention is the problem of finding a minimum-weight subgraph of G in which *every* pair of vertices is two-connected. This problem is called *two-edge-connected*

spanning subgraph, and is NP-hard [8] and max-SNP complete [6] in general graphs. Frederickson and JáJá [9] gave a 3-approximation algorithm for this problem. The approximation ratio was improved to 2 by Khuller and Vishkin [14]. For the unweighted case, they gave a 1.5-approximation algorithm. Jothi, Raghavachari, and Varadarajan [13] improved the approximation ratio to 5/4.

In planar graphs the problem is NP-hard. Berger et al. [2] and Berger and Grigni [3] gave PTASes for the unweighted and weighted cases, respectively, in planar graphs. In both cases, the degrees of the polynomial depend on the desired precision ϵ. All the above algorithms work for the case where the output is not allowed to duplicate edges. For the case where duplication is allowed, the techniques of Klein [15] can be applied to obtain a linear-time approximation scheme.

Beyond spanning. For the more general case where a subset Q of the vertices need only be spanned, Ravi [18] showed that Frederickson and JáJá's approach could be generalized to give a 3-approximation algorithm (in general graphs). Klein and Ravi [17] gave a 2-approximation for a more general problem in which the input specifies which pairs of vertices must be connected up. This result was greatly generalized by Williamson, Goemans, Mihail, and Vazirani [20], Goemans, Goldberg, Plotkin, Shmoys, Tardos, and Williamson [10], and Jain [12]. These algorithms did not require duplication of edges.

In their recent paper on the spanning case, Berger and Grigni raise the question of whether there is a PTAS for finding a minimum-weight two-edge-connected subgraph of a planar graph. In this paper, we answer that question in the affirmative, at least when edge duplications are allowed.

1.2 Notation

For a path P, $P[x, y]$ denotes the x-to-y subpath of P for vertices x and y of P. For paths A and B, $A \circ B$ denotes the concatenation of A and B. For a subgraph H of a graph G, we use $V(H)$ to denote the set of vertices in H. We similarly use the notation $V(P)$, etc.

We employ the usual definitions of planar embedded graphs. For a face f, the cycle of edges making up the boundary of f is denoted ∂f. We assume the planar graph G is connected and is embedded in the plane, so there is a single infinite face, and we denote its boundary by ∂G.

For a cycle C in a planar embedded graph, $C[x, y]$ denotes an x-to-y path in C for vertices x and y of C. There are two such paths and the choice between the two possibilities will be disambiguated by specifying an orientation of the cycle (clockwise or counterclockwise). A cycle C is said to *enclose* the faces that are embedded inside it. C encloses an edge/vertex if the edge/vertex is embedded inside it or on it. In the former case, C *strictly encloses* the edge/vertex.

See Figure 1 for an illustration of the notion of paths crossing. A cycle is non-self-crossing if every pair of subpaths of the cycle do not cross. Two trees are noncrossing if no path in one crosses a path of the other.

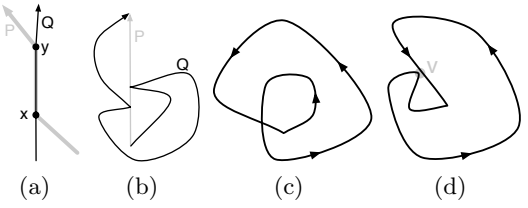


Fig. 1. (a) P crosses Q. (b) P and Q are noncrossing. (c) A self-crossing cycle. (d) A non-self-crossing cycle (non-self-crossing allows for repeated vertices, i.e. v.).

1.3 Outline

In Section 2, we establish some key properties of two-edge-connectivity between boundary vertices of a planar graph. In Section 3, we prove Theorem 2 by giving a linear-time algorithm for the special case of finding the minimum two-edge-connected subgraph containing a subset of the boundary vertices of a planar embedded graph. In Section 4, we build on the results in Section 2 to give a decomposition of solutions to the two-edge connectivity survivable network problem in planar graphs where all terminals are on the boundary.

The remainder of the paper is devoted to proving the PTAS of Theorem 1. The approximation scheme employs an approach used by Borradaile, Klein, and Mathieu [5] to obtain an approximation scheme for Steiner tree. In Section 5, we outline the approach. In particular, what is needed is a structural theorem that states that the interaction between different parts of an optimal solution can be restricted to be "simple" while paying only a small penalty (in relative terms) in weight. We restate this theorem (Theorem 4) as given in [5] for the Steiner tree problem. The corresponding theorem for two-edge connectivity (Theorem 5) appears in Section 6. The proof draws on the results of Sections 2 and 4 and the corresponding structure theorem for Steiner trees. Finally, in Section 7, we briefly outline the dynamic program that is at the heart of the computation.

2 Basic Structural Properties of Boundary Connectivity

The results of this section hold for both subgraphs and sub-multigraphs. In this section, we investigate the structure of sub-(multi)graphs of G that achieve up to $\{0, 1, 2\}$-edge-connectivity between vertices of ∂G.

Since we are only interested in connectivity up to and including two-edge connectivity, we define the following: for a graph H and vertices x, y, let

$$c_H(x, y) = \min\{2, \text{maximum number of edge-disjoint } x\text{-to-}y \text{ paths in } H\}.$$

For two sub-multigraphs H and H' of a common graph G and for a subset S of the vertices of G, we say H' *achieves the two-connectivity of* H for S if $c_{H'}(x, y) \geq c_H(x, y)$ for every $x, y \in S$. We say H' achieves the *boundary* two-connectivity of H if it achieves the two-connectivity of H for $S = V(\partial G)$.

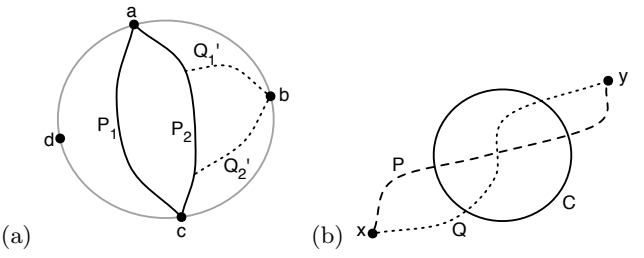


Fig. 2. (a) An illustration of the paths in Lemma 2. (b) An illustration of the proof of Lemma 3: there are edge-disjoint x-to-y paths that do not use edges enclosed by C.

Lemma 1 (Transitivity). *For any graph H, for vertices $u, v, w \in V(H)$, $c_H(u, w) \geq \min\{c_H(u, v), c_H(v, w)\}$*

Lemma 2 (Crossing). *Let G be a planar embedded graph, and let its boundary be $v_1 v_2 \ldots v_n$. For integers $1 \leq i < j < k < \ell \leq n$, for any subgraph H of G, $c_H(v_i, v_j) \geq \min\{c_H(v_i, v_k), c_H(v_j, v_\ell)\}$.*

Proof (Sketch). For the case of connectivity two, see Figure 2(a). Given 2 edge-disjoint a-to-c paths (P_1 and P_2) and 2 edge-disjoint b-to-d paths (whose prefixes are Q_1' and Q_2'), it is easy to construct 2 a-to-b edge-disjoint paths. The proof for connectivity one is similar but simpler. □

Lemma 3. *Let H be a sub-(multi)graph of G and let C be a non-self-crossing cycle of H. Let H' be the subgraph of H obtained by removing the edges of H that are strictly enclosed by C. H' achieves the boundary 2-connectivity of H.*

Proof. See Figure 2(b). Without loss of generality, let C be a simple cycle that is clockwise according to the planar embedding. Consider two vertices x and y of ∂G. We show that there are $c_H(x, y)$ edge-disjoint x-to-y paths in H that do not use edges strictly enclosed by C. There are two non-trivial cases: $c_H(x, y) = 1$ and $c_H(x, y) = 2$. We omit the former case, as the latter is illustrative.

Let P and Q be edge-disjoint x-to-y paths in H. If Q does not intersect C, then P' and Q are edge-disjoint paths, neither of which has a dart strictly enclosed by C (where P' is as defined above). Suppose that both P and Q intersect C. Let x_Q and y_Q be vertices of Q defined as for P. Suppose these vertices are ordered x_P, x_Q, y_Q, y_P around C. Then $P[x, x_P] \circ C[x_P, y_Q] \circ Q[y_Q, y]$ and $Q[x, x_Q] \circ rev(C[y_P, x_Q]) \circ P[y_P, y]$ are edge disjoint x-to-y paths that do not use any edges enclosed by C. This case is illustrated in Figure 2(b); other cases follow similarly.

We have shown that we can achieve the boundary two-connectivity of H without using any edges enclosed by a cycle of H. The lemma follows. □

Corollary 1. *Let H be a subgraph of G and let H' be a minimal subgraph of H that achieves the boundary two-connectivity of H. Then in H' every cycle C strictly encloses no edges.*

Lemma 4. *Let H be a subgraph of G. Let S be a subset of $V(\partial G)$ such that, for every $x, y \in S$, $c_H(x, y) = 2$. Then there is a non-self-crossing cycle C in H such that $S \subseteq V(C)$ and the order that C visits the vertices in S is the same as their order along ∂G.*

Proof (sketch). Assume that the vertices of S are in the order $s_1, s_2, \ldots, s_k$ along ∂G. Let $\partial G[s_{i+1}, s_i]$ denote the subpath of the boundary of G between s_{i+1} and s_i that does not go through s_j for $j \neq i, i+1$. Let P_i be the s_i-to-s_{i+1} path in H (taking the indices mod k) such that the cycle $P_i \circ \partial G[s_{i+1}, s_i]$ encloses only one s_i-to-s_{i+1} path (namely, P_i). One can show that P_i does not cross P_j for any pair i, j. The cycle $C = P_1 \circ P_2 \circ \cdots \circ P_{k-1}$ has the properties required by the lemma. □

3 Linear-Time Exact Algorithm for a Boundary Two-Edge-Connectivity Problem

Here we give a linear-time algorithm for the following problem: given a weighted, planar graph G and a subset Q of the vertices of ∂G, find a minimum-weight two-edge-connected sub-multigraph of G that spans Q. This will prove Theorem 2, as stated in the Introduction. The algorithm whose correctness will follow from Lemma 4, is:
BOUNDARY2EC(G, Q)

1. Let $q_1, q_2, \ldots$ be the cyclic ordering of the terminals in Q along ∂G.
2. For $i = 1, \ldots$, let P_i be the shortest q_i-to-q_{i+1} path in G (taking the indices mod $|Q|$).
3. Return the disjoint union $\cup_i P_i$.

Using the following lemma, we show that BOUNDARY2EC can be implemented in linear time using the linear-time shortest path algorithm for planar graphs [11].

Lemma 5. *Let a, b and c be vertices ordered along the clockwise boundary ∂G of a planar graph G. Let T_a be the shortest-path tree rooted at a. Then there is a shortest b-to-c path in G that is enclosed by the cycle $\partial G[b, c] \circ T[c, b]$.*

Proof (sketch). Suppose that the shortest b-to-c path P in G is not enclosed by the cycle $\partial G[b, c] \circ T[c, b]$. Then there is a subpath of P that contradicts the shortness of T. □

A linear-time implementation of BOUNDARY2EC is: compute a shortest-path tree T rooted at terminal q_1 in linear time; for each i, consider the graph G_i enclosed by $C_i = \partial G[q_i, q_{i+1}] \circ T[q_{i+1}, q_i]$; compute the shortest q_i-to-q_{i+1} path P_i in G_i. By Lemma 5, P_i is a shortest q_i-to-q_{i+1} path in G. Since each edge of G appears in at most two subgraphs G_i and G_j, the paths P_i can be computed in linear time.

We now argue that BOUNDARY2EC finds the minimum-weight two-edge-connected multi-subgraph of G that spans Q. Certainly BOUNDARY2EC returns a 2-edge-connected multi-subgraph that spans Q. We show that the graph

BOUNDARY2EC finds is of minimum weight. Let H be the optimal solution. By Lemma 4, there is a cycle C in H that visits the vertices $q_1, q_2, \ldots$ in order. This cycle can be written as $L_1 \circ L_2 \circ \cdots$ where L_i is a q_i-to-q_{i+1} path. Let P_i be the shortest q_i-to-q_{i+1} path. Then $w(P_1 \circ P_2 \circ \cdots) \leq w(L_1 \circ L_2 \circ \cdots) \leq w(H)$.

4 Decomposition Result for Boundary Connectivity

For the theorem given in this section, we have to generalize the notion of connectivity requirements. Connectivity requirements so far assign an integer to each vertex; the corresponding subgraph must ensure connectivity at least $\min\{r_u, r_v\}$ between u and v. One can instead specify a connectivity requirements for each pair of vertices, using a function from the set of two-element subsets of $V(\partial G)$ (written $\binom{V(\partial G)}{2}$) to $\{0, 1, 2\}$.

Theorem 3. *Let G be a connected planar embedded graph. Let $r : \binom{V(\partial G)}{2} \longrightarrow \{0, 1, 2\}$ be a function specifying connectivity requirements among the boundary vertices. There is a collection $\mathcal{X} = \{X_1, \ldots, X_k\}$ of subsets of $V(\partial G)$ that are noncrossing with respect to ∂G such that a minimal subgraph H of G satisfies connectivity requirements $r(\cdot)$ iff H contains edge-disjoint non-crossing trees $T_1, T_2, \ldots, T_k$ where, for each i, T_i spans X_i.*

In the following we will assume for notational convenience that the boundary of the graph G is a simple cycle; that is, a vertex appears at most once along ∂G. Let us see why it suffices to prove the theorem with this assumption. Suppose the boundary of G is not simple: there is a vertex v that appears at least twice along ∂G. Partition G into two graphs G_1 and G_2 such that v appears exactly once along ∂G_1 and $E(\partial G) = E(\partial G_1) \cup E(\partial G_2)$. Let x be a vertex of ∂G_1 and let y be a vertex of ∂G_2. Then $c_G(x, y) = \min\{c_{G_1}(x, v), c_{G_2}(v, y)\}$.

Let $a_1 a_2 a_3 a_4 \cdots a_m$ be the alternating sequence of vertices and edges of ∂G in the order in which they are encountered during a clockwise traversal. We say $\{a_i, a_k\}$ and $\{a_j, a_\ell\}$ *cross* if $i < j < k < \ell$.

We start with some definitions that will lead to the definition of the sets making up $\mathcal{X}$:

- $\sim_2$ is a relation on the vertices of ∂G: $u \sim_2 v$ if $r(\{u, v\}) = 2$.
- $\sim_2^*$ is the transitive and crossing closure of $\sim_2$. That is, $\sim_2^*$ is the minimal superset of $\sim_2$ such that if $x \sim_2^* y$ and $u \sim_2^* v$ and either $\{x, y\}$ crosses $\{u, v\}$ or $y = u$, then $x \sim_2^* v$.
- $\sim_1$ is a relation on the vertices of ∂G: $x \sim_1 y$ if $r(\{x, y\}) \geq 1$. Let $\sim_1^*$ be the transitive and crossing closure of $\sim_1$.
- $r_1^* : \binom{V(\partial G)}{2} \longrightarrow \{0, 1\}$ is a requirement function such that $r_1^*(\{u, v\}) = 1$ iff $u \sim_1^* v$.
- $\sim_0$ is a relation on the edges of ∂G: $a \sim_0 b$ if there is no set $\{u, v\} \subset V(\partial G)$ that crosses $\{a, b\}$ such that $u \sim_2^* v$. It is easy to see that $\sim_0$ is an equivalence relation.

Let $E_1, \ldots, E_\ell$ be the equivalence classes of $\sim_0$. For $i = 1, \ldots, \ell$, let $Z_i = \bigcup\{\text{endpoints of } e \ : \ e \in E_i\} \cap V(H)$, and let $\mathcal{X}_i = \{W \cap Z_i \ : \ W \text{ an equivalence class of } r_1^*\}$. Let $\mathcal{X} = \bigcup_i \mathcal{X}_i$, and write $\mathcal{X}_i = \{X_1, \ldots, X_k\}$. There are two parts to the proof of the theorem.

Part 1: For $i = 1, \ldots, k$, let T_i be a tree that connects X_i where the T_i's are edge-disjoint. Let $H = \bigcup_i T_i$. We will show that H satisfies the original connectivity requirements $r(\cdot)$, thus proving the forward direction of the theorem. We must show (A) if $r(\{x, y\}) = 2$ then $c_H(x, y) = 2$, and (B) if $r(\{x, y\}) = 1$ then $c_H(x, y) \geq 1$.

Let $Y_1, \ldots, Y_\ell$ be the equivalence classes of $\sim_2^*$. For each Y_i, we will show that H contains a cycle C_i through the vertices of Y_i, which will prove (A). Let the vertices of Y_i be $x_0, x_1, \ldots, x_{p-1}$, numbered according to their occurrence in a clockwise traversal of ∂G.

Claim 1: For $j = 0, \ldots, p-1$, there is some $X \in \mathcal{X}$ that contains x_j and $x_{j+1 \bmod p}$.

Proof. Let e be the edge immediately after x_j in clockwise traversal of ∂G, and let e' be the edge immediately before $x_{j+1 \bmod p}$. Suppose there were a subset $\{u, v\} \subset V(\partial G)$ that crosses $\{e, e'\}$ such that $u \sim_2^* v$. Assume without loss of generality that u occurs after e and before e' in clockwise traversal of ∂G. Then v occurs after e' and before e. It follows that one of the following must hold: $v = x_j$ or $v = x_{j+1 \bmod p}$ or $\{u, v\}$ crosses $\{x_j, x_{j+1 \bmod p}\}$. In each case, since $\sim_2^*$ is closed under crossing and transitivity, $u \sim_2^* x_j$, contradicting the fact that x_j and $x_{j+1 \bmod p}$ are consecutive elements of Y_i. This shows that $e \sim_0 e'$, which shows in turn shows that x_j and $x_{j+1 \bmod p}$ are in a common set $Z \in \mathcal{Z}$. Since $r(\{x_j, x_{j+1}\}) = 2$, we infer $r_1^*(\{x_j, x_{j+1}\}) = 1$, so there is a set $X \in \mathcal{X}$ (with $X \subset Z$) that contains x_j and x_{j+1}. □

Let P_j be the x_j-to-x_{j+1} path in T (the tree that spans X). By combining these paths for $j = 0, 1, \ldots, p-1$, we obtain a cycle C_i, proving (A).

Now we prove (B). Let $U_1, \cdots, U_s$ be all the equivalence classes of $\sim_2^*$ such that there is pair in U_i crosses $\{x, y\}$ for every i. Assume that these sets are ordered according to their distance from x along ∂G (in, say, the clockwise direction). Let u_i and v_i be two distinct vertices of U_i chosen such that u_1 is the vertex of U_1 closest to x along ∂G and u_s is the vertex of U_s closest to y along ∂G. (See Figure 3.)

If $s = 0$ then there are edges e_x and e_y adjacent to x and y respectively such that $e_x \sim e_y$. So x and y are in a common set $Z \in \mathcal{Z}$. Since $r(\{x, y\}) = 1$ and $x \sim_1^* y$ then $r^*(\{x, y\}) = 1$ and x and y are in a common set $X_i \in \mathcal{X}$. Therefore T_i (and hence H) contains an x-to-y path.

Suppose that $s > 0$. The argument is illustrated in Figure 3. Since $x \sim_1^* y$ and $\sim_1^*$ is closed under crossing, $x \sim_1^* u_i$ for $i = 1, \ldots, s$. Since $\sim_1^*$ is closed under transitivity, $u_i \sim_1^* u_{i+1}$ for $j = 1, \ldots, s-1$. By choice of u_1, $x \sim_0 u_1$, so x and u_1 are in a common set $X_i \in \mathcal{X}$. Therefore T_i (and hence H) contains an x-to-u_1 path. Similarly H contains a u_s-to-y path.

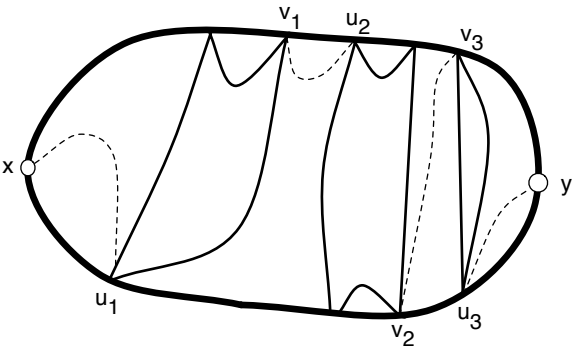


Fig. 3. The argument for one-connectivity is illustrated. Because $\{x, y\}$ crosses $\{u_1, u_2\}$, a connectivity requirement arises between x and u_2. Hence there is an x-to-u_2 path. Moreover, for each equivalence class, there is a cycle (indicated in medium-bold) connecting the members u_i. Combining the paths with the cycles yields an x-to-y path.

By (A), H contains a u_i-to-v_i path for $i = 1, \ldots, s$. For $i = 1, \ldots, s-1$ we argue that H contains a u_i-to-u_{i+1} path. Since $u_i \sim_0 u_{i+1}$ and by the transitivity of $\sim_1^*$, u_i and u_{i+1} are in a common set X_i, so T_i contains such a path. Combining these paths, we obtain an x-to-y path in H, proving (B) and the forward direction of Theorem 3.

Part 2: Let H be a subgraph of G that satisfies the connectivity requirements $r(\cdot)$. Assume without loss of generality that H is edge-minimal subject to this condition. We will show how to decompose H into noncrossing, edge-disjoint subgraphs $T_1, \ldots, T_k$, so that T_i spans X_i.

By Lemmas 1 and 2, for any vertices $x, y \in V(\partial G)$, if $x \sim_2^* y$ then $c_H(x, y) = 2$. As in the proof of Part 1, let $Y_1, \ldots, Y_\ell$ be the equivalence classes of $\sim_2^*$. For each Y_j, let C_j be the corresponding non-self-crossing cycle in H whose existence is guaranteed by Lemma 4. By Corollary 1, C_j does not strictly enclose any edges.

Let R be the subgraph of G consisting of $\partial G \cup \bigcup_{j=1}^{\ell} C_j$. Let $\mathcal{F}$ be the set of faces of R other than the infinite face and the faces in the interiors of cycles C_j. For each face $f \in \mathcal{F}$, let H_f be the subgraph of H consisting of edges enclosed by ∂f.

Claim 2: For distinct faces $f_1, f_2 \in \mathcal{F}$, H_{f_1} and H_{f_2} are edge-disjoint.

Proof. The set of edges *strictly* enclosed by f_1 and the set of edges enclosed by f_2 are clearly disjoint. We need to address the case of edges not strictly enclosed by f_1, i.e. edges of ∂f_1. Every edge e of R belongs either to ∂G or to some cycle C_i, so e is on the boundary of some face not in $\mathcal{F}$. Hence e is on the boundary of at most one face in $\mathcal{F}$. □

Claim 3: For any face $f \in \mathcal{F}$ and any vertices $x, y \in V(\partial f \cap \partial G)$, if $x, y \in X \in \mathcal{X}$ then H_f contains an x-to-y path.

Proof. By Lemmas 1 and 2, if $x \sim_1^* y$ then $c_H(x, y) \geq 1$. Hence H contains such a path P. Suppose P is not a path of H_f, and consider a maximal subpath P' of P that is not enclosed by ∂f. By maximality, the endpoints of P' must lie on ∂f. Since P' is enclosed by ∂G, the endpoints of P' must lie on a subpath Q of $\partial f \cap (\bigcup_{j=1}^{\ell} C_j)$. Thus Q belongs to H, and therefore to H_f. The subpath P' of P can therefore be replaced by Q. Similarly replacing each such subpath yields an x-to-y path in H_f. □

Claim 4: Let f be a face in $\mathcal{F}$, and let a, b be edges of $\partial G \cap \partial f$. Then there is some equivalence class E_i of $\sim_0$ that contains a and b.

Proof. Assume for a contradiction that there is a subset $\{u, v\} \subset V(\partial G)$ that crosses $\{a, b\}$ such that $u \sim_2^* v$. There is some subset Y_j containing u and v, and therefore some cycle C_j that passes through u and v. Since the edges of C_j belong to R, this contradicts the fact that a and b lie on the boundary of a common face of R. □

Now we can complete the proof of Part 2. For $i = 1, \ldots, k$, let W_i be the set of faces f in $\mathcal{F}$ such that $V(\partial f)$ intersects Z_i. By Claim 4, the W_i's are disjoint. Let $H_i = \bigcup_{f \in W_i} H_f$. By Claim 2, the H_i's are edge-disjoint. By Claim 3, H_i spans X_i. By the disjointness of the W_i's, no path in H_{i_1} crosses a path in H_{i_2} if $i_1 \neq i_2$. Since the connectivity requirements are $\{0, 1\}$, each H_i contains a forest F_i that satisfies the requirements $r_1^*(\cdot)$ among vertices of Z_i. The components of F_i span the sets in $\mathcal{X}_i$. The union of all these trees is a subgraph that, by Part 1, satisfies connectivity requirements $r(\cdot)$. This completes the proof of Part 2 and the reverse direction of Theorem 3.

5 A PTAS Framework for Connected Problems in Planar Graphs

In this section, we review the approach used in [5] to give a PTAS for the Steiner tree problem in planar graphs as we will use the same approach for this survivable network problem.

The framework relies on an algorithm for finding a subgraph MG of G, called the *mortar graph* [5]. The mortar graph spans Q and has total weight no more than $f(\epsilon)$ times the minimum weight of a Steiner tree in G spanning Q (and so has weight no more than $f(\epsilon) \cdot \mathrm{OPT}$ where OPT denotes the optimal value of the Steiner tree *or* the survivable network problem). The first step in constructing MG is to find an approximate Steiner tree and recursively augmenting this with short paths.

The mortar graph is a grid-like subgraph (the bold edges in Figure 4(a)). For each cell or face of the mortar graph, the subgraph of G enclosed by that face is called a *brick* (Figure 4(b)). The properties of bricks needed for this work are summarized by the following lemma.

Lemma 6 (from Lemma 4 [5]). *The boundary of a brick B, in counterclockwise order, is the concatenation of four paths $W_B \cup S_B \cup E_B \cup N_B$ such that:*

(B1) The set of edges $B \setminus \partial B$ is nonempty.
(B2) Every terminal of Q that is in B is on N_B or on S_B.
(B3) N_B and S_B are ϵ-short.

A path P in a graph G is ϵ-short if for every pair of vertices x and y on P, the distance from x to y along P is at most $(1+\epsilon)$ times the distance from x to y in G: $dist_P(x,y) \leq (1+\epsilon) dist_G(x,y)$.

The mortar graph and the bricks are building blocks of the structural properties required for designing an approximation scheme. In [5], it was demonstrated that there is a near-optimal Steiner tree whose interaction with the mortar graph is "simple". To formalize this notion (Theorem 4), we say that there is near-optimal Steiner tree that *joins* the boundary of each brick a small number of times. A *joining vertex* of graph H with a path P is a vertex of P that is the endpoint of an edge of $H \setminus P$. The intersection of a tree with a brick might not be connected, and so the theorem applies to forests inside bricks.

Theorem 4 (Structural property of bricks for $\{0,1\}$-edge connectivity, Theorem 4 [5]). *Let B be a plane graph with boundary $N \cup E \cup S \cup W$ satisfying the brick properties of Lemma 6. Let F be a subgraph of B. There is a forest $\tilde{F}$ of B with the following properties:*

(F1) If two vertices of $N \cup S$ are connected in F, then they are connected in $\tilde{F}$.
(F2) The number of joining vertices of $\tilde{F}$ with both N and S is at most $\alpha(\epsilon)$.
(F3) $\ell(\tilde{F}) \leq (1+c\epsilon)\ell(F)$.

In the above, $\alpha(\epsilon) = o(\epsilon^{-5.5})$ and c is a fixed constant.

This theorem is a key ingredient to the proof of correctness of the PTAS for Steiner tree and will be used in proving a similar theorem (Theorem 5) for the $\{0,1,2\}$-edge connectivity problem we solve here.

5.1 Approximation Scheme

The approximation scheme consists of the following steps. Only Step 5 depends on the specifics of the optimization problem, though Step 4 depends on a constant that comes out of the Structure Theorem for the problem.

Step 1: Find the mortar graph *MG*.
Step 2: Decompose *MG* into "parcels", subgraphs with the following properties:

(a) Each parcel consists of the boundaries of a disjoint set of faces of *MG*. Since each edge of *MG* belongs to the boundaries of exactly two faces, it follows that each edge belongs to at most two parcels.
(b) The weight of all boundary edges (those edges belonging to two parcels) is at most $(1/\eta)$*weight*(MG). We choose η so that this bound is $(\epsilon/2)$ *weight*(OPT).
(c) The planar dual of each parcel has a spanning tree of depth at most $\eta + 1$.

Each parcel P corresponds to a subgraph of G, namely the subgraph consisting of the bricks corresponding to the faces making up P. Let us refer to this subgraph as the *filled-in* version of P.
Step 3: Select a set of "artificial" terminals on the boundaries of parcels so that for each filled-in parcel, there is a feasible (with respect to original and artificial terminals) solution whose weight is at most the parcel's boundary plus the weight of the intersection of OPT with the filled-in parcel, and the union over all parcels of such feasible solutions is a feasible solution for the original graph.
Step 4: For each brick, designate as *portals* a constant number of vertices on the boundary of each brick. The constant is chosen, depending on the Structure Theorem, so that there exists a near-optimal feasible solution that is *portal-respecting*, i.e. passes through a portal whenever it passes from one face of MG to another.
Step 5: For each filled-in parcel, find a optimal portal-respecting solution. Output the union of these solutions.

Step 1 can be carried out in $O(n \log n)$ time. Details are in [5,4,16]. Step 2 can be done in linear time. It consists of doing breadth-first search in the planar dual of MG, and then applying a "shifting" technique in the tradition of Baker [1]. Details are in [5]. Step 3 uses the fact that each parcel's boundary consists of edge-disjoint, noncrossing cycles. If such a cycle strictly encloses an original terminal and does not enclose all terminals, a vertex on the cycle is designated an artificial terminal. Under this condition, any feasible solution for the original graph must cross the cycle; by adding the edges of the cycle, we get a feasible solution that also spans the artificial terminal. Step 3 can be implemented in linear time. Step 5 is achieved in linear time using dynamic programming.

Step 4 uses a simple greedy algorithm to designate portals along the boundary ∂B of a brick B so that there are at most $\theta + 1$ portals chosen, and that each vertex on the boundary is within distance at most $weight(\partial B)/\theta$ of some portal. We discuss the choice of θ presently.

5.2 Portal-Connected Graph

In order to make more precise the notion of a portal-respecting feasible solution, we introduce an auxiliary graph, the *portal-connected graph* (PCG) of a parcel. See Figure 4. Starting with a parcel (which consists of edges of the mortar graph), within each face, insert a duplicate of the brick corresponding to that face, and use artificial zero-weight edges to connect the occurrences of the portals in the duplicate brick to the occurrences of the same vertices in the parcel.[1]

A path P in the filled-in parcel from a vertex x interior to a brick B to a vertex y on the boundary of the brick B corresponds to a path $\tilde{P}$ in the PCG from x to the occurrence of y in the parcel; $\tilde{P}$ must take a detour through an artificial edge, and must therefore go through a portal. The increase is weight is at most $2weight(\partial B)/\theta$.

[1] Our usage of the term *PCG* differs slightly from that in [5], where the PCG was defined for the entire graph, not just a parcel.

The Structure Theorem states that the subgraph of OPT embedded strictly inside a brick can be modified so that it touches the boundary of the brick at no more than $\alpha(\epsilon)$ vertices. Rerouting each of these connections so it occurs at a portal incurs a weight of $2weight(\partial B)/\theta$, for a total of $2\alpha(\epsilon)weight(\partial B)/\theta$. The sum of boundary lengths of all bricks is twice the length of the mortar graph, which in turn is at most $f(\epsilon)$ times the value of OPT. The value of θ is chosen to ensure that the total rerouting weight is at most ϵ times the value of OPT. In order to find a nearly optimal solution in the filled-in parcel, therefore, it suffices to find an optimal solution in the PCG.

Recall that the planar dual of the parcel has a spanning tree of depth $\eta + 1$. Since each brick has at most $\theta + 1$ portals, it follows that the planar dual of the PCG has a spanning tree of depth at most $(\eta + 1)(\theta + 1)$. It follows that there is a rooted spanning tree of the PCG (the primal) such that, for each vertex v, there at most $2(\eta+1)(\theta+1)+1$ edges from descendents of v to non-descendents. This spanning tree is used to guide the dynamic program (Section 7).

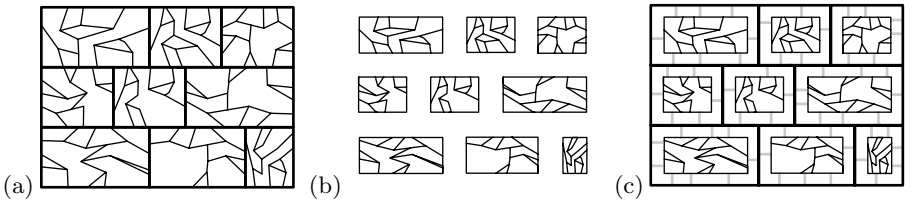


Fig. 4. The mortar graph in bold (a), the set of bricks (b), and the portal connected graph (c)

6 Applying the PTAS Framework

Theorem 4 applies directly to the Steiner-tree problem: the intersection of a tree with a brick is a forest and since the terminals are vertices of MG, it is enough to maintain connectivity between vertices on the boundary of a brick. However, for the 2-EC problem, the intersection of a solution with a brick has a more complicated structure.

In this section we prove the following counterpart to Theorem 4 that maintains up to 2 connectivity between vertices on the north and south boundary of a brick.

Theorem 5 (Structural property of bricks for $\{0,1,2\}$-edge connectivity). *Let B be a plane graph with boundary $N \cup E \cup S \cup W$ and satisfying the brick properties of Lemma 6. Let H be a subgraph of B. There is another subgraph $\widehat{H}$ that is the disjoint union of three forests $\widehat{F}_1, \widehat{F}_2, \widehat{F}_3$ of B with the following properties:*

(H1) $\widehat{H}$ achieves the 2-connectivity of H for vertices of $N \cup S$.
(H2) The number of joining vertices of $\widehat{H}$ with both N and S is at most $2\alpha(\epsilon)$.
(H3) $\ell(\widehat{H}) \leq (1 + c\epsilon)\ell(H)$.

In the above, $\alpha(\epsilon) = o(\epsilon^{-5.5})$ and c is a fixed constant.

Proof. The theorem is proved as follows. We first show that there is a subgraph H' of H that is the disjoint union of a set of trees $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ (where k can be very large) such that H' achieves the 2-connectivity of H (Theorem 3). We then show that we can partition this set of trees into three sets such that the disjoint union of each set is a forest. We then apply Theorem 4 to each of these forests, proving Theorem 5.

Let H' be a minimal subgraph of H such that H' achieves the 2-connectivity of H for vertices on $N \cup S$.

By the *only-if* direction of Theorem 3, H' is the union of a set of noncrossing edge-disjoint trees $\mathcal{T} = \{T_1, T_2, \ldots\}$, where each tree T_i achieves connectivity between a set X_i of vertices of $N \cup S$. Partition $\mathcal{T}$ into two sets:

$\mathcal{T}_1 = \{T_i \in \mathcal{T}$ such that $X_i \subseteq V(N)$ or $X_i \subseteq V(S)\}$.
$\mathcal{T}_2 = \{T_i \in \mathcal{T}$ such that X_i has vertices in both $V(N)$ and $V(S)\}$.

We further partition $\mathcal{T}_2$ into two sets. Let T_i and T_j be two trees in $\mathcal{T}_2$. Since T_i and T_j do not cross each other, if the vertices $X_i \cap V(S)$ appear before $X_j \cap V(S)$ along S then the vertices in $X_i \cap V(N)$ appear before $X_j \cap V(N)$ along N. It follows that there is an ordering of the trees in $\mathcal{T}_2$ from left to right in the brick, ordered according to the vertices in S to which they connect. Let $\mathcal{T}_A$ be the set of trees of $\mathcal{T}_2$ that are even-numbered in this ordering and let $\mathcal{T}_B$ be the set that are odd-numbered. That is, the trees in $\mathcal{T}_2$ alternate between $\mathcal{T}_A$ and $\mathcal{T}_B$.

Any two trees in $\mathcal{T}_A$ are separated by a tree in $\mathcal{T}_B$. Assume for a contradiction that a cycle was formed by some trees in $\mathcal{T}_A$. Then the cycle would have to strictly enclose an edge of a tree in $\mathcal{T}_B$. This contradicts Corollary 1. This shows that the trees in $\mathcal{T}_A$ form a forest. Similarly, the trees in $\mathcal{T}_B$ form a forest.

Consider a tree $T_i \in \mathcal{T}_1$. We describe how to select a corresponding tree $\widehat{T}_i$. Suppose that $X_i \subseteq V(N)$. Let $\widehat{T}$ be the minimal subpath of N that spans X_i. The case where $X_i \subseteq V(S)$ is analogous.

Let $\widehat{F}_1$ be the *disjoint* union of $\{\widehat{T} : T \in \mathcal{T}_1\}$. (That is, the multiplicity of an edge in $\widehat{F}_1$ is the sum of its multiplicities in $\{\widehat{T} : T \in \mathcal{T}_1\}$.) . Let $\widehat{F}_A$ be the forest guaranteed by Theorem 4 for the forest obtained by taking the union of the trees in $\mathcal{T}_A$. Similarly define $\widehat{F}_B$. Let $\widehat{H}$ be the union of $\widehat{F}_1, \widehat{F}_A, \widehat{F}_B$.

We show that $\widehat{H}$ achieves the required properties.

It is clear from the construction that $\widehat{F}_1$ does not have any joining vertices with N or S. By Theorem 4, each of $\widehat{F}_A$ and $\widehat{F}_B$ has at most $\alpha(\epsilon)$ joining vertices with $N \cup S$. Therefore $\widehat{H}$ has at most $2\alpha(\epsilon)$ joining vertices with $N \cup S$, proving Property H2.

Since N and S are ϵ-short paths, $\ell(\widehat{F}_1)$ is at most $1+\epsilon$ times the total length of all trees in $\mathcal{T}_1$. By Theorem 4, $\ell(\widehat{F}_A) \leq (1+c\epsilon)\ell(\widehat{F}_A)$ and $\ell(\widehat{F}_B) \leq (1+c\epsilon)\ell(\widehat{F}_B)$. It follows that $\ell(\widehat{H}) \leq (1 + c\epsilon)\ell(H)$, proving Property H3.

We now show that if two vertices of $N \cup S$ are 2-edge connected in H', then they are 2-edge connected in $\widehat{H}$. Showing this for 1-edge connectivity is simpler; the argument is omitted here. This will complete the proof. We were particular

in partitioning the trees into forests $\mathcal{T}_1, \mathcal{T}_A, \mathcal{T}_B$ because applying Theorem 4 to a tree with two edges incident to a vertex $v \in \partial B$ could result in a tree with only one edge incident v. This could remove edge connectivity.

Let a and b be vertices of $N \cup S$ that are 2-edge connected in H'. Let C be the minimal cycle 2-connecting x and y as guaranteed by Lemma 4 and let $Y = V(C \cap (N \cup S))$. Let $y_1, y_2, \ldots, y_k$ be the order of the vertices of Y along the boundary of the brick. Let X_i be the set such that $y_i, y_{i+1} \in X_i$ and $X_i \subseteq V(\partial B[y_i, y_{i+1}])$ (as guaranteed by the construction given in Theorem 3). There are two cases:

$Y \subseteq V(N)$ or $Y \subseteq V(S)$: Without loss of generality, assume that y_1 is the first vertex and y_k is the last vertex of Y along N. Then $X_1, \ldots, X_{k-1}$ are subsets of N. X_k may contain vertices of S. Let $\widehat{T}_i$ be a tree in $\widehat{F}_1$ that spans X_i (for $i = 1, \ldots, k-1$). Since $\widehat{F}_1$ is the disjoint union of these trees, there is a path P in $\widehat{F}_1$ that visits each vertex $y_1, \ldots, y_k$ in order. If X_k spans a vertex of S then $X_k \in F_A$ (without loss of generality). The vertices X_k are spanned by $\widehat{F}_A$ and so there is a y_k-to-y_1 path Q in $\widehat{H}$ that is edge disjoint from P. $P \circ Q$ is a cycle such that $Y \subseteq V(P \cup Q)$. The vertices in Y are 2-edge connected in $\widehat{H}$.

$Y \cap V(N) \neq \emptyset$ and $Y \cap V(S) \neq \emptyset$: Without loss of generality, assume that y_1 and y_l are the first and last vertices of Y along N. Then y_k and y_{l+1} are the first and last vertices of Y along S. By the argument used in the above case, there is a path P in $\widehat{H}$ that visits the vertices $y_1, \ldots, y_l$ in order. Likewise, there is a path Q in $\widehat{H}$ that visits the vertices $y_{l+1}, \ldots, y_k$ in order. We now argue that there are edge-disjoint y_l-to-y_{l+1} and y_k-to-y_1 paths in $\widehat{H}$ by showing that T_l (the tree corresponding to X_l) is in F_A and T_k (the tree corresponding to X_k) is in F_B: by Lemma 1, there are no trees enclosed by C in H', so T_l and T_k are ordered sequentially in $\mathcal{T}_B$. □

7 Dynamic Program

Here we give an outline of the dynamic program used to find an optimal solution in each *filled-in parcel*. As discussed at the end of Section 5.2, we use a rooted tree such that, for each vertex v, there at most $2(\eta+1)(\theta+1)+1$ edges from descendents of v to non-descendents. Each vertex gives rise to a subproblem in the dynamic program. Both θ and η depend polynomially on $1/\epsilon$. The interaction between two subproblems is limited to this set of edges. Each brick in the brick decomposition corresponds to a base case of the dynamic program. All other base cases are trivial, corresponding to single vertices in our input graph.

For each subproblem, we consider all possible $\{0, 1, 2\}$-connectivity patterns (or *configurations*) on the vertex set U. (A configuration is given by a forest with no degree-2 vertices whose vertices correspond to 2-edge connected components and whose edges correspond to adjacency between these components. Such a forest corresponds to a block-cut tree of the solution it encodes.) The leaves of the forests are identified with edges in the cut corresponding to the vertex set U.

Since there are $O(\theta\eta)$ edges in the cut, there are at most $O((\theta\eta)(\theta\eta)^{\theta\eta})$ forests representing configurations (by way of Cayley's formula).

It remains to show that we can solve a base case corresponding to a brick. The number of edges between a brick and the rest of the parcel is the number of portal edges, η, that connects the brick in the filled-in parcel. A configuration for the brick is a set of 2-connectivity requirements between the portal edges. Given such a set of requirements, we can use the algorithm implied by Theorem 3 to find a set of subsets $\mathcal{X}$ of the portal edges such that independently connecting each set in $\mathcal{X}$ will satisfy the given 2-connectivity requirements (Theorem 3). For each set in $\mathcal{X}$, we find the minimum-length Steiner tree using the algorithm of Erickson et al. [7]. For a constant number of terminals, using the algorithm of [11], this algorithm can be implemented to run in linear time. The resulting running time of the dynamic program, including the dependence on ϵ is $2^{o(\epsilon^{-9.5})}n$.

Comments

The PTAS framework used is potentially applicable to problems where (i) the input consists of a planar graph G with edge-weights and a subset Q of the vertices of G (we call Q the set of *terminals*), and where (ii) the output spans the terminals. Steiner tree and two-edge connectivity have been solved using this framework. The PTAS for the subset tour problem [16] (which was the inspiration for this framework) can be reframed using this technique. Recently, with David Pritchard, we have extended this work to give a PTAS for the $\{0, 1, \ldots, k\}$-edge connectivity problem in planar multigraphs. Details will follow in a longer version.

References

1. Baker, B.: Approximation algorithms for NP-complete problems on planar graphs. J. ACM 41(1), 153–180 (1994)
2. Berger, A., Czumaj, A., Grigni, M., Zhao, H.: Approximation schemes for minimum 2-connected spanning subgraphs in weighted planar graphs. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 472–483. Springer, Heidelberg (2005)
3. Berger, A., Grigni, M.: Minimum weight 2-edge-connected spanning subgraphs in planar graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 90–101. Springer, Heidelberg (2007)
4. Borradaile, G., Kenyon-Mathieu, C., Klein, P.: A polynomial-time approximation scheme for Steiner tree in planar graphs. In: 18th SODA, pp. 1285–1294 (2007)
5. Borradaile, G., Klein, P., Mathieu, C.: Steiner tree in planar graphs: An $O(n \log n)$ approximation scheme with singly exponential dependence on epsilon. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 275–286. Springer, Heidelberg (2007)
6. Czumaj, A., Lingas, A.: On approximability of the minimum cost k-connected spanning subgraph problem. In: 10th SODA, pp. 281–290 (1999)
7. Erickson, R., Monma, C., Veinott, A.: Send-and-split method for minimum-concave-cost network flows. Math. Op. Res. 12, 634–664 (1987)

8. Eswaran, K., Tarjan, R.: Augmentation problems. SIAM J. Comput. 5(4), 653–665 (1976)
9. Frederickson, G., Jájá, J.: Approximation algorithms for several graph augmentation problems. SIAM J. Comput. 10(2), 270–283 (1981)
10. Goemans, M., Goldberg, A., Plotkin, S., Shmoys, D., Tardos, É., Williamson, D.: Improved approximation algorithms for network design problems. In: 5th SODA, pp. 223–232 (1994)
11. Henzinger, M., Klein, P., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. J. Comput. System Sci. 55(1), 3–23 (1997)
12. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21(1), 39–60 (2001)
13. Jothi, R., Raghavachari, B., Varadarajan, S.: A 5/4-approximation algorithm for minimum 2-edge-connectivity. In: 14th SODA, pp. 725–734 (2003)
14. Khuller, S., Vishkin, U.: Biconnectivity approximations and graph carvings. J. ACM 41(2), 214–235 (1994)
15. Klein, P.: A linear-time approximation scheme for planar weighted TSP. In: 46th FOCS, pp. 647–647 (2005)
16. Klein, P.: A subset spanner for planar graphs, with application to subset TSP. In: 38th STOC, pp. 749–756 (2006)
17. Klein, P., Ravi, R.: When cycles collapse: A general approximation technique for constraind two-connectivity problems. In: 3rd IPCO, pp. 39–55 (1993)
18. Ravi, R.: Approximation algorithms for Steiner augmentations for two-connectivity. Technical Report TR-CS-92-21, Brown University (1992)
19. Resende, M., Pardalos, P. (eds.): Handbook of Optimization in Telecommunications. Springer, Heidelberg (2006)
20. Williamson, D., Goemans, M., Mihail, M., Vazirani, V.: A primal-dual approximation algorithm for generalized Steiner network problems. In: 35th STOC, pp. 708–717 (1993)

Efficiently Testing Sparse $GF(2)$ Polynomials

Ilias Diakonikolas, Homin K. Lee, Kevin Matulef,
Rocco A. Servedio, and Andrew Wan

{ilias,homin,rocco,atw12}@cs.columbia.edu, matulef@mit.edu

Abstract. We give the first algorithm that is both query-efficient and time-efficient for testing whether an unknown function $f : \{0,1\}^n \to \{0,1\}$ is an s-sparse $GF(2)$ polynomial versus ϵ-far from every such polynomial. Our algorithm makes $\mathrm{poly}(s, 1/\epsilon)$ black-box queries to f and runs in time $n \cdot \mathrm{poly}(s, 1/\epsilon)$. The only previous algorithm for this testing problem [DLM+07] used $\mathrm{poly}(s, 1/\epsilon)$ queries, but had running time exponential in s and super-polynomial in $1/\epsilon$.

Our approach significantly extends the "testing by implicit learning" methodology of [DLM+07]. The learning component of that earlier work was a brute-force exhaustive search over a concept class to find a hypothesis consistent with a sample of random examples. In this work, the learning component is a sophisticated exact learning algorithm for sparse $GF(2)$ polynomials due to Schapire and Sellie [SS96]. A crucial element of this work, which enables us to simulate the membership queries required by [SS96], is an analysis establishing new properties of how sparse $GF(2)$ polynomials simplify under certain restrictions of "low-influence" sets of variables.

1 Introduction

Background and motivation. Given black-box access to an unknown function $f : \{0,1\}^n \to \{0,1\}$, a natural question to ask is whether the function has a particular form. Is it representable by a small decision tree, or small circuit, or sparse polynomial? In the field of computational learning theory, the standard approach to this problem is to assume that f belongs to a specific class $\mathcal{C}$ of functions of interest, and the goal is to identify or approximate f. In contrast, in property testing nothing is assumed about the unknown function f, and the goal of the testing algorithm is to output "yes" with high probability if $f \in \mathcal{C}$ and "no" with high probability if f is ϵ-far from every $g \in \mathcal{C}$. (Here the distance between two functions f, g is measured with respect to the uniform distribution on $\{0,1\}^n$, so f and g are ϵ-far if they disagree on more than an ϵ fraction of all inputs.) The complexity of a testing algorithm is measured both in terms of the number of black-box queries it makes to f (*query complexity*) as well as the time it takes to process the results of those queries (*time complexity*).

There are many connections between learning theory and testing, and a growing body of work relating the two fields (see [Ron07] and its references). Testing algorithms have been given for a range of different function classes such as linear functions over $GF(2)$ (i.e. parities) [BLR93]; degree-d $GF(2)$ polynomials [AKK+03]; Boolean literals, conjunctions, and s-term monotone DNF formulas [PRS02]; k-juntas (i.e. functions which depend on at most k variables) [FKR+04]; halfspaces [MORS07]; and more.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 502–514, 2008.

Recently, Diakonikolas et al. [DLM+07] gave a general technique, called "testing by implicit learning," which they used to test a variety of different function classes that were not previously known to be testable. Intuitively, these classes correspond to functions with "concise representations," such as s-term DNFs, size-s Boolean formulas, size-s Boolean circuits, and s-sparse polynomials over constant-size finite fields. For each of these classes, the testing algorithm of [DLM+07] makes only $\text{poly}(s, 1/\epsilon)$ queries (independent of n).

The main drawback of the [DLM+07] testing algorithm is its time complexity. For each of the classes mentioned above, the algorithm's running time is $2^{\omega(s)}$ as a function of s, and $\omega(\text{poly}(1/\epsilon))$ as a function of ϵ.[1] Thus, a natural question asked by [DLM+07] is whether any of these classes can be tested with both time complexity and query complexity $\text{poly}(s, 1/\epsilon)$.

Our result: efficiently testing sparse $GF(2)$ polynomials. In this paper we focus on the class of *s-sparse polynomials over $GF(2)$*. Polynomials over $GF(2)$ (equivalently, parities of ANDs of input variables) are a simple and well-studied representation for Boolean functions. It is well known that every Boolean function has a unique representation as a multilinear polynomial over $GF(2)$, so the sparsity (number of monomials) of this polynomial is a very natural measure of the complexity of f. Sparse $GF(2)$ polynomials have been studied by many authors from a range of different perspectives such as learning [BS90, FS92, SS96, Bsh97a, BM02], approximation and interpolation [Kar89, GKS90, RB91], the complexity of (approximate) counting [EK89, KL93, LVW93], and property testing [DLM+07].

The main result of this paper is a testing algorithm for s-sparse $GF(2)$ polynomials that is both time-efficient and query-efficient:

Theorem 1. *There is a $\text{poly}(s, 1/\epsilon)$-query algorithm with the following performance guarantee: given parameters s, ϵ and black-box access to any $f : \{0,1\}^n \to \{0,1\}$, it runs in time $\text{poly}(s, 1/\epsilon)$ and tests whether f is an s-sparse $GF(2)$ polynomial versus ϵ-far from every s-sparse polynomial.*

This answers the question of [DLM+07] by exhibiting an interesting and natural class of functions with "concise representations" that can be tested efficiently, both in terms of query complexity and running time.

We obtain our main result by extending the "testing by implicit learning" approach of [DLM+07]. In that work the "implicit learning" step used a naive brute-force search for a consistent hypothesis; in this paper we employ a sophisticated proper learning algorithm due to Schapire and Sellie [SS96]. It is much more difficult to "implicitly" run the [SS96] algorithm than the brute-force search of [DLM+07]. One of the main technical contributions of this paper is a new structural theorem about how s-sparse $GF(2)$ polynomials are affected by certain carefully chosen restrictions; this is an essential ingredient that enables us to use the [SS96] algorithm. We elaborate on this below.

[1] We note that the algorithm also has a linear running time dependence on n, the number of input variables; this is in some sense inevitable since the algorithm must set n bit values just to pose a black-box query to f. Our algorithm has running time linear in n for the same reason. For the rest of the paper we discuss the running time only as a function of s and ϵ.

Techniques. We begin with a brief review of the main ideas of [DLM+07]. The approach of [DLM+07] builds on the observation of Goldreich et al. [GGR98] that any *proper* learning algorithm for a function class $\mathcal{C}$ can be used as a testing algorithm for $\mathcal{C}$. (Recall that a proper learning algorithm for $\mathcal{C}$ is one which outputs a hypothesis h that itself belongs to $\mathcal{C}$.) The idea behind this observation is that if the function f being tested belongs to $\mathcal{C}$ then a proper learning algorithm will succeed in constructing a hypothesis that is close to f, while if f is ϵ-far from every $g \in \mathcal{C}$ then any hypothesis $h \in \mathcal{C}$ that the learning algorithm outputs must necessarily be far from f. Thus any class $\mathcal{C}$ can be tested to accuracy ϵ using essentially the same number of queries that are required to properly learn the class to accuracy $\Theta(\epsilon)$.

The basic approach of [GGR98] did not yield query-efficient testing algorithms (with query complexity independent of n) since virtually every interesting class of functions over $\{0,1\}^n$ requires $\Omega(\log n)$ examples for proper learning. However, [DLM+07] showed that for many classes of functions defined by a size parameter s, it is possible to "implicitly" run a (very naive) proper learning algorithm over a number of variables that is independent of n, and thus obtain an overall query complexity independent of n. More precisely, they first observed that for many classes $\mathcal{C}$ every $f \in \mathcal{C}$ is "very close" to a function $f' \in \mathcal{C}$ for which the number r of relevant variables is polynomial in s and independent of n; roughly speaking, the relevant variables for f' are the variables that have high influence in f. (For example, if f is an s-sparse $GF(2)$ polynomial, an easy argument shows that there is a function f' – obtained by discarding from f all monomials of degree more than $\log(s/\tau)$ – that is τ-close to f and depends on at most $r = s\log(s/\tau)$ variables.) They then showed how, using ideas of Fischer et al. [FKR+04] for testing juntas, it is possible to construct a sample of uniform random examples over $\{0,1\}^r$ which with high probability are all labeled according to f'. At this point, the proper learning algorithm employed by [DLM+07] was a naive brute-force search. The algorithm tried all possible functions in $\mathcal{C}$ over r (as opposed to n) variables, to see if any were consistent with the labeled sample. [DLM+07] thus obtained a testing algorithm with overall query complexity $\mathrm{poly}(s/\epsilon)$ but whose running time was dominated by the brute-force search. For the class of s-sparse $GF(2)$ polynomials, their algorithm used $\tilde{O}(s^4/\epsilon^2)$ queries but had running time at least $2^{\omega(s)} \cdot (1/\epsilon)^{\log\log(1/\epsilon)}$.

Current approach. The high-level idea of the current work is to employ a much more sophisticated – and efficient – proper learning algorithm than brute-force search. In particular we would like to use a proper learning algorithm which, when applied to learn a function over only r variables, runs in time polynomial in r and in the size parameter s. For the class of s-sparse $GF(2)$ polynomials, precisely such an algorithm was given by Schapire and Sellie [SS96]. Their algorithm, which we describe in Section 2.1, is computationally efficient and generates a hypothesis h which is an s-sparse $GF(2)$ polynomial. But this power comes at a price: the algorithm requires access to a *membership query* oracle, i.e. a black-box oracle for the function being learned. Thus, in order to run the Schapire/Sellie algorithm in the "testing by implicit learning" framework, it is necessary to simulate membership queries to an approximating function $f' \in \mathcal{C}$ which is close to f but depends on only r variables. This is significantly more challenging than generating uniform random examples labeled according to f', which is all that is required in the original [DLM+07] approach.

To see why membership queries to f' are more difficult to simulate than uniform random examples, recall that f and the f' described above (obtained from f by discarding high-degree monomials) are τ-close. Intuitively this is extremely close, disagreeing only on a $1/m$ fraction of inputs for an m that is much larger than the number of random examples required for learning f' via brute-force search (this number is "small" – independent of n – because f' depends on only r variables). Thus in the [DLM+07] approach it suffices to use f, the function to which we actually have black-box access, rather than f' to label the random examples used for learning f'; since f and f' are so close, and the examples are uniformly random, with high probability all the labels will also be correct for f'. However, in the membership query scenario of the current paper, things are no longer that simple. For any given f' which is close to f, one can no longer assume that the learning algorithm's queries to f' are uniformly distributed and hence unlikely to hit the error region – indeed, it is possible that the learning algorithm's membership queries to f' are clustered on the few inputs where f and f' disagree.

In order to successfully simulate membership queries, we must somehow consistently answer queries according to a particular f', even though we only have oracle access to f. Moreover this must be done implicitly in a query-efficient way, since explicitly identifying even a single variable relevant to f' requires at least $\Omega(\log n)$ queries. This is the main technical challenge in the paper.

We meet this challenge by showing that for any s-sparse polynomial f, an approximating f' can be obtained as a restriction of f by setting certain carefully chosen subsets of variables to zero. Roughly speaking, this restriction is obtained by randomly partitioning all of the input variables into r subsets and zeroing out all subsets whose variables have small "collective influence" (more precisely, small variation in the sense of [FKR+04]). It is important that the restriction sets these variables to zero, rather than a random assignment; intuitively this is because setting a variable to zero "kills" all monomials that contain the variable, whereas setting it to 1 does not. Our main technical theorem (Theorem 3, given in Section 3) shows that this f' is indeed close to f and has at most one of its relevant variables in each of the surviving subsets. We moreover show that these relevant variables for f' all have high influence in f (the converse is not true; examples can be given which show that not every variable that has "high influence" in f will in general become a relevant variable for f'). This property is important in enabling our simulation of membership queries. In addition to the crucial role that Theorem 3 plays in the completeness proof for our test, we feel that the new insights the theorem gives into how sparse polynomials "simplify" under (appropriately defined) random restrictions may be of independent interest.

Organization. In Section 4, we present our testing algorithm, **Test-Sparse-Poly**, along with a high-level description and sketch of correctness. In Section 2.1 we describe in detail the "learning component" of the algorithm. In Section 3 we state Theorem 3, which provides intuition behind the algorithm and serves as the main technical tool in the completeness proof. Due to space limitations, the proof of Theorem 3 is presented in Appendix A, while the completeness and soundness proofs are given in Appendices B and C, respectively (see full version available online).

2 Preliminaries and Background

***GF*(2) Polynomials:** A $GF(2)$ polynomial is a parity of monotone conjunctions (monomials). It is *s-sparse* if it contains at most s monomials (including the constant-1 monomial if it is present). The *length* of a monomial is the number of distinct variables that occur in it; over $GF(2)$, this is simply its degree.

Notation: For $i \in \mathbb{N}^*$, denote $[i] \stackrel{\text{def}}{=} \{1, 2, \ldots, i\}$. It will be convenient to view the output range of a Boolean function f as $\{-1, 1\}$ rather than $\{0, 1\}$, i.e. $f : \{0,1\}^n \to \{-1, 1\}$. We view the hypercube as a measure space endowed with the uniform product probability measure. For $I \subseteq [n]$ we denote by $\{0,1\}^I$ the set of all partial assignments to the coordinates in I. For $w \in \{0,1\}^{[n]\setminus I}$ and $z \in \{0,1\}^I$, we write $w \sqcup z$ to denote the assignment in $\{0,1\}^n$ whose i-th coordinate is w_i if $i \in [n] \setminus I$ and is z_i if $i \in I$. Whenever an element z in $\{0,1\}^I$ is chosen randomly (we denote $z \in_{\mathrm{R}} \{0,1\}^I$), it is chosen with respect to the uniform measure on $\{0,1\}^I$.

Influence, Variation and the Independence Test: Recall the classical notion of *influence* [KKL88]: The *influence* of the i-th coordinate on $f : \{0,1\}^n \to \{-1,1\}$ is $\mathrm{Inf}_i(f) \stackrel{\text{def}}{=} \Pr_{x \in_{\mathrm{R}} \{0,1\}^n}[f(x) \neq f(x^{\oplus i})]$, where $x^{\oplus i}$ denotes x with the i-th bit flipped. The following generalization of influence, the *variation* of a subset of the coordinates of a Boolean function, plays an important role for us:

Definition 1 (variation, [FKR+04]). *Let $f : \{0,1\}^n \to \{-1,1\}$, and let $I \subseteq [n]$. We define the* variation *of f on I as* $\mathrm{Vr}_f(I) \stackrel{def}{=} \mathbb{E}_{w \in_R \{0,1\}^{[n]\setminus I}}\left[\mathbb{V}_{z \in_R \{0,1\}^I}[f(w \sqcup z)]\right]$.

When $I = \{i\}$ we will sometimes write $\mathrm{Vr}_f(i)$ instead of $\mathrm{Vr}_f(\{i\})$. It is easy to check that $\mathrm{Vr}_f(i) = \mathrm{Inf}_i(f)$, so variation is indeed a generalization of influence. Intuitively, the variation is a measure of the ability of a set of variables to sway a function's output. The following two simple properties of the variation will be useful for the analysis of our testing algorithm:

Lemma 1 (monotonicity and sub-additivity, [FKR+04]). *Let $f : \{0,1\}^n \to \{-1,1\}$ and $A, B \subseteq [n]$. Then $\mathrm{Vr}_f(A) \leq \mathrm{Vr}_f(A \cup B) \leq \mathrm{Vr}_f(A) + \mathrm{Vr}_f(B)$.*

Lemma 2 (probability of detection, [FKR+04]). *Let $f : \{0,1\}^n \to \{-1,1\}$ and $I \subseteq [n]$. If $w \in_R \{0,1\}^{[n]\setminus I}$ and $z_1, z_2 \in_R \{0,1\}^I$ are chosen independently, then $\Pr[f(w \sqcup z_1) \neq f(w \sqcup z_2)] = \frac{1}{2}\mathrm{Vr}_f(I)$.*

We now recall the *independence test* from [FKR+04], a simple two query test used to determine whether a function f is independent of a given set $I \subseteq [n]$ of coordinates.

Independence test: Given $f : \{0,1\}^n \to \{-1,1\}$ and $I \subseteq [n]$, choose $w \in_{\mathrm{R}} \{0,1\}^{[n]\setminus I}$ and $z_1, z_2 \in_{\mathrm{R}} \{0,1\}^I$ independently. Accept if $f(w \sqcup z_1) = f(w \sqcup z_2)$ and reject if $f(w \sqcup z_1) \neq f(w \sqcup z_2)$.

Lemma 2 implies that the independence test rejects with probability exactly $\frac{1}{2}\mathrm{Vr}_f(I)$.

Random Partitions: Throughout the paper we will use the following notion of a random partition of the set $[n]$ of input coordinates:

Definition 2. *A* random partition *of* $[n]$ *into* r *subsets* $\{I_j\}_{j=1}^r$ *is constructed by independently assigning each* $i \in [n]$ *to a randomly chosen* I_j *for some* $j \in [r]$.

We now define the notion of low- and high-variation subsets with respect to a partition of the set $[n]$ and a parameter $\alpha > 0$.

Definition 3. *For* $f : \{0,1\}^n \to \{-1,1\}$, *a partition of* $[n]$ *into* $\{I_j\}_{j=1}^r$ *and a parameter* $\alpha > 0$, *define* $L(\alpha) \stackrel{def}{=} \{j \in [r] \mid \mathrm{Vr}_f(I_j) < \alpha\}$ *(low-variation subsets) and* $H(\alpha) \stackrel{def}{=} [r] \setminus L(\alpha)$ *(high-variation subsets). For* $j \in [r]$ *and* $i \in I_j$, *if* $\mathrm{Vr}_f(i) \geq \alpha$ *we say that the variable* x_i *is a* high-variation element *of* I_j.

Finally, the notion of a *well-structured* subset will be important for us:

Definition 4. *For* $f : \{0,1\}^n \to \{-1,1\}$ *and parameters* $\alpha > \Delta > 0$, *we say that a subset* $I \subseteq [n]$ *of coordinates is* (α, Δ)-well structured *if there is an* $i \in I$ *such that* $\mathrm{Vr}_f(i) \geq \alpha$ *and* $\mathrm{Vr}_f(I \setminus \{i\}) \leq \Delta$.

Note that since $\alpha > \Delta$, by monotonicity, the $i \in I$ in the above definition is unique. Hence, a well-structured subset contains a single high-influence coordinate, while the remaining coordinates have small total variation.

2.1 Background on Schapire and Sellie's Algorithm

In [SS96] Schapire and Sellie gave an algorithm, which we refer to as **LearnPoly**, for exactly learning s-sparse $GF(2)$ polynomials using membership queries (i.e. black-box queries) and equivalence queries. Their algorithm is *proper*; this means that every equivalence query the algorithm makes (including the final hypothesis of the algorithm) is an s-sparse polynomial. (We shall see that it is indeed crucial for our purposes that the algorithm is proper.) Recall that in an equivalence query the learning algorithm proposes a hypothesis h to the oracle: if h is logically equivalent to the target function being learned then the response is "correct" and learning ends successfully, otherwise the response is "no" and the learner is given a counterexample x such that $h(x) \neq f(x)$.

Schapire and Sellie proved the following about their algorithm:

Theorem 2. *[[SS96], Theorem 10] Algorithm* **LearnPoly** *is a proper exact learning algorithm for the class of* s*-sparse* $GF(2)$ *polynomials over* $\{0,1\}^n$. *The algorithm runs in poly*(n, s) *time and makes at most* $\mathrm{poly}(n, s)$ *membership queries and at most* $ns + 2$ *equivalence queries.*

We can easily also characterize the behavior of **LearnPoly** if it is run on a function f that is not an s-sparse polynomial. In this case, since the algorithm is proper all of its equivalence queries have s-sparse polynomials as their hypotheses, and consequently no equivalence query will ever be answered "correct." So if the $(ns+2)$-th equivalence query is not answered "correct," the algorithm may infer that the target function is not an s-sparse polynomial, and it returns "not s-sparse."

A well-known result due to Angluin [Ang88] says that in a Probably Approximately Correct or PAC setting (where there is a distribution $\mathcal{D}$ over examples and the goal is to construct an ϵ-accurate hypothesis with respect to that distribution), equivalence

queries can be straightforwardly simulated using random examples. This is done simply by drawing a sufficiently large sample of random examples for each equivalence query and evaluating both the hypothesis h and the target function f on each point in the sample. This either yields a counterexample (which simulates an equivalence query), or if no counterexample is obtained then simple arguments show that for a large enough ($O(\log(1/\delta)/\epsilon)$-size) sample, with probability $1-\delta$ the functions f and h must be ϵ-close under the distribution $\mathcal{D}$, which is the success criterion for PAC learning. This directly gives the following corollary of Theorem 2:

Corollary 1. *There is a uniform distribution membership query proper learning algorithm, which we call* **LearnPoly**$'(s, n, \epsilon, \delta)$*, which makes* $Q(s, n, \epsilon, \delta) \stackrel{def}{=} \mathrm{poly}(s, n, 1/\epsilon, \log(1/\delta))$ *membership queries and runs in* $\mathrm{poly}(Q)$ *time to learn* s*-sparse polynomials over* $\{0,1\}^n$ *to accuracy* ϵ *and confidence* $1-\delta$ *under the uniform distribution.*

3 On Restrictions Which Simplify Sparse Polynomials

This section presents Theorem 3, which gives the intuition behind our testing algorithm, and lies at the heart of the completeness proof. We give the full proof of Theorem 3 in Appendix A (see the full version).

Roughly speaking, the theorem says the following: consider any s-sparse $GF(2)$ polynomial p. Suppose that its coordinates are randomly partitioned into $r = \mathrm{poly}(s)$ many subsets $\{I_j\}_{j=1}^r$. The first two statements say that w.h.p. a randomly chosen "threshold value" $\alpha \approx 1/\mathrm{poly}(s)$ will have the property that no single coordinate i, $i \in [n]$, or subset I_j, $j \in [r]$, has $\mathrm{Vr}_p(i)$ or $\mathrm{Vr}_p(I_j)$ "too close" to α. Moreover, the high-variation subsets (w.r.t. α) are precisely those that contain a single high variation element i (i.e. $\mathrm{Vr}_p(i) \geq \alpha$), and in fact each such subset I_j is well-structured (part 3). Also, the number of such high-variation subsets is small (part 4). Finally, let p' be the restriction of p obtained by setting all variables in the low-variation subsets to 0. Then, p' has a nice structure: it has at most one relevant variable per high-variation subset (part 5), and it is close to p (part 6).

Theorem 3. *Let* $p : \{0,1\}^n \to \{-1,1\}$ *be an* s*-sparse polynomial. Fix* $\tau \in (0,1)$ *and* Δ *such that* $\Delta \leq \Delta_0 \stackrel{def}{=} \tau/(1600 s^3 \log(8s^3/\tau))$ *and* $\Delta = \mathrm{poly}(\tau/s)$*. Let* $r \stackrel{def}{=} 4Cs/\Delta$*, for a suitably large constant* C*. Let* $\{I_j\}_{j=1}^r$ *be a random partition of* $[n]$*. Choose* α *uniformly at random from the set* $\mathcal{A}(\tau, \Delta) \stackrel{def}{=} \{\frac{\tau}{4s^2} + (8\ell - 4)\Delta : \ell \in [K]\}$ *where* K *is the largest integer such that* $8K\Delta \leq \frac{\tau}{4s^2}$*. Then with probability at least* $9/10$ *(over the choice of* α *and* $\{I_j\}_{j=1}^r$*), all of the following statements hold:*

1. *Every variable* x_i*,* $i \in [n]$*, has* $\mathrm{Vr}_p(i) \notin [\alpha - 4\Delta, \alpha + 4\Delta]$.
2. *Every subset* I_j*,* $j \in [r]$*, has* $\mathrm{Vr}_p(I_j) \notin [\alpha - 3\Delta, \alpha + 4\Delta]$.
3. *For every* $j \in H(\alpha)$*,* I_j *is* (α, Δ)*-well structured.*
4. $|H(\alpha)| \leq s \log(8s^3/\tau)$.

Let $p' \stackrel{def}{=} p|_{0 \leftarrow \cup_{j \in L(\alpha)} I_j}$ *(the restriction obtained by fixing all variables in low-variation subsets to* 0*).*

5. *For every $j \in H(\alpha)$, p' has at most one relevant variable in I_j (hence p' is a $|H(\alpha)|$-junta).*
6. *The function p' is τ-close to p.*

Theorem 3 naturally suggests a testing algorithm, whereby we attempt to partition the coordinates of a function f into "high-variation" subsets and "low-variation" subsets, then zero-out the variables in low-variation subsets and implicitly learn the remaining function f' on only $\mathrm{poly}(s, 1/\epsilon)$ many variables. This is exactly the approach we take in the next section.

4 The Testing Algorithm Test-Sparse-Poly

In this section we present our main testing algorithm and give high-level sketches of the arguments establishing its completeness and soundness. The algorithm, which is called **Test-Sparse-Poly**, takes as input the values s, $\epsilon > 0$ and black-box access to $f : \{0,1\}^n \to \{-1,1\}$. It is presented in full in Figure 1.

The first thing **Test-Sparse-Poly** does (Step 2) is randomly partition the coordinates into $r = \tilde{O}(s^4/\tau)$ subsets. In Steps 3 and 4 the algorithm attempts to distinguish subsets that contain a high-influence variable from subsets that do not; this is done by using the independence test to estimate the variation of each subset (see Lemma 2).

Once the high-variation and low-variation subsets have been identified, intuitively we would like to focus our attention on the high-influence variables. Thus, Step 5 of the algorithm defines a function $\widetilde{f}'$ which "zeroes out" all of the variables in all low-variation subsets. Step 6 of **Test-Sparse-Poly** checks that f is close to $\widetilde{f}'$.

The final step of **Test-Sparse-Poly** is to run the algorithm **LearnPoly**′ of [SS96] to learn a sparse polynomial, which we call $\widetilde{f}''$, which is isomorphic to $\widetilde{f}'$ but is defined only over the high-influence variables of f (recall that if f is indeed s-sparse, there is at most one from each high-variation subset). The overall **Test-Sparse-Poly** algorithm accepts f if and only if **LearnPoly**′ successfully returns a final hypothesis (i.e. does not halt and output "fail"). The membership queries that the [SS96] algorithm requires are simulated using the **SimMQ** procedure, which in turn uses a subroutine called **Set-High-Influence-Variables**.

The procedure **Set-High-Influence-Variable** (**SHIV**) is presented in Figure 2. The idea of this procedure is that when it is run on a well-structured subset of variables I, it returns an assignment in which the high-variation variable is set to the desired bit value. Intuitively, the executions of the independence test in the procedure are used to determine whether the high-variation variable $i \in I$ is set to 0 or 1 under the assignment x. Depending on whether this setting agrees with the desired value, the algorithm either returns x or the bitwise negation of x (this is slightly different from **Construct-Sample**, the analogous subroutine in [DLM+07], which is content with a random x and thus never needs to negate coordinates).

Figure 3 gives the **SimMQ** procedure. When run on a function f and a collection $\{I_j\}_{j \in H}$ of disjoint well-structured subsets of variables, **SimMQ** takes as input a string z of length $|H|$ which specifies a desired setting for each high-variation variable in each I_j ($j \in H$). **SimMQ** constructs a random assignment $x \in \{0,1\}^n$ such that the

Algorithm **Test-Sparse-Poly**(f, s, ϵ)
Input: Black-box access to $f : \{0,1\}^n \to \{-1,1\}$; sparsity parameter $s \geq 1$; error parameter $\epsilon > 0$
Output: **"yes"** if f is an s-sparse $GF(2)$ polynomial, **"no"** if f is ϵ-far from every s-sparse $GF(2)$ polynomial

1. Let $\tau = \Theta(\epsilon), \Delta = \Theta(\mathrm{poly}(\tau, 1/s)), r = \Theta(s/\Delta), \delta = \Theta(\mathrm{poly}(\tau, 1/s))$.[a]
2. Set $\{I_j\}_{j=1}^r$ to be a random partition of $[n]$.
3. Choose α uniformly at random from the set $\mathcal{A}(\tau, \Delta) \stackrel{\text{def}}{=} \{\frac{\tau}{4s^2} + (8\ell - 4)\Delta : 1 \leq \ell \leq K\}$ where K is the largest integer such that $8K\Delta \leq \frac{\tau}{4s^2}$.
4. For each subset $I_1, \ldots, I_r$ run the independence test $M \stackrel{\text{def}}{=} \frac{2}{\Delta^2} \ln(200r)$ times and let $\widetilde{\mathrm{Vr}}_f(I_j)$ denote $2 \times$ (fraction of the M runs on I_j that the test rejects). If any subset I_j has $\widetilde{\mathrm{Vr}}_f(I_j) \in [\alpha - 2\Delta, \alpha + 3\Delta]$ then exit and return "**no**," otherwise continue.
5. Let $\widetilde{L}(\alpha) \subseteq [r]$ denote $\{j \in [r] : \widetilde{\mathrm{Vr}}_f(I_j) < \alpha - 2\Delta < \alpha\}$ and let $\widetilde{H}(\alpha)$ denote $[r] \setminus \widetilde{L}(\alpha)$. Let $\widetilde{f}' : \{0,1\}^n \to \{-1,1\}$ denote the function $f|_{0 \leftarrow \cup_{j \in \widetilde{L}(\alpha)} I_j}$.
6. Draw a sample of $m \stackrel{\text{def}}{=} \frac{2}{\epsilon} \ln 12$ uniform random examples from $\{0,1\}^n$ and evaluate both $\widetilde{f}'$ and f on each of these examples. If f and $\widetilde{f}'$ disagree on any of the m examples then exit and return "**no**." If they agree on all examples then continue.
7. Run the learning algorithm **LearnPoly**$'(s, |\widetilde{H}(\alpha)|, \epsilon/4, 1/100)$ from [SS96] using **SimMQ**$(f, \widetilde{H}(\alpha), \{I_j\}_{j \in \widetilde{H}(\alpha)}, \alpha, \Delta, z, \delta/Q(s, |\widetilde{H}(\alpha)|, \epsilon/4, 1/100))$ to simulate each membership query on a string $z \in \{0,1\}^{|\widetilde{H}(\alpha)|}$ that **LearnPoly**$'$ makes. If **LearnPoly**$'$ returns "not s-sparse" then exit and return "**no**." Otherwise the algorithm terminates successfully; in this case return "**yes**."

[a] More precisely, we set $\tau = \epsilon/600$, $\Delta = \min\{\Delta_0, (\tau/8s^2)(\delta/\ln(2/\delta))\}$, $r = 4Cs/\Delta$ (for a suitable constant C from Theorem 3), where $\Delta_0 \stackrel{\text{def}}{=} \tau/(1600s^3 \log(8s^3/\tau))$ and $\delta \stackrel{\text{def}}{=} 1/\Big(100s \log(8s^3/\tau) Q\big(s, s\log(8s^3/\tau), \epsilon/4, 1/100\big)\Big)$

Fig. 1. The algorithm **Test-Sparse-Poly**

high-variation variable in each I_j ($j \in H$) is set in the desired way in x, and it returns the value $f'(x)$.

4.1 Time and Query Complexity of Test-Sparse-Poly

As stated in Figure 1, the **Test-Sparse-Poly** algorithm runs **LearnPoly**$'(s, |\widetilde{H}(\alpha)|, \epsilon/4, 1/100)$ using **SimMQ**$(f, \widetilde{H}(\alpha), \{I_j\}_{j \in \widetilde{H}(\alpha)}, \alpha, \Delta, z, 1/(100Q(s, |\widetilde{H}(\alpha)|, z, 1/100)))$ to simulate each membership query on an input string $z \in \{0,1\}^{|\widetilde{H}(\alpha)|}$. Thus the algorithm is being run over a domain of $|\widetilde{H}(\alpha)|$ variables. Since we certainly have $|\widetilde{H}(\alpha)| \leq r \leq \mathrm{poly}(s, \frac{1}{\epsilon})$, Corollary 1 gives that **LearnPoly**$'$ makes at most $\mathrm{poly}(s, \frac{1}{\epsilon})$ many calls to **SimMQ**. From this point, by inspection of **SimMQ**, **SHIV** and **Test-Sparse-Poly**, it is straightforward to verify that **Test-Sparse-Poly** indeed makes $\mathrm{poly}(s, \frac{1}{\epsilon})$ many queries to f and runs in time

Algorithm **Set-High-Influence-Variable**$(f, I, \alpha, \Delta, b, \delta)$
Input: Black-box access to $f : \{0,1\}^n \to \{-1,1\}$; (α, Δ)-well-structured set $I \subseteq [n]$; bit $b \in \{0,1\}$; failure parameter δ.
Output: assignment $w \in \{0,1\}^I$ to the variables in I such that $w_i = b$ with probability $1 - \delta$

1. Draw x uniformly from $\{0,1\}^I$. Define $I^0 \stackrel{\text{def}}{=} \{j \in I : x_j = 0\}$ and $I^1 \stackrel{\text{def}}{=} \{j \in I : x_j = 1\}$.
2. Apply $c = \frac{2}{\alpha}\ln(\frac{2}{\delta})$ iterations of the *independence test* to (f, I^0). If any of the c iterations reject, mark I^0. Do the same for (f, I^1).
3. If both or neither of I^0 and I^1 are marked, stop and output "fail".
4. If I^b is marked then return the assignment $w = x$. Otherwise return the assignment $w = \overline{x}$ (the bitwise negation of x).

Fig. 2. The subroutine **Set-High-Influence-Variable**

Algorithm **SimMQ**$(f, H, \{I_j\}_{j\in H}, \alpha, \Delta, z, \delta)$
Input: Black-box access to $f : \{0,1\}^n \to \{-1,1\}$; subset $H \subseteq [r]$; disjoint subsets $\{I_j\}_{j\in H}$ of $[n]$; parameters $\alpha > \Delta$; string $z \in \{0,1\}^{|H|}$; failure probability δ
Output: bit b which, with probability $1 - \delta$ is the value of f' on a random assignment x in which each high-variation variable $i \in I_j$ ($j \in H$) is set according to z

1. For each $j \in H$, call **Set-High-Influence-Variable**$(f, I_j, \alpha, \Delta, z_j, \delta/|H|)$ and get back an assignment (call it w^j) to the variables in I_j.
2. Construct $x \in \{0,1\}^n$ as follows: for each $j \in H$, set the variables in I_j according to w^j. This defines x_i for all $i \in \cup_{j\in H} I_j$. Set $x_i = 0$ for all other $i \in [n]$.
3. Return $b = f(x)$.

Fig. 3. The subroutine **SimMQ**

$\text{poly}(s, \frac{1}{\epsilon})$ as claimed in Theorem 1. Thus, to prove Theorem 1 it remains only to establish correctness of the test.

4.2 Sketch of Completeness

The main tool behind our completeness argument is Theorem 3. Suppose f is indeed an s-sparse polynomial. Then Theorem 3 guarantees that a randomly chosen α will w.h.p. yield a "gap" such that subsets with a high-influence variable have variation above the gap, and subsets with no high-influence variable have variation below the gap. This means that the estimates of each subset's variation (obtained by the algorithm in step 4) are accurate enough to effectively separate the high-variation subsets from the low-variation ones in step 5. Thus, the function $\widetilde{f'}$ defined by the algorithm will w.h.p be equal to the function p' from Theorem 3.

Assuming that f is an s-sparse polynomial (and that $\widetilde{f'}$ is equal to p'), Theorem 3 additionally implies that the function $\widetilde{f'}$ will be close to the original function (so Step 6 will pass), that $\widetilde{f'}$ only depends on $\text{poly}(s, 1/\epsilon)$ many variables, and that all of the

subsets I_j that "survive" into $\widetilde{f}'$ are well-structured. As we show in Appendix B, this condition is sufficient to ensure that **SimMQ** can successfully simulate membership queries to $\widetilde{f}''$. Thus, for f an s-sparse polynomial, the **LearnPoly**$'$ algorithm can run successfully, and the test will accept.

4.3 Sketch of Soundness

Here, we briefly argue that if **Test-Sparse-Poly** accepts f with high probability, then f must be close to some s-sparse polynomial (we give the full proof in Appendix C). Note that if f passes Step 4, then **Test-Sparse-Poly** must have obtained a partition of variables into "high-variation" subsets and "low-variation" subsets. If f passes Step 6, then it must moreover be the case that f is close to the function $\widetilde{f}'$ obtained by zeroing out the low-variation subsets.

In the last step, **Test-Sparse-Poly** attempts to run the **LearnPoly**$'$ algorithm using $\widetilde{f}'$ and the high-variation subsets; in the course of doing this, it makes calls to **SimMQ**. Since f could be an arbitrary function, we do not know whether each high-variation subset has at most one variable relevant to $\widetilde{f}'$ (as would be the case, by Theorem 3, if f were an s-sparse polynomial). However, we are able to show (Lemma 11) that, if with high probability all calls to the **SimMQ** routine are answered without its ever returning "fail," then $\widetilde{f}'$ must be close to a junta g whose relevant variables are the individual "highest-influence" variables in each of the high-variation subsets. Now, given that **LearnPoly**$'$ halts successfully, it must be the case that it constructs a final hypothesis h that is itself an s-sparse polynomial and that agrees with many calls to **SimMQ** on random examples. Lemma 12 states that, in this event, h must be close to g, hence close to $\widetilde{f}'$, and hence close to f.

5 Conclusion and Future Directions

An obvious question raised by our work is whether similar methods can be used to efficiently test s-sparse polynomials over a general finite field $\mathbb{F}$, with query and time complexity polynomial in s, $1/\epsilon$, and $|\mathbb{F}|$. The basic algorithm of [DLM+07] uses $\tilde{O}((s|\mathbb{F}|)^4/\epsilon^2)$ queries to test s-sparse polynomials over $\mathbb{F}$, but has running time $2^{\omega(s|\mathbb{F}|)} \cdot (1/\epsilon)^{\log\log(1/\epsilon)}$ (arising, as discussed in Section 1, from brute-force search for a consistent hypothesis). One might hope to improve that algorithm by using techniques from the current paper. However, doing so requires an algorithm for properly learning s-sparse polynomials over general finite fields. To the best of our knowledge, the most efficient algorithm for doing this (given only black-box access to $f : \mathbb{F}^n \to \mathbb{F}$) is the algorithm of Bshouty [Bsh97b] which requires $m = s^{O(|\mathbb{F}| \log |\mathbb{F}|)} \log n$ queries and runs in $\mathrm{poly}(m, n)$ time. (Other learning algorithms are known which do not have this exponential dependence on $|\mathbb{F}|$, but they either require evaluating the polynomial at complex roots of unity [Man95] or on inputs belonging to an extension field of $\mathbb{F}$ [GKS90, Kar89].) It would be interesting to know whether there is a testing algorithm that simultaneously achieves a polynomial runtime (and hence query complexity) dependence on both the size parameter s and the cardinality of the field $|\mathbb{F}|$.

Another goal for future work is to apply our methods to other classes beyond just polynomials. Is it possible to combine the "testing by implicit learning" approach of [DLM+07] with other membership-query-based learning algorithms, to achieve time and query efficient testers for other natural classes?

References

[AKK+03] Alon, N., Kaufman, T., Krivelevich, M., Litsyn, S., Ron, D.: Testing low-degree polynomials over GF(2). In: Proc. RANDOM, pp. 188–199 (2003)

[Ang88] Angluin, D.: Queries and concept learning. Machine Learning 2, 319–342 (1988)

[BLR93] Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. J. Comp. Sys. Sci 47, 549–595 (1993); Earlier version in STOC 1990

[BM02] Bshouty, N., Mansour, Y.: Simple Learning Algorithms for Decision Trees and Multivariate Polynomials. SIAM J. Comput. 31(6), 1909–1925 (2002)

[BS90] Blum, A., Singh, M.: Learning functions of k terms. In: Proceedings of the 3rd Annual Workshop on Computational Learning Theory (COLT), pp. 144–153 (1990)

[Bsh97a] Bshouty, N.: On learning multivariate polynomials under the uniform distribution. Information Processing Letters 61(3), 303–309 (1997)

[Bsh97b] Bshouty, N.: Simple learning algorithms using divide and conquer. Computational Complexity 6, 174–194 (1997)

[DLM+07] Diakonikolas, I., Lee, H., Matulef, K., Onak, K., Rubinfeld, R., Servedio, R., Wan, A.: Testing for concise representations. In: Proc. 48th Ann. Symposium on Computer Science (FOCS), pp. 549–558 (2007)

[EK89] Ehrenfeucht, A., Karpinski, M.: The computational complexity of (xor,and)-counting problems. Technical report (preprint 1989)

[FKR+04] Fischer, E., Kindler, G., Ron, D., Safra, S., Samorodnitsky, A.: Testing juntas. Journal of Computer & System Sciences 68, 753–787 (2004)

[FS92] Fischer, P., Simon, H.U.: On learning ring-sum expansions. SIAM Journal on Computing 21(1), 181–192 (1992)

[GGR98] Goldreich, O., Goldwaser, S., Ron, D.: Property testing and its connection to learning and approximation. Journal of the ACM 45, 653–750 (1998)

[GKS90] Grigoriev, D., Karpinski, M., Singer, M.: Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. SIAM Journal on Computing 19(6), 1059–1063 (1990)

[Kar89] Karpinski, M.: Boolean circuit complexity of algebraic interpolation problems (TR-89-027) (1989)

[KKL88] Kahn, J., Kalai, G., Linial, N.: The influence of variables on boolean functions. In: Proc. 29th FOCS, pp. 68–80 (1988)

[KL93] Karpinski, M., Luby, M.: Approximating the Number of Zeros of a $GF[2]$ Polynomial. Journal of Algorithms 14, 280–287 (1993)

[LVW93] Luby, M., Velickovic, B., Wigderson, A.: Deterministic approximate counting of depth-2 circuits. In: Proceedings of the 2nd ISTCS, pp. 18–24 (1993)

[Man95] Mansour, Y.: Randomized interpolation and approximation of sparse polynomials. SIAM Journal on Computing 24(2), 357–368 (1995)

[MORS07] Matulef, K., O'Donnell, R., Rubinfeld, R., Servedio, R.: Testing Halfspaces. Technical Report 128, Electronic Colloquium in Computational Complexity (2007)

[PRS02] Parnas, M., Ron, D., Samorodnitsky, A.: Testing basic boolean formulae. SIAM J. Disc. Math. 16, 20–46 (2002)

[RB91] Roth, R., Benedek, G.: Interpolation and approximation of sparse multivariate polynomials over $GF(2)$. SIAM J. Comput. 20(2), 291–314 (1991)

[Ron07] Ron, D.: Property testing: A learning theory perspective. In: Bshouty, N.H., Gentile, C. (eds.) COLT. LNCS (LNAI), vol. 4539. Springer, Heidelberg (2007), http://www.eng.tau.ac.il/~danar/Public-ppt/colt07.ppt

[SS96] Schapire, R., Sellie, L.: Learning sparse multivariate polynomials over a field with queries and counterexamples. J. Comput. & Syst. Sci. 52(2), 201–213 (1996)

Testing Properties of Sets of Points in Metric Spaces

Krzysztof Onak*

Massachusetts Institute of Technology, Cambridge MA 02139, USA

Abstract. Given query access to a set of points in a metric space, we wish to quickly check if it has a specific property. More precisely, we wish to distinguish sets of points that have the property from those that need to have at least an ε fraction of points modified to achieve it.

We show one-sided error testers that immediately follow from known characterizations of metric spaces. Among other things, we give testers for tree metrics and ultrametrics which are optimal among one-sided error testers. Our tester for embeddability into the line is optimal even among two-sided error testers, and runs in sublinear time. We complement our algorithms with several lower bounds. For instance, we present lower bounds for testing dimensionality reduction in the ℓ_1 and ℓ_∞ metrics, which improve upon lower bounds given by Krauthgamer and Sasson (SODA 2003). All our lower bounds are constructed by using a generic approach.

We also look at the problem from a streaming perspective, and give a method for converting each of our property testers into a streaming tester.

1 Introduction

Many real-world data sets are sets of points in a metric space. If the metric space is complicated or, like high-dimensional spaces in many applications, expensive to deal with, then a natural question is that of finding a simplified representation of the input set of points. In many cases, we are not as much interested in the actual points as in the distances between them. We may then consider mapping the data set to a simpler space so that the distances between the points are either exactly or approximately preserved.

The best example of a tool that allows for such transformation is the Johnson-Lindenstrauss lemma (see [1] and [2]). It states that for any $\varepsilon > 0$, there exists a mapping of an n-point set of points in ℓ_2 into $\ell_2^{O(\log(n)/\varepsilon^2)}$ with multiplicative distortion $1 + \varepsilon$. For instance, if small distortion is acceptable, and we have an algorithm that runs in time exponential in the dimension, then by using the Johnson-Lindenstrauss lemma, we may get a polynomial-time approximation algorithm.

Another possible approach is to take advantage of profound properties of our data sets in constructing an embedding into a simpler space. But how can one

* Supported by an Akamai Presidential Fellowship and NSF grant 0514771.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 515–526, 2008.

efficiently discover such properties? This problem was addressed in a variety of settings by Parnas and Ron [3] and Krauthgamer and Sasson [4], who focused on constructing testers for multiple metric properties. By using those testers, one may check if a data set or a metric is close to a specific property, and if that turns out to be the case, try to use the property to construct a nice embedding into a simpler space. We continue this line of research. In particular, we follow and generalize the model of Krauthgamer and Sasson. We describe existing models and previous results on them in more detail later in this section.

1.1 Property Testing

In *property testing* (see [5,6]), one is interested in checking if the input (for instance, a set of points) has a specific property. We, however, do not attempt to answer this question exactly. Instead, we try to quickly distinguish, by reading a small part of the input, between sets of points which have the property, and sets of points which are significantly different from any set that has the property. We assume some notion of the distance from the property. Usually, the distance is defined as the minimum fraction of the input that needs to be modified to achieve the property. If the distance of an input x from the property is at least ε, then we say that x is *ε-far* from the property. We now define what a tester is.

Definition 1. *A* (two-sided error) tester *for a property $\mathcal{P}$ is an algorithm that accepts an input that has property $\mathcal{P}$ with probability at least $2/3$, and rejects with probability at least $2/3$ every input which is ε-far from $\mathcal{P}$. Moreover, if the tester never rejects any input that has property $\mathcal{P}$, we say that such a tester has* one-sided error*.*

Note that one-sided error testers only reject an input if they find evidence that it does not have a given property. Traditionally, the main quantities minimized in property testing are the query complexity and the running time of a tester.

1.2 Considered Models and Previous Results

The Model of Parnas and Ron. Parnas and Ron [3] assumed that the input metric on n points was given as an $n \times n$ matrix of distances between each pair of points. The distance to a property was in their setting defined as the minimum number of matrix entries that must be modified to achieve the property. They showed one-sided error testers for verifying if the input metric embeds into ℓ_2^d, if it is a tree metric, an ultrametric or an approximate ultrametric. Their testers choose a random subset of points of size independent of the size of the metric, and check if the metric restricted to them has the property. We consider almost the same set of properties in a different setting. Unfortunately, in our setting the numbers of queries must depend on the size of the metric.

It is also worth mentioning that Abraham et al. [7] considered a related notion of embeddings that preserve all but a small fraction of distances.

The Model of Krauthgamer and Sasson. The problem of testing a dimension of a set of points was stated by Krauthgamer and Sasson [4]. They assumed that their input is a set of points in a given metric space. What differentiates their model from the model of Parnas and Ron is that the distance from a property equals the minimum fraction of points, rather than the minimum fraction of distances, that must be modified. Krauthgamer and Sasson asked if given a set of points in ℓ_p^k, we can efficiently determine if it isometrically embeds into ℓ_p^d, for some fixed d. They showed that for $p = 2$, it suffices to read a random subset of points of size $O(d/\varepsilon)$ to find with constant probability a certificate that the entire set does not embed into ℓ_p^d, provided it is ε-far from embeddability into ℓ_p^d. Furthermore, Krauthgamer and Sasson showed that for $p = 1$, any tester for embeddability of a set of points in the ℓ_1 metric into ℓ_1^d must query $\Omega(\sqrt[4]{n})$ points. They also gave a lower bound of $\Omega(\sqrt{n/\Delta})$ for testing embeddability of a set of points in ℓ_2^m with distortion Δ into ℓ_2^d, and a lower bound of $\Omega(\min\{\sqrt{n}, \sqrt{m/\log m}\})$ for testing if a set of points in ℓ_2^m can be perturbed by $\delta > 0$ so that it isometrically embeds into ℓ_2^d.

Our Generalized Model. We also assume that the input is a set of points. We, however, take a more general look at the model proposed by Krauthgamer and Sasson. As opposed to them, we do not assume anything about the metric space the input set of points lies in. Instead, our testers have query access to a distance oracle for the underlying metric space. The distance to a property is defined as follows throughout the whole paper.

Definition 2. *We say that a set S of points is* ε-far *from a property if any subset of S of more than $(1-\varepsilon)|S|$ points does not have the property.*

1.3 Considered Properties

For completeness, we first recall the notion of a metric space.

Definition 3. *Let $\mathcal{M}$ be a pair $\langle S, \delta\rangle$, where S is a set of points and δ is a function from a pair of points in S to $\mathbb{R}_{\geq 0}$, the set of non-negative reals. We say that $\mathcal{M}$ is a* metric space *if for any $x, y, z \in S$, $\delta(x, y) = \delta(y, x)$, $\delta(x, y) = 0$ iff $x = y$, and $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$.*

Properties that we test are easy to express via embeddings.

Definition 4. *A metric space $\mathcal{M}_1 = \langle S_1, \delta_1\rangle$ is* embeddable *(or* embeds*) into a metric space $\mathcal{M}_2 = \langle S_2, \delta_2\rangle$ if there exists a mapping f from S_1 to S_2 that preserves distances, i.e. for any pair x and y of points in S, it holds that*

$$\delta_1(x, y) = \delta_2(f(x), f(y)).$$

We now define what a tree metric and an ultrametric are. Note that every ultrametric is a tree metric.

Definition 5. *A metric $\mathcal{M}$ is a* tree metric *if it can be embedded into the shortest-path metric of some weighted tree.*

Alternatively, $\mathcal{M} = \langle S, \delta \rangle$ is a tree metric if it meets the following 4-point condition:

$$\forall x, y, z, w \in S, \quad \delta(x, y) + \delta(z, w) \leq \max\{\delta(x, z) + \delta(y, w), \delta(x, w) + \delta(z, y)\}.$$

Definition 6. *A metric $\mathcal{M}$ is an* ultrametric *if there exists a weighted rooted tree T of all leaves at the same distance from the root, and M embeds into the shortest-path metric of T with all points in $\mathcal{M}$ mapped to points corresponding to the leaves of T.*

Alternatively, $\mathcal{M} = \langle S, \delta \rangle$ is an ultrametric if it meets the following 3-point condition:

$$\forall x, y, z \in S, \quad \delta(x, y) \leq \max\{\delta(x, z), \delta(y, z)\}.$$

1.4 Our Results

Embeddability into the Line: We show an optimal $O(\sqrt{n/\varepsilon})$-query one-sided tester. We prove that any tester, including two-sided testers, must query $\Omega(\sqrt{n/\varepsilon})$ points even if the set of points is a subset of ℓ_1^2. This improves upon the $\Omega(\sqrt[4]{n})$ lower bound of Krauthgamer and Sasson [4].

Tree Metrics and Ultrametrics: We exhibit one-sided error testers of query complexity $O(n^{2/3}\varepsilon^{-1/3})$. The testers are optimal among one-sided tester.

Embeddability into ℓ_1^2 and ℓ_2^d: We exhibit one-sided testers of query complexity $O(n^{5/6}\varepsilon^{-1/6})$ and $O(n^{(d+2)/(d+3)}\varepsilon^{-1/(d+3)})$, respectively. Note that the spaces ℓ_1^2 and ℓ_∞^2 are isometric.

Dimension Reduction Testing in ℓ_1 and ℓ_∞: We strengthen the results of Krauthgamer and Sasson [4] on the dimension reduction. We show that any one-sided tester for testing if a set of points in ℓ_1^{d+1} embeds into ℓ_1^d must query $\Omega(n^{d/(d+1)}\varepsilon^{-1/(d+1)})$ points for sufficiently small ε. For the analogous setting in ℓ_∞, we prove a lower bound of $\Omega(n^{2^{d-1}/(2^{d-1}+1)}\varepsilon^{-1/(2^{d-1}+1)})$.

Embeddability into the Line with Distortion: In any metric ℓ_p, for every $\delta > 0$, there exists a dimension d, and a constant $C > 1$ such that a one-sided error tester for embeddability of points in ℓ_p^d into the line with distortion at most C must query $\Omega(n^{1-\delta})$ points.

Most of our lower bounds only apply to one-sided testers, but all known testers for these and related problems have one-sided error. Due to the space limitation, many of our results are not included in this version of the paper.

1.5 A Streaming Perspective

The Model. We also take a look at the testing problem from a streaming perspective (see [8] for a survey on streaming algorithms). For our purposes, a *streaming algorithm* is an algorithm that takes an input stream, and computes a result in one pass over the input. A streaming algorithm can read the *entire*

input, but only once. The main quantity that is minimized in streaming is the space complexity.

Feigenbaum et al. [9] considered a model that combines streaming and property testing. A *streaming tester* takes an input stream, and accepts with probability at least $2/3$, if the input has a given property, and rejects with probability $2/3$, if the input does not have the property.

Our Results. We first show that the exact verification of properties considered by us requires at least $\Omega(n)$ bits of space. This lower bound can easily be overcome for most of properties that we consider by using stream testers. In particular, we show that for each property that we had an algorithm that used $O(n^{(d-1)/d}\varepsilon^{-1/d})$ samples in the property testing approach, there is a streaming tester that needs space to keeps only $O(n^{(d-2)/(d-1)}\varepsilon^{-1/(d-1)})$ points. For instance, for embeddability into the line, this gives a streaming tester that keeps only $O(1/\varepsilon)$ points.

1.6 Our Techniques

Testers via Small Subspace Characterizations. There are several properties (see for instance [10,11]) that can be characterized by a property that holds for any subset of points of size of at most c, for some constant c. In this case, we can create a one-sided tester that looks for a small subset of at most c points that do not have the property. Using this approach we get a tester for ultrametrics which is optimal among one-sided testers.

Moreover, to build an efficient tester for embeddability into ℓ_1^2, we use the algorithm of Edmonds [12] to check if a collected sample embeds into ℓ_1^2. In the case of testing for embeddability into the line and for being a tree metric, this general approach does not yield an optimal tester, but we prove that with respect to some fixed number of points, it suffices to find a smaller group of points, and therefore, we can improve the query complexity of the testers. For instance, in testing for tree metrics, it essentially suffices to find a triple, not a quadruple of points of a specific property, and therefore, the query complexity improves.

Lower Bounds for Property Testing. All our lower bounds follow from the same approach. We construct a gadget, and make several copies of it. Any subset of points that does not contain an entire copy of the gadget has the property, but the whole set of points is far from the property. This implies that a one-sided tester must read an entire copy of the gadget to reject the input. To construct such gadgets in ℓ_1, we use a theorem of Hadlock and Hoffman [13]. We show and use an analogue of this result in ℓ_∞.

Streaming Testers. All our lower bounds follow from a simple application of the set disjointness lower bound [14,15]. As for algorithms, we notice that whenever a standard property tester looks for a k-tuple of points to find evidence that the input does not have a property, a streaming tester may draw only the first $k-1$ points of the tuple in the stream, and it will notice the k-th complementing point, when it reads it. An improvement follows from the fact that finding $(k-1)$-tuples is easier than finding k-tuples.

2 Two Simple Probability Facts

We use two probability facts throughout the paper. Suppose that a set contains many disjoint groups of elements, and by selecting elements of the set at random, we wish to draw at least one of the groups entirely. The facts below specify what number of samples is sufficient and what number of samples is necessary. We omit their proof in this version of the paper.

Fact 7 (Upper bound). *Let S be a set of n items where some of them constitute g disjoint groups of size k each. It suffices to select $\min\left\{\frac{2n}{g^{1/k}}, n\right\} = O\left(\frac{n}{g^{1/k}}\right)$ items at random to draw at least one of the groups entirely with constant probability.*

Fact 8 (Lower bound). *Let S be a set of n items where some of them constitute g disjoint groups of size k each. The probability that by we draw at least one group entirely by choosing at random q items from S is not greater than $g \cdot (q/n)^k$.*

3 Testing Via a Small Subset Characterization

Some properties P can be expressed as a condition which says that there exists a constant c such that a metric space $\mathcal{M}$ has property P if and only if every subspace of $\mathcal{M}$ of at most c points has a computable property P'. Apart from the alternative definitions of a tree metric and an ultrametric, we list here the following two examples:

- A metric spaces M embeds into ℓ_1^2 (or equivalently into ℓ_∞^2) if and only if each subset of M of at most 6 points embeds into ℓ_1^2 (Bandelt and Chepoi [10]).
- A metric spaces M embeds into ℓ_2^d if and only if each subset of M of at most $d+3$ points embeds into ℓ_2^d (Menger [11]).

All properties of this form yield testers of sublinear query complexity.

Theorem 9. *Let c be a constant such that a set S of points has a property P if and only if every subset of S at most c points has a computable property P'. There exists a one-sided error tester for P that queries $O\left(n^{1-1/c}\varepsilon^{-1/c}\right)$ points. The tester finds with constant probability evidence that S does not have P, provided S is ε-far from having it.*

Proof. Let S be ε-far from P. This implies that any subset of S of at least $(1-\varepsilon)n$ does not have the property P.

Let S_0 be equal to S. As long as $|S_i| > (1-\varepsilon)n$, we inductively define S_{i+1} and T_{i+1} as follows. Since S_i does not have the property P, there exists a subset of S_i of at most c points that does not have the property P'. Let T_{i+1} be any such subset, and let $S_{i+1} = S_i \backslash T_{i+1}$. Eventually, we have at least $\varepsilon n/c$ disjoint

groups, each of size at most c such that any of them proves that the set does not have P.

By Fact 7 it suffices to draw $O\left(n^{1-1/c}\left(\frac{c}{\varepsilon}\right)^{1/c}\right) = O\left(n^{1-1/c}\varepsilon^{-1/c}\right)$ random elements to entirely draw with constant probability at least one of these groups, and hence to discover that S does not have P. Then, because P' is computable, it suffices to verify that P' holds for every subset of at most c points. □

Theorem 9 and the aforementioned characterizations yield sublinear-query testers. Their running time can be improved, by checking if the whole sample subset has the given property. One can check if a metric on n points is a tree metric or an ultrametric in $O(n^2)$ time [16], and check if it embeds into ℓ_1^2 in $O(n^2 \log^3 n)$ time [12]. We summarize all the results in the corollary below.

Corollary 10. *There are sublinear-query one-sided error testers if the input set of points*

- *spans a tree metric (query complexity: $O(n^{3/4}\varepsilon^{-1/4})$, time $O(n^{3/2}\varepsilon^{-1/2})$),*
- *spans an ultrametric (query complexity: $O(n^{2/3}\varepsilon^{-1/3})$, time $O(n^{4/3}\varepsilon^{-2/3})$),*
- *embeds into ℓ_1^2 (query complexity: $O(n^{5/6}\varepsilon^{-1/6})$, time $O(n^{5/3}\varepsilon^{-1/3}\log^3 n)$),*
- *embeds into ℓ_2^d (query complexity: $O(n^{(d+2)/(d+3)}\varepsilon^{-1/(d+3)})$).*

4 Improved Testers

4.1 Testing Tree Metrics

We now show a slightly more efficient algorithm for testing if a metric spanned by a set of points is a tree metric. Recall that Corollary 10 gave us a tester of query complexity $O(n^{3/4}\varepsilon^{-1/4})$. The reason behind the complexity is that the tester looks for quadruples of points. The lemma below implies that it really suffices to look for triples of points, and we can therefore improve the query complexity to $O(n^{2/3}\varepsilon^{-1/3})$. We omit the proof in this version of the paper.

Lemma 11. *Let $S = \{x, y, s, t\}$ be a subset of four points in a metric space that spans a non-tree submetric. Let p be an arbitrary point in the same metric space. There exists a subset S' of S of size 3 such that $S' \cup \{p\}$ spans a non-tree submetric as well.*

Corollary 12. *There is a one-sided error tester for being a tree metric that queries only $O(n^{2/3}\varepsilon^{-1/3})$ points and runs in $O(n^{4/3}\varepsilon^{-2/3})$ time.*

4.2 Testing Embeddability into the Line

We now show an optimal tester for embeddability into the line. The query complexity of the tester is $O(\sqrt{n/\varepsilon})$. Note that this significantly improves on $O(n^{3/4}\varepsilon^{-1/4})$, the query complexity given by Corollary 10.

Theorem 13. *There is a one-sided error tester for isometric embeddability into the line that queries $O(\sqrt{n/\varepsilon})$ points.*

Proof. Consider first the following algorithm. Query $O(1/\varepsilon)$ random points. If all points in the sample are identical, accept the input. Otherwise, let p and q be the first two different drawn points in the sample. Place p and q on the line at distance $\delta(p,q)$. Now for any other point r in the set, the placement of p and q uniquely determines the position of r on the line, provided the subspace $\{p,q,r\}$ embeds into the line. Draw $O(\sqrt{n/\varepsilon})$ new points, and if for any point r in the new sample, the subspace $\{p,q,r\}$ does not embed into the line, reject the input. Otherwise, place all the points from the sample on the line with respect to p and q, and verify if all the pairwise distances on the line equal the distances in the original metric. If at least one of them is different, reject. Otherwise, accept.

We assume that $\varepsilon \geq 1/n$, since every set of points is either embeddable into the line, or is $1/n$-far from this property. This implies in particular that $1/\varepsilon = O(\sqrt{n/\varepsilon})$, and thus, the query complexity of the algorithm is $O(\sqrt{n/\varepsilon})$.

Let us prove that the above algorithm works. Clearly, it can only reject inputs that are not embeddable into the line. Suppose that an input is accepted by the above algorithm with probability at least 2/3. We show that the input is $\varepsilon/2$-close to a set embeddable into the line. The input can be accepted in two different steps of the algorithm, and it must be accepted with probability at least 1/6 in one of them. If it is accepted with probability at least 1/6 because all points in the first sample are identical, the set must be $\varepsilon/2$-close to an input that consists of n copies of a single point. Suppose now that it passes the other two tests with probability at least 1/6 for arbitrary p and q fixed in the first phase of the algorithm. Let S' be the maximum size subset of the input set such that each point r in S' embeds into the line with respect to p and q, and all pairwise distances for the points in S' are preserved in this embedding. We claim that $|S'| \geq (1-\varepsilon/2)|S|$, i.e., there is a subset of the input of size $(1-\varepsilon/2)n$ that isometrically embeds into the line. Firstly, the fraction of points in S that do not embed with respect to p and q must be smaller than $\varepsilon/4$, since the constant hidden in the big-Oh notation is sufficiently large to detect every fraction greater than $\varepsilon/4$ of these points with probability greater than 5/6. Secondly, the fraction of points of S in $S\backslash S'$ that embed with respect to p and q also cannot be to large. Denote the set of those points by U. Suppose that $|U| \geq \varepsilon n/4$. Let $X_i = S' \cup U$, and iteratively create X_i as follows. As long as $|X_i| > |S'|$, there is a pair of points (a_i, b_i) in X_i such that the distance between a_i and b_i changes after embedding into the line with respect to p and q. We create X_{i+1} by removing these two points from X_i. If $|U| \geq \varepsilon n/4$, there are at least $\varepsilon n/8$ such disjoint pairs of points, and by Fact 7, we find such a pair with probability greater than 5/6. Hence the size of T must be less than $\varepsilon n/4$, and the size of S' is at least $(1-\varepsilon/2)\cdot n$. Therefore, the input is $\varepsilon/2$-close to an input embeddable into the line, which finishes the proof of the correctness of the algorithm. □

One can show that there is an algorithm that for a set of s points, checks in time $O(s(T+\log s))$ if it exactly embeds into the line or not, where T is the time complexity of computing the distance between two points.

Corollary 14. *There is a one-sided error tester for isometric embeddability into the line that queries $O(\sqrt{n/\varepsilon})$ points and runs in $O(\sqrt{n/\varepsilon}(T+\log n))$ time, where T is the time necessary to compute the distance between two points.*

5 Lower Bounds

We give a number of lower bounds for testing. All of them follow from the same approach. We create a constant size gadget that is repeated several times. Until we read entirely at least one of the copies of the gadget, the subset of points has a considered property. At the same time the whole input is far from the property. A one-sided tester must therefore read an entire copy of the gadget, which requires many queries. One can also show that each of our lower bounds can be transformed into an $\Omega(\sqrt{n/\varepsilon})$ lower bound for two-sided testers.

5.1 A Lower Bound for Testing Dimension Reduction in ℓ_1

A say that a set of points in ℓ_p^m is *d-dimensional* if it isometrically embeds into ℓ_p^d. We now present a general lower bound for one-sided error testers, which shows that a d-dimensionality tester with one-side error must query many points for small ε and large d. To prove the lower bound, we make use of a nonembeddability lemma by Hadlock and Hoffman [13]. They showed that to embed a tree metric into ℓ_1 one needs exactly $\lceil k/2 \rceil$ dimensions, where k is the number of leaves in the underlying tree. Here we only make use of the nonembeddability part of their result.

Lemma 15 (Hadlock and Hoffman [13]). *Let $\mathcal{M} = (S, \delta)$ be a tree metric of $k \geq 3$ leaves. $\mathcal{M}$ does not embed into ℓ_1^m for any $m < k/2$.*

Theorem 16. *Any one-sided tester for d-dimensionality must query $\Omega(n^{d/(d+1)} \cdot \varepsilon^{-1/(d+1)})$ points for $\varepsilon < 1/(2d+2)$, even if the host space is ℓ_1^{d+1}.*

Proof. A one-sided tester for inputs that are ε-far from d-dimensionality needs to detect with constant probability evidence of non-d-dimensionality. In our case, it must read with constant probability a subset of points which is not d-dimensional.

We will exhibit a $d+1$-dimensional set which is hard for one-sided testers. Before we pass this set to the tester, we randomly shuffle the list of the points. Thus we can assume that the tester reads random points from the set. (Bar-Yossef et al. [17] conduct an interesting analysis of testers for the properties that do not depend on the order of the elements in the input).

We will define a set of points in ℓ_1^{d+1}, which will not be d-dimensional. Let $\mathbf{e}_i$, $1 \leq i \leq d+1$, be the unit vector in $\mathbb{R}^{d+1}$ of the i-th coordinate equal to 1 and all the others equal to 0. Also define $\mathbf{1}$ and $\mathbf{0}$ to be the vectors of ones and zeros in all coordinates, respectively.

We construct an input set S as follows. Let $p = \varepsilon n$. First, we add $n - p(2d+2)$ copies of $\mathbf{0}$. Then, for each $1 \leq i \leq p$, we add the following group G_i of $2d+2$ points:

- $u_i = 3i \cdot \mathbf{1}$,
- $v_{ij} = 3i \cdot \mathbf{1} - \mathbf{e}_j$, for each $1 \le j \le d+1$,
- $w_{ij} = 3i \cdot \mathbf{1} + \mathbf{e}_j$, for each $1 \le j \le d$.

Note that each G_i is the shortest-path metric of the unweighted star of $2d+1$ leaves. Thus, by Lemma 15, G_i is not d-dimensional. To turn S into a d-dimensional set, we need to remove at least one point from each G_i, therefore S is ε-far from d-dimensionality. On the other hand, if we remove at least one point v_{ij} for each $1 \le i \le p$, we get a d-dimensional set. Since all the points v_{ij}, for fixed i, are symmetric in terms of the distance to the other points, we can assume without loss of generality that we remove $v_{i,d+1}$ for each i. We can define a distance-preserving embedding f of the remaining points into ℓ_1^d:

$$f(x) = \begin{cases} \mathbf{0}, & \text{if } x = \mathbf{0}; \\ 3\frac{d+1}{d} i \cdot \mathbf{1}, & \text{if } x = u_i; \\ 3\frac{d+1}{d} i \cdot \mathbf{1} - \mathbf{e}_j, & \text{if } x = v_{ij}; \\ 3\frac{d+1}{d} i \cdot \mathbf{1} + \mathbf{e}_j, & \text{if } x = w_{ij}. \end{cases}$$

One can easily check that this embedding does preserve all the distances.

Moreover, this implies that to find evidence that S is not d-dimensional, the tester needs to read all the v_{ij} for some i. If the tester queries q points and finds evidence with constant probability, it follows from Fact 8 that $p\left(\frac{q}{n}\right)^{d+1} = \Omega(1)$, which implies that $q = \Omega\left(\frac{n^{d/(d+1)}}{\varepsilon^{1/(d+1)}}\right)$. □

6 Streaming Testers

6.1 A Linear Lower Bound for the Exact Property Verification

We now give a sketch of how to prove a lower bound for the exact verification of properties in the streaming model. We omit many technical details. Each of our lower bounds for property testing can easily be turned into a lower bound for exactly checking a property in streaming. For each of those lower bounds, we design a size-k gadget for some constant k. Whenever an entire copy of the gadget is present in the input, the input does not have the property. We can also break each of the gadgets into two parts of the same, or almost the same size such that when only one of the halves is present, it does not contradict the property. We start from an input that has n/k copies of the gadget. One of the halves of each gadget is assigned to Alice, and the other one to Bob. Alice picks her set of points by selecting an arbitrary subset of her halves of gadgets. So does Bob. If Alice and Bob picked halves that compose to an entire copy of the gadget, the union of their sets of points does not have the property. Otherwise, it does. Clearly, we can now use any streaming algorithm for the exact property verification to give a protocol for set disjointness on the set $\{1, \ldots, n/k\}$. Alice first simulates the algorithm on her set of points, passes the intermediate state to Bob, and Bob continues the simulation on his set of points. In the worst case,

Alice must pass at least $\Omega(n/k)$ bits to Bob, so the amount of space used by the streaming algorithm is at least $\Omega(n/k)$. We state a corollary for embeddability into the line.

Lemma 17. *The exact verification of embeddability into the line requires $\Omega(n)$ bits of space in the streaming model.*

6.2 A Lower Bound for Streaming Testers

The above approach can easily be modified to give a lower bound for streaming testers. Instead of n/k different copies of the gadget, we now only have $1/(\varepsilon k)$ different copies, but we always repeat each of them εn times. Because of this, whenever the subsets of $\{1, \ldots, 1/(\varepsilon k)\}$ chosen by Alice and Bob intersect, there are εn copies of the gadget, which makes the set of points ε-far from the property. By the same argument as before, we get a lower bound of $\Omega(1/(\varepsilon k))$ bits of space. In particular, the following lower bound holds for embeddability into the line.

Lemma 18. *A streaming tester for embeddability into the line must use $\Omega(1/\varepsilon)$ bits of space.*

6.3 Algorithms

Note that if there is a property tester of query complexity T, then there is a streaming tester that keeps only T points. It collects T random points when it goes over the stream, and at the end simulates the property tester on the sample. Here, we show that the number of points kept can be decreased.

All our property testing algorithms look for a k-tuple of points that is used as (a part of) a certificate that the input does not have a property. There are always at least $\Omega(\varepsilon n/k)$ such k-tuples, if the input is ε-far from a property. The improvement comes from the fact that it suffices to draw the first $k-1$ points of one of the k-tuples, and then, going over the stream, check for each point if it complements a k-tuple. By Fact 7, we only need to collect $O(n^{(k-2)/(k-1)}\varepsilon^{-1/(k-1)})$ sample points from the stream as opposed to $O(n^{(k-1)/k}\varepsilon^{-1/k})$ samples in the property testing model.

Moreover, for testing embeddability into the line (testing tree metrics), we need two different fixed points (one fixed point). We can use for that the first two different points (the first point) of the stream. For embeddability into the line, we get the following lemma.

Lemma 19. *There is a one-sided error streaming tester for embeddability into the line that stores $\Omega(1/\varepsilon)$ points.*

Acknowledgments. The author would like to thank Alexandru Andoni and Ronitt Rubinfeld for useful comments on an early version of the paper.

References

1. Johnson, W.B., Lindenstrauss, J.: Extensions of Lipschitz mappings into a Hilbert space. In: Conference in Modern Analysis and Probability, New Haven, 1982. Comtemporary Mathematics, vol. 26, pp. 189–206. American Mathematical Society, Providence (1984)
2. Indyk, P.: Algorithmic applications of low-distortion geometric embeddings. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, pp. 10–33 (2001)
3. Parnas, M., Ron, D.: Testing metric properties. Information and Computation 187(2), 155–195 (2003)
4. Krauthgamer, R., Sasson, O.: Property testing of data dimensionality. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 18–27 (2003)
5. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM Journal on Computing 25(2), 252–271 (1996)
6. Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connection to learning and approximation. J. ACM 45(4), 653–750 (1998)
7. Abraham, I., Bartal, Y., Chan, H.T.H., Dhamdhere, K., Gupta, A., Kleinberg, J.M., Neiman, O., Slivkins, A.: Metric embeddings with relaxed guarantees. In: FOCS, pp. 83–100 (2005)
8. Muthukrishnan, S.: Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci. 1(2), 117–236 (2005)
9. Feigenbaum, J., Kannan, S., Strauss, M., Viswanathan, M.: Testing and spot-checking of data streams. Algorithmica 34(1), 67–80 (2002)
10. Bandelt, H.J., Chepoi, V.: Embedding metric spaces in the rectilinear plane: a six-point criterion. Discrete & Computational Geometry 15(1), 107–117 (1996)
11. Menger, K.: Untersuchungen uber allgemeine Metrik. Mathematische Annalen 100, 75–163 (1928)
12. Edmonds, J.: Embedding into ℓ_∞^2 is easy, embedding into ℓ_∞^3 is NP-complete. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 522–531 (2007)
13. Hadlock, F., Hoffman, F.: Manhattan trees. Utilitas Mathematica 13, 55–67 (1978)
14. Kalyanasundaram, B., Schnitger, G.: The probabilistic communication complexity of set intersection. SIAM J. Discrete Math. 5(4), 545–557 (1992)
15. Razborov, A.A.: On the distributional complexity of disjointness. In: ICALP, pp. 249–253 (1990)
16. Waterman, M.S., Smith, T.F., Singh, M., Beyer, W.A.: Additive evolutionary trees. Journal of Theoretical Biology 64, 199–213 (1977)
17. Bar-Yossef, Z., Kumar, R., Sivakumar, D.: Sampling algorithms: lower bounds and applications. In: Proceedings on 33rd Annual ACM Symposium on Theory of Computing, pp. 266–275 (2001)

An Expansion Tester for Bounded Degree Graphs

Satyen Kale[1,⋆] and C. Seshadhri[2]

[1] Microsoft Research
One Microsoft Way
Redmond, WA 98052
satyen.kale@microsoft.com
[2] Dept. of Computer Science,
Princeton University
35 Olden St, Princeton, NJ 08540
csesha@cs.princeton.edu

Abstract. We consider the problem of testing graph expansion (either vertex or edge) in the bounded degree model [10]. We give a property tester that given a graph with degree bound d, an expansion bound α, and a parameter $\varepsilon > 0$, accepts the graph with high probability if its expansion is more than α, and rejects it with high probability if it is ε-far from any graph with expansion α' with degree bound d, where $\alpha' < \alpha$ is a function of α. For edge expansion, we obtain $\alpha' = \Omega(\frac{\alpha^2}{d})$, and for vertex expansion, we obtain $\alpha' = \Omega(\frac{\alpha^2}{d^2})$. In either case, the algorithm runs in time $\tilde{O}(\frac{n^{(1+\mu)/2}d^2}{\varepsilon\alpha^2})$ for any given constant $\mu > 0$.

1 Introduction

With the presence of large data sets, reading the whole input may be a luxury. It becomes important to design algorithms which run in time that is *sublinear* in (or even independent of) the size of the input. Sublinear algorithms are often achieved by dealing with a relaxed version of the decision problem. In *property testing* [7,14], we wish to accept inputs that satisfy some given property, and reject those that are sufficiently "far" from having that property. There is usually a well-defined notion of the "distance" of an input to a given property. In recent times, many advances have been made on algorithms for testing a variety of combinatorial, algebraic, and geometric properties (see surveys [5,6,13]). For property testing in graphs [7], there has been a large amount of work for testing in *dense graphs*. Here, it is assumed that the graph is given as an adjacency matrix. There are very general results about classes of properties that can tested in time independent of the size of the graph ([1,2]).

The problem of property testing for bounded degree graphs was first dealt with by Goldreich and Ron [8]. The input graph G is assumed that have a constant degree bound d. The graph G is represented by *adjacency lists* - for

⋆ Part of this work was done when the author was at Princeton University.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 527–538, 2008.

every vertex v, there is list of vertices (of size at most d) adjacent to v. This allows testing algorithms to perform walks in the graph G. Given a property $\mathcal{P}$ and positive $\varepsilon < 1$, the graph G is ε-far from having $\mathcal{P}$ if G has to be modified at more than εnd edges for it to have property $\mathcal{P}$. Note that this includes both additions and deletions, and we want to keep the degree bound constant (usually, we require that the degree bound d is preserved). In this model, there aren't any general results about testable properties as in the case of dense graphs. Czumaj and Sohler [3] made the first attempt in this direction, and showed testability results for classes of graphs that do not contain expanders. Using random walks, Goldreich and Ron [9] proved that bipartiteness is testable with $\widetilde{O}(\sqrt{n})$ queries to the graph. In later work, Goldreich and Ron [10] posed the question of testing *expansion*. Given positive parameters $\lambda, \varepsilon < 1$, they provided a $\widetilde{O}(\sqrt{n})$-time algorithm that was *conjectured* to accept every graph G whose second largest eigenvalue $\lambda(G)$ is less than λ, and reject every graph that is ε-far from having second eigenvalue less than λ' (here λ' could be much larger than λ, but $\lambda' \leq \lambda^{\Omega(1)}$). The running time is essentially tight (in n), since it has been proven that a property tester for expansion requires $\Omega(\sqrt{n})$ queries [9].

One of the major parts of the analysis of the algorithm of [9] for bipartiteness deals with the expansion properties of the graph. Their main technique involves performing random walks on the graph. In the adjacency list model, the basic operation that we possess is that of walking in the graph, and random walks seem like a very natural operation to perform. This immediately raises the question of whether random walks can be used to test expansion. Furthermore, the results of [3] show that classes of graphs which do *not* contain expanders can be tested. All of this indicates that, regarding property testing in bounded degree graphs, testing expansion is a very natural and central issue. The problem of designing a property tester for expansion remained open for more than 6 years, until recently, when Czumaj and Sohler [4] provided a tester for *vertex expansion*. We describe this problem more formally below.

We are given an input graph $G = (V, E)$ on n vertices with degree bound d. Assume that d is a sufficiently large constant. Given a cut $(S, \bar{S})$ (where $\bar{S} = V \setminus S$) in the graph, let $E(S, \bar{S})$ be the number of edges crossing the cut. The edge expansion of the cut is $\frac{E(S,\bar{S})}{\min\{|S|,|\bar{S}|\}}$. The edge expansion of the graph is the minimum edge expansion of any cut in the graph. The vertex expansion of the cut is $\frac{|\partial S|}{|S|}$, where ∂S is the set of nodes in $\bar{S}$ that are adjacent to nodes in S. The vertex expansion of the graph is the minimum vertex expansion of any cut in the graph.

Hereafter, when we use the term "graph", we are only concerned with graphs having degree bound d. We are interested in designing a property tester for expansion (either edge or vertex). The graph is represented by an adjacency list, so we have constant time access to the neighbors of any vertex. Given parameters, $\alpha > 0$ and $\varepsilon > 0$, we want to accept to all graphs with expansion greater than α, and reject all graphs that ε-far from having expansion less than $\alpha' < \alpha$ (where α' is some function of α). This means that G has to be changed at least εnd edges

(either removing or adding, keeping the degree bound d) to make the expansion at least α'.

1.1 Our Results

The problem of testing vertex expansion was first discussed by Czumaj and Sohler [4]. Their algorithm was based on that of Goldreich and Ron [10], and they used combinatorial techniques to prove the correctness of their algorithm. Their tester runs in time $O(\alpha^{-2}\varepsilon^{-3}d^2\sqrt{n}\ln(n/\varepsilon))$ and has parameter $\alpha' = \Theta(\frac{\alpha^2}{d^2 \log n})$.

Independently, using the same algorithm but via algebraic proof techniques, we gave an analysis [11] which allowed us to remove the dependence of n in α', and we obtain $\alpha' = \Theta(\frac{\alpha^2}{d^2})$ for vertex expansion and $\alpha' = \Theta(\frac{\alpha^2}{d})$ for edge expansion. This improvement in α' is significant since in most algorithmic applications of expanders, we need the graph to have *constant* expansion, and our property tester allows us to distinguish graphs which have constant expansion from those that are far from having (a smaller) constant expansion.

However, in the initial unpublished version of this paper which appeared as a tech report on ECCC [11], we prove that the tester rejects graphs that are ε-far from any graph of expansion α' with degree bound $2d$, rather than degree bound d. In this version of the paper, in addition to our previous results, we also show how a small modification to our earlier techniques improves the degree bound to d. We recently found out that independently, the degree bound improvement was also obtained by Nachmias and Schapira [12] using a combination of our techniques and those of Czumaj and Sohler.

To describe our results, we set up some preliminaries. Consider the following slight modification of the standard random walk on the graph: starting from any vertex, the probability of choosing any outgoing edge is $1/2d$, and with the remaining probability, the random walk stays at the current node. Thus, for a vertex of degree $d' \leq d$, the probability of a self-loop is $1-d'/2d \geq 1/2$. This walk is symmetric and reversible; therefore, its stationary distribution is uniform over the entire graph. Consider a cut $(S, \bar{S})$ with $|S| \leq n/2$. The *conductance* of this cut is the probability that, starting from the stationary distribution the random walk leaves the set S in one step, conditioned on the event that the starting state is in S. For our chain, the conductance thus becomes $E(S, \bar{S})/2d|S|$, which is just the expansion of the cut divided by $2d$. The conductance of the graph, Φ_G, is the minimum conductance of any cut in the graph.

Our goal is to design a property tester for graph conductance. The tester is given two parameters Φ and ε. The tester must (with high probability[1]) accept if $\Phi_G > \Phi$ and reject if G is ε-far from having $\Phi_G > c\Phi^2$ (for some absolute constant c). Our tester is almost identical to the one described in [10]. Now we present our main result:

Theorem 1. *Given any conductance parameter Φ, and any constant $\mu > 0$, there is an algorithm which runs in time $O(\frac{n^{(1+\mu)/2}\log(n)\log(1/\varepsilon)}{\varepsilon\Phi^2})$ and with high*

[1] Henceforth, "with high probability" means with probability at least 2/3.

probability, accepts any graph with degree bound d whose conductance is at least Φ, and rejects any graph that is ε-far from any graph of conductance at least $c\Phi^2$ with degree bound d, where c is a constant[2] *which depends on μ.*

REMARK: In Theorem 1, even though we have specified μ to be a constant, the theorem still goes through even if μ were a function of n, though naturally the conductance bound degrades. For instance, if $\mu = 1/\log(n)$, then the running time of our algorithm matches that of [4], but the conductance bound becomes $\Omega(\Phi^2/\log(n))$.

In our bounded degree graph model, the following easy relations hold:

$$\text{edge expansion} \;=\; \text{conductance}/2d,$$

$$(\text{vertex expansion})/2 \;\geq\; \text{conductance} \;\geq\; (\text{vertex expansion})/2d.$$

Using these relations, we immediately obtain property testers for vertex and edge expansion for a given expansion parameter α by running the property tester for conductance with parameter $\Phi = \alpha/2d$, and we get the following corollary to Theorem 1:

Corollary 1. *Given any expansion parameter α, and any constant $\mu > 0$, there is an algorithm which runs in time $O(\frac{d^2 n^{(1+\mu)/2} \log(n) \log(1/\varepsilon)}{\varepsilon \alpha^2})$ and with high probability, accepts any graph with degree bound d whose expansion is at least α, and rejects any graph that is ε-far from any graph of expansion at least α' with degree bound d. For edge expansion, $\alpha' = \Omega(\frac{\alpha^2}{d})$, and for vertex expansion, $\alpha' = \Omega(\frac{\alpha^2}{d^2})$.*

Goldreich and Ron's formulation of the problem [10] asks for a property testing algorithm that given a parameter $\lambda < 1$, accepts any graph with second largest eigenvalue (of the transition matrix of the lazy random walk) less than λ, and rejects any graph that is ε-far from having second largest eigenvalue less than λ', for some $\lambda' \leq \lambda^{\Omega(1)}$. Given a graph G, the following well known inequality (see [15]) states that the second largest eigenvalue $\lambda(G)$ satisfies

$$1 - \Phi_G \;\leq\; \lambda(G) \;\leq\; 1 - \Phi_G^2/2.$$

Now, if we assume that $\lambda \leq 1 - \alpha$ for some constant $\alpha > 0$, then we obtain a property tester in the Goldreich-Ron formulation, for $\lambda' = (1 - c^2(1-\lambda)^4/2) \leq \lambda^{\Omega(1)}$, since $\lambda \leq 1 - \Omega(1)$. Here, c is the constant from Theorem 1. We run our property tester for conductance with parameter $\Phi = 1 - \lambda$. For any graph G with $\lambda(G) \leq \lambda$, we have $\Phi_G \geq \Phi$, so the tester accepts G. Any graph G with $\Phi_G \geq c\Phi^2$ has $\lambda(G) \leq \lambda'$, so the tester rejects any graph which is ε-far from having $\lambda(G) \leq \lambda'$. Thus, we have the following corollary to Theorem 1:

Corollary 2. *Given any parameter $\lambda < 1 - \Omega(1)$, and any constant $\mu > 0$, there is an algorithm which runs in time $O(\frac{n^{(1+\mu)/2} \log(n) \log(1/\varepsilon)}{\varepsilon (1-\lambda)^2})$ and with high probability, accepts any graph with degree bound d with $\lambda(G) \leq \lambda$, and rejects any graph that is ε-far from having $\lambda(G) \leq \lambda'$ with degree bound d, for some $\lambda' \leq \lambda^{\Omega(1)}$.*

[2] We can set $c = \mu/400$.

2 Description of the Property Tester

We first define a procedure called VERTEX TESTER which will be used by the expansion tester.

VERTEX TESTER
Input: Vertex $v \in V$.
Parameters: $\ell = 2 \ln n/\Phi^2$ and $m = 8n^{(1+\mu)/2}$.

1. Perform m random walks of length ℓ from s.
2. Let A be the number of pairwise collisions between the endpoints of these walks.
3. The quantity $A/\binom{m}{2}$ is the *estimate* of the vertex tester. If $A/\binom{m}{2} \geq (1 + 2n^{-\mu})/n$, then output **Reject**, else output **Accept**.

Now, we define the Conductance Tester.

CONDUCTANCE TESTER
Input: Graph $G = (V, E)$.
Parameters: $t = \Omega(\varepsilon^{-1})$ and $N = \Omega(\log(\varepsilon^{-1}))$.

1. Choose a set S of t random vertices in V.
2. For each vertex $v \in S$:
   (a) Run VERTEX TESTER on v for N trials.
   (b) If a majority of the trials output **Reject**, then the CONDUCTANCE TESTER aborts and outputs **Reject**.
3. Output **Accept**.

3 Proof of Theorem 1

Before we give the details of the proof, we give a high level exposition of the ideas. We characterize vertices of the graph as *strong* or *weak* (this was already implicit in the ideas of [10]). Random walks of length ℓ starting from strong vertices mix very rapidly, while those from weak vertices do not. We expect the vertex tester to accept strong vertices and reject weak ones.

One of the main differences from the result of Czumaj-Sohler is that we have a very strict definition of strong vertices. We need the mixing from strong vertices to be *very rapid*, and this is what allows us to remove the dependence of n from α'. In the main technical contribution of this paper, we prove that a bad conductance cut will contain a sufficiently large number of weak vertices. We get very strong quantitative bounds using algebraic techniques to analyze the random walks starting from inside the bad cut. We then show that if there are very few weak vertices in G (and therefore, the tester will probably accept the graph), there is a patch-up procedure that can add εnd edges to boost the expansion to α' and preserves the degree bound. This completes the proof.

3.1 Preliminaries

Let us fix some notation. The probability of reaching u by performing a random walk of length l from v is $p^l_{v,u}$. Denote the (row) vector of probabilities $p^l_{v,u}$ by $\boldsymbol{p}_v{}^l$. The *collision probability* for random walks of length l starting from v is denoted by $\gamma_l(v)$ - this is the probability that two independent random walks of length l starting from v will end at the same vertex. It is easy to see that $\gamma_l(v) = \|\boldsymbol{p}_v\| = \sum_v (p^l_{v,u})^2$ (henceforth, we use $\|\cdot\|$ to denote the $\mathcal{L}_2$ norm). Let $\mathbf{1}$ denote the all 1's vector. The norm of the discrepancy from the stationary distribution will be denoted by $\Delta_l(v)$:

$$\Delta_l(v)^2 = \|\boldsymbol{p}_v{}^l - \mathbf{1}/n\|^2 = \sum_{u \in V} (p^l_{v,u} - 1/n)^2 = \sum_{u \in V} (p^l_{v,u})^2 - 1/n = \gamma_l(v) - 1/n.$$

Since l will usually be equal to ℓ, in that case we drop the subscripts (or superscripts). The relationship between $\Delta(v)$ and $\gamma(v)$ is central to the functioning of the tester. The parameter $\Delta(v)$ is a measure of how well a random walk from s mixes. The parameter $\gamma(v)$ can be estimated in sublinear time, and by its relationship with $\Delta(v)$, allows us to test mixing of random walks in sublinear time. The following is basically proven in [10]:

Lemma 1. *The estimate of $\gamma(v)$, viz. $A/\binom{m}{2}$, provided by the* Vertex Tester *lies outside the range $[(1-2n^{-\mu})\gamma(v), (1+2n^{-\mu})\gamma(v)]$ with probability $< 1/3$.*

Proof given in full version. For clarity of notation, we set $\sigma = n^{-\mu/4}$. We now have the following corollary:

Corollary 3. *The following holds with probability at least $5/6$. For all vertices v in the random sample S chosen by the* Conductance Tester*, if $\gamma(v) < (1+\sigma)/n$, then the majority of the N trials of* Vertex Tester *run on v return* **Accept**. *If $\gamma(v) > (1+6\sigma)/n$, then the majority of the N trials of* Vertex Tester *run on v return* **Reject**.

This is an easy consequence of the fact that we run $N = \Omega(\log(\varepsilon^{-1}))$ trials, by an direct application of Chernoff's bound and using Lemma 1. We are now ready to analyze the correctness of our tester.

First, we show the easy part. Let M denote the transition matrix of the random walk. The top eigenvector of M is $\mathbf{1}$. We will also need the matrix $L = I - M$, which is the (normalized) Laplacian (I denotes the identity matrix). The eigenvalues of L are of the form $(1-\lambda)$, where λ is an eigenvalue of M.

Lemma 2. *If $\Phi_G \geq \Phi$, then the* Conductance Tester *accepts with probability at least $2/3$.*

Proof. Let λ_G be the second largest eigenvalue of M. It is well known (see, e.g., [15]) that $\lambda_G \leq 1 - \Phi_G^2/2 \leq 1 - \Phi^2/2$. Thus, we have for any $v \in V$, if $\boldsymbol{e}_v$ denotes the row vector which is 1 on coordinate v and zero elsewhere,

$$\begin{aligned}\|\boldsymbol{p}_v - \mathbf{1}/n\|^2 &= \|(\boldsymbol{e}_v - \mathbf{1}/n)M^\ell\|^2 \\ &\leq \|\boldsymbol{e}_v - \mathbf{1}/n\|^2 \lambda_G^{2\ell} \\ &\leq (1 - \Phi^2/2)^{4\Phi^{-2}\ln n} \\ &\leq 1/n^2.\end{aligned}$$

The second inequality follows because $\boldsymbol{e}_v - \mathbf{1}/n$ is orthogonal to the top eigenvector $\mathbf{1}$. As a result, $\Delta(v)^2 \leq 1/n^2$, and $\gamma(v) < (1+\sigma)/n$ for all $v \in V$. By Corollary 3, the tester accepts with probability at least 2/3. □

We now show that if G is ε-far from having conductance $\Omega(\Phi^2)$, then the tester rejects with high probability. Actually, we will prove the contrapositive : if the tester does *not* reject with high probability, then G is ε-close to having conductance $\Omega(\Phi^2)$. Call a vertex s *weak* if $\gamma(v) > (1+6\sigma)/n$, all others will be called *strong*. Suppose there are more than $\frac{1}{25}\varepsilon n$ weak vertices. Then with probability at least 5/6, the random sample S chosen by the CONDUCTANCE TESTER has a weak vertex, since the sample has $\Omega(\varepsilon^{-1})$ random vertices. Thus, the CONDUCTANCE TESTER will reject with high probability.

Let us therefore assume that there are at most $\frac{1}{25}\varepsilon n$ weak vertices. Now, we will show that εnd edges can be added to make the conductance $\Omega(\Phi^2)$.

3.2 Algebraic Lemmas

We now state and prove the key algebraic lemmas connecting bad conductance cuts to bad mixing. The quantitative bounds given here are the main tool used to prove that if the graph G has few weak vertices, then G is close to being an expander.

Lemma 3. *Consider a set $S \subset V$ of size $s \leq n/2$ such that the cut $(S, \overline{S})$ has conductance less than δ. Then, for any integer $l > 0$, there exists a node $v \in S$ such that $\Delta_l(v) > (2\sqrt{s})^{-1}(1-4\delta)^l$.*

Proof. Denote the size of S by s ($s \leq n/2$). Let us consider the starting distribution $\boldsymbol{p}$ where:

$$p_v = \begin{cases} 1/s & v \in S \\ 0 & v \notin S \end{cases}$$

Let $\boldsymbol{u} = \boldsymbol{p} - \mathbf{1}/n$. Note that $\boldsymbol{u}M^l = \boldsymbol{p}M^l - \mathbf{1}/n$. Let $1 = \lambda_1 \geq \lambda_2 \cdots \geq \lambda_n > 0$ be the eigenvalues of M and $\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_n$ be the corresponding orthogonal unit eigenvectors. Note that $\boldsymbol{f}_1 = \mathbf{1}/\sqrt{n}$. We represent $\boldsymbol{u}$ in the orthonormal basis formed by the eigenvectors of M as $\boldsymbol{u} = \sum_i \alpha_i \boldsymbol{f}_i$. Here, $\alpha_1 = 0$, since $\boldsymbol{u} \cdot \mathbf{1} = 0$.

$$\begin{aligned}\sum_i \alpha_i^2 &= \|\boldsymbol{u}\|_2^2 \\ &= s\left(\frac{1}{s} - \frac{1}{n}\right)^2 + \frac{n-s}{n^2} \\ &= \frac{1}{s} - \frac{1}{n}.\end{aligned}$$

Taking the Rayleigh quotient with the Laplacian L:

$$\begin{aligned}\boldsymbol{u}^\top L\boldsymbol{u} &= \boldsymbol{u}^\top I\boldsymbol{u} - \boldsymbol{u}^\top M\boldsymbol{u}\\ &= \|\boldsymbol{u}\|_2^2 - \sum_i \alpha_i^2\lambda_i.\end{aligned}$$

On the other hand, using the fact that the conductance of the cut $(S, \bar{S})$ is less than δ, we have

$$\boldsymbol{u}^\top L\boldsymbol{u} \;=\; \sum_{i<j} M_{ij}(u_i - u_j)^2 \;<\; 2\delta ds \times \frac{1}{2d} \times \frac{1}{s^2} \;=\; \frac{\delta}{s}.$$

Putting the above together:

$$\begin{aligned}\sum_i \alpha_i^2\lambda_i &> \left(\frac{1}{s} - \frac{1}{n}\right) - \frac{\delta}{s}\\ &= \frac{1-\delta}{s} - \frac{1}{n}.\end{aligned}$$

If $\lambda_i > (1 - 4\delta)$, call it *heavy*. Let H be the index set of heavy eigenvalues, and $\bar{H}$ be the index set of the rest. Since $\sum_i \alpha_i^2\lambda_i$ is large, we expect many of the α_i corresponding to heavy eigenvalues to be large. This would ensure that the starting distribution $\boldsymbol{p}$ will not mix rapidly. We have

$$\sum_{i\in H} \alpha_i^2\lambda_i + \sum_{i\in\bar{H}} \alpha_i^2\lambda_i \;>\; \frac{1-\delta}{s} - \frac{1}{n}.$$

Setting $x = \sum_{i\in H} \alpha_i^2$:

$$x + (\sum_i \alpha_i^2 - x)(1 - 4\delta) \;>\; \frac{1-\delta}{s} - \frac{1}{n}.$$

We therefore get:

$$\begin{aligned}4\delta x + \left(\frac{1}{s} - \frac{1}{n}\right)(1 - 4\delta) &> \frac{1-\delta}{s} - \frac{1}{n}\\ \therefore\; x &> \frac{3}{4s} - \frac{1}{n}\\ &\geq \frac{1}{4s}. \qquad \because n \geq 2s \end{aligned} \tag{1}$$

Note that $\boldsymbol{u}M^l = \sum_i \alpha_i\lambda^l\boldsymbol{f}_i$. Thus,

$$\begin{aligned}\|\boldsymbol{u}M^l\|_2^2 &= \sum_i \alpha_i^2\lambda_i^{2l}\\ &\geq \sum_{i\in H} \alpha_i^2\lambda_i^{2l}\\ &> \frac{1}{4s}(1 - 4\delta)^{2l}.\end{aligned}$$

So, $\|\boldsymbol{u}M^l\|_2 > \frac{1}{2\sqrt{s}}(1-4\delta)^l$. Note that $\boldsymbol{u} = \frac{1}{s}\sum_{v\in S}(\boldsymbol{e}_v - \frac{\mathbf{1}}{n})$, and hence $\boldsymbol{u}M^l = \frac{1}{s}\sum_{v\in S}(\boldsymbol{e}_v M^l - \frac{\mathbf{1}}{n})$. Now, $\boldsymbol{e}_v M^l - \frac{\mathbf{1}}{n}$ is the discrepancy vector of the probability distribution of the random walk starting from v after l steps. Thus, by Jensen's inequality, we conclude that

$$\frac{1}{s}\sum_{v\in S}\Delta_l(v) \;\geq\; \|\boldsymbol{u}M^l\| \;>\; \frac{1}{2\sqrt{s}}(1-4\delta)^l.$$

Hence, there is some $v \in S$ for which $\Delta_l(v) > (2\sqrt{s})^{-1}(1-4\delta)^l$. □

Lemma 4. *Consider sets $T \subseteq S \subseteq V$ such that the cut $(S, \bar{S})$ has conductance less than δ. Let $|T| = (1-\theta)|S|$. Assume $0 < \theta \leq \frac{1}{8}$. Then, for any integer $l > 0$, there exists a node $v \in T$ such that $\Delta_l(v) > \frac{(1-2\sqrt{2\theta})}{2\sqrt{s}}(1-4\delta)^l$.*

Proof. Let $\boldsymbol{u}_S$ (resp., $\boldsymbol{u}_T$) be the uniform distribution over S (resp., T) minus $\frac{\mathbf{1}}{n}$. Let s and t be the sizes of S and T resp. Let $\boldsymbol{u}_S = \sum_i \alpha_i \boldsymbol{f}_i$ and $\boldsymbol{u}_T = \sum_i \beta_i \boldsymbol{f}_i$ be representation of $\boldsymbol{u}_S$ and $\boldsymbol{u}_T$ in the basis $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n\}$, the unit eigenvectors of M. Note that $\alpha_1 = \beta_1 = 0$ since $\boldsymbol{u}_S$ and $\boldsymbol{u}_T$ are orthogonal to $\mathbf{1}$.

Since the conductance of S is less than δ, by applying inequality (11) from Lemma 3, we have that

$$\sum_{i\in H}\alpha_i^2 > \frac{1}{4s}.$$

We have

$$\|\boldsymbol{u}_S - \boldsymbol{u}_T\|^2 \;=\; \frac{1}{t} - \frac{1}{s} \;=\; \frac{\theta}{(1-\theta)s} \;\leq\; \frac{2\theta}{s}.$$

Furthermore,

$$\|\boldsymbol{u}_S - \boldsymbol{u}_T\|^2 \;=\; \sum_i(\alpha_i - \beta_i)^2 \;\geq\; \sum_{i\in H}(\alpha_i - \beta_i)^2.$$

Using the triangle inequality $\|\boldsymbol{a} - \boldsymbol{b}\| \geq \|\boldsymbol{a}\| - \|\boldsymbol{b}\|$, we get that

$$\sum_{i\in H}\beta_i^2 \;\geq\; \left[\sqrt{\sum_{i\in H}\alpha_i^2} - \sqrt{\sum_{i\in H}(\alpha_i-\beta_i)^2}\right]^2 \;>\; \left[\frac{1}{2\sqrt{s}} - \frac{\sqrt{2\theta}}{\sqrt{s}}\right]^2 \geq \frac{(1-2\sqrt{2\theta})^2}{4s}.$$

Finally, reasoning as in Lemma 3, we get that $\|\boldsymbol{u}_T M^l\| > \frac{(1-2\sqrt{2\theta})}{2\sqrt{s}}(1-\delta)^l$, and thus, by Jensen's inequality, there is a $v \in T$ such that $\Delta_l(v) > \frac{(1-2\sqrt{2\theta})}{2\sqrt{s}}(1-4\delta)^l$. □

This lemma immediately yields the following corollary:

Corollary 4. *Consider a set $S \subseteq V$ such that the cut $(S, \bar{S})$ has conductance less than δ. For positive $\theta \leq \frac{1}{8}$ and any integer $l > 0$, there exist at least $\theta|S|$ nodes $v \in S$ such that $\Delta_l(v) > \frac{(1-2\sqrt{2\theta})}{2\sqrt{s}}(1-4\delta)^l$.*

Using the above lemmas, we can now show that G looks almost like an expander.

Lemma 5. *There is a partition of the graph G into two pieces, A and $\bar{A} := V \setminus A$, with the following properties:*

1. $|A| \leq \frac{2}{5}\varepsilon n$.
2. *Any cut in the induced subgraph on $\bar{A}$ has conductance $\Omega(\Phi^2)$.*

Proof. We use a recursive partitioning technique: start out with $A = \{\}$. Let $\bar{A} = V \setminus A$. If there is a cut $(S, \bar{S})$ in $\bar{A}$ with $|S| \leq |\bar{A}|/2$ with conductance less than $c\Phi^2$, then we set $A := A \cup S$, and continue as long as $|A| \leq n/2$. Here, c is a small constant to be chosen later.

We claim that the final set A has the required properties. If $|A| > \frac{2}{5}\varepsilon n$, then consider the cut $(A, \bar{A})$ in G. It has conductance at most $c\Phi^2$. Now, Corollary 4 implies (with $\theta = 1/10$) that there are at least $\frac{1}{10}|A| > \frac{1}{25}\varepsilon n$ nodes in A such that for all such nodes v, and for $b = \frac{(1-2\sqrt{1/5})}{\sqrt{2}}$, we have

$$\Delta_\ell(v) \;>\; \frac{b}{\sqrt{n}}(1 - 4c\Phi^2)^\ell \;>\; \sqrt{6\sigma/n}$$

for a suitable choice of c in terms of μ (say, $c = \mu/200$ suffices).

Thus, for all such nodes v, we have $\gamma_\ell(v) = \Delta_\ell(v)^2 + 1/n > (1 + 6\sigma)/n$, which implies that all such nodes are weak, a contradiction since there are only $\frac{1}{25}\varepsilon n$ weak nodes.

Since $|A| \leq \frac{2}{5}\varepsilon n < n/2$, when the recursive partitioning procedure terminates, any cut in the induced subgraph on $\bar{A}$ has conductance $\Omega(\Phi^2)$. □

3.3 Getting an Expander

Armed with the partitioning algorithm of Lemma 5, we are ready to present the patch-up algorithm, which changes the graph in εnd edges and raises its conductance to $\Omega(\Phi^2)$. Note that we do not perform this patch-up algorithm as part of our tester. It is merely used to show that G is close to an expander. The trivial patch-up algorithm would just add d random edges to every vertex in A. This would only add at most εnd edges and make the conductance $\Omega(\Phi^2)$. The drawback is that the degree bound will not be preserved. We have to be more careful to ensure that we can find a graph G' ε-close to G which is an expander *and* has a degree bound of d.

PATCH-UP ALGORITHM

1. Partition the graph into two pieces A and $\bar{A}$ with the properties given in Lemma 5.
2. Remove all edges incident on nodes in A.
3. For each node $u \in A$, repeat the following process until the degree of u becomes $d - 1$ or d: choose a vertex $v \in \bar{A}$ at random. If the current degree of v is less than d, add the edge $\{u, v\}$. Otherwise, if there is an edge $\{v, w\}$ such that $w \in \bar{A}$, remove $\{v, w\}$, and add the edges $\{u, v\}$ and $\{u, w\}$ (call these newly added edges "paired"). Otherwise, re-sample the vertex v from $\bar{A}$, and repeat.

To implement Step 3, we need to ensure that the set of nodes in $\bar{A}$ with degree less than d or having an edge to another node in $\bar{A}$ is non-empty. In fact, we can show a stronger fact:

Lemma 6. *At any stage in the patch-up algorithm, there are at least* $\frac{1}{4}|\bar{A}| \geq \frac{1}{4}(1 - 2\varepsilon/5)n$ *nodes in* $\bar{A}$ *with degree less than* d *or having an edge to another node in* $\bar{A}$.

Proof. Let $X \subseteq \bar{A}$ be the set of nodes of degree at most $d/2$ before starting the second step, and let $Y := \bar{A} \setminus X$. Now we have two cases:

1. $|X| \geq \frac{1}{2}|\bar{A}|$: We add at most $\frac{2}{5}\varepsilon nd$ edges, since $|A| \leq \frac{2}{5}\varepsilon n$. At most half the nodes in X can have their degree increased to d, since $\frac{2}{5}\varepsilon nd \leq \frac{1}{2}|X| \cdot \frac{d}{2}$, since $|X| \geq \frac{1}{2}(1 - 2\varepsilon/5)n$. Here, we assume that $\varepsilon \leq 1/4$. Thus, at any stage we have at least $\frac{1}{4}|\bar{A}|$ nodes with degree less than d.
2. $|Y| \geq \frac{1}{2}|\bar{A}|$: we remove at most $\frac{1}{5}\varepsilon nd$ edges from the subgraph induced by $\bar{A}$. At most half of the nodes in Y can have their (induced) degrees reduced to 0, $\frac{1}{5}\varepsilon nd \leq \frac{1}{2}|Y| \cdot \frac{d}{2}$, since $|Y| \geq \frac{1}{2}(1 - 2\varepsilon/5)n$. Again, we assume that $\varepsilon \leq 1/4$. Thus, at any stage we have at least $\frac{1}{4}|\bar{A}|$ nodes with at least one edge to some other node in $\bar{A}$.

Now, we prove that the patch-up algorithm works:

Theorem 2. *If there are less than* $\frac{1}{25}\varepsilon n$ *weak vertices, then* εnd *edges can be added or removed to make the conductance* $\Omega(\Phi^2)$*, while ensuring that all degrees are at most* d.

Proof. We run the patch-up algorithm on the given graph. It is easy to see that at the end of the algorithm, every node has degree bounded by d. Also, the total number of edges deleted is at most $\frac{2}{5}\varepsilon nd + \frac{1}{5}\varepsilon nd$, and the number of edges added is at most $\frac{2}{5}\varepsilon nd$. Thus the total number of edges changed is at most εnd.

Now, let $(S, \bar{S})$ be a cut in the graph with $|S| \leq n/2$. Let $S_A = S \cap A$, and $S_{\bar{A}} = S \cap \bar{A}$. Let $m := |S|$. We have two cases now:

1. $|S_{\bar{A}}| \geq m/2$: In this case, note that in the subgraph of original graph induced on $\bar{A}$, the set $S_{\bar{A}}$ had conductance at least $c\Phi^2$, and hence the cut $(S_{\bar{A}}, \bar{A} \setminus S_{\bar{A}})$ had at least $2c\Phi^2|S_{\bar{A}}|d \geq c\Phi^2 md$ edges crossing it.
   For any edge $\{v, w\}$ that was in the cut $(S_{\bar{A}}, \bar{A} \setminus S_{\bar{A}})$ and was removed by the construction, we added two new edges $\{u, v\}$ and $\{u, w\}$ for some $u \in A$. Now it is easy to check that regardless of whether $u \in S_A$ or $u \notin S_A$, one of the two edges $\{u, v\}$ and $\{u, w\}$ crosses the cut $(S, \bar{S})$. Thus, at least $c\Phi^2 md$ edges cross the cut $(S, \bar{S})$, and hence it has conductance at least $\frac{c}{2}\Phi^2$.
2. $|S_{\bar{A}}| \leq m/2$: In this case, for each node $u \in S_A$, we chose at least $d/2$ random edges connecting u to nodes in $\bar{A}$ (for now, disregarding one edge in every set of paired edge from step 3.). By Lemma 6, and since $|S_{\bar{A}}| \leq |S_A| \leq |A| \leq 2\varepsilon n/5$, the probability that for any such edge, the endpoint in $\bar{A}$ was actually in $S_{\bar{A}}$ is at most
$$\frac{|S_{\bar{A}}|}{\frac{1}{4}|\bar{A}|} \leq \frac{2\varepsilon/5}{\frac{1}{4}(1 - 2\varepsilon/5)} \leq 1/4$$
   assuming $\varepsilon \leq 1/8$.

Since $|S_A| \geq m/2$, the total number of edges added to nodes in S_A is at least $md/4$ (again, disregarding one edge out of every set of paired edges). The expected number of these edges going into $S_{\bar{A}}$ is at most $md/16$. By the Chernoff-Hoeffding bounds, the probability that more than $md/8$ randomly chosen edges lie completely in S is less than $n^{-\Omega(md)} \leq 1/3n^{m+1}$, if we assume d is at least a large enough constant.

Taking a union bound over all sets of size m (the number of which is at most n^m), and then summing over all m, we get the with probability at least $2/3$, none of these events happen, and thus at least at least $md/8$ edges cross the cut $(S, \overline{S})$. Therefore, the conductance of this cut is at least $1/16 > \Omega(\Phi^2)$, since $\Phi \leq 1$. □

References

1. Alon, N., Fischer, E., Newman, I., Shapira, A.: A Combinatorial Characterization of the Testable Graph Properties: it's all about Regularity. In: Proc. 38th STOC, pp. 251–260 (2006)
2. Alon, N., Shapira, A.: A Charaterization of the (Natural) Graph Properties Testable with One-sided Error. In: Proc. 46th FOCS, pp. 429–438 (2005)
3. Czumaj, A., Sohler, C.: On Testable Properties in Bounded Degree Graphs. In: Proc. 18th SODA, pp. 494–501 (2007)
4. Czumaj, A., Sohler, C.: Testing Expansion in Bounded Degree Graphs. In: Proc. 48th FOCS, pp. 570–578 (2007)
5. Fischer, E.: The Art of Uninformed Decisions: A Primer to Property Testing. Bulletin of EATCS 75, 97–126 (2001)
6. Goldreich, O.: Combinatorial property testing - A survey. In: Randomization Methods in Algorithm Design, pp. 45–60 (1998)
7. Goldreich, O., Goldwasser, S., Ron, D.: Property Testing and its Connection to Learning and Approximation. J. ACM 45, 653–750 (1998)
8. Goldreich, O., Ron, D.: Property Testing in Bounded Degree Graphs. Algorithmica 32(2), 302–343 (2002)
9. Goldreich, O., Ron, D.: A Sublinear Bipartiteness Property Tester for Bounded Degree Graphs. Combinatorica 19(3), 335–373 (1999)
10. Goldreich, O., Ron, D.: On Testing Expansion in Bounded-Degree Graphs. ECCC, TR00-020 (2000)
11. Kale, S., Seshadhri, C.: Testing Expansion in Bounded Degree Graphs. ECCC, TR07-076 (2007)
12. Nachmias, A., Shapira, A.: Testing the Expansion of a Graph ECCC, TR07-118 (2007)
13. Ron, D.: Property testing. In: Handbook on Randomization, vol. II, pp. 597–649 (2001)
14. Rubinfeld, R., Sudan, M.: Robust characterization of polynomials with applications to program testing. SIAM J. Comput. 25, 647–668 (1996)
15. Sinclair, A.: Algorithms for Random Generation and Counting: a Markov Chain Approach. Birkhaüser Progress In Theoretical Computer Science Series (1993)

Property Testing on k-Vertex-Connectivity of Graphs

Yuichi Yoshida and Hiro Ito

School of Informatics, Kyoto University, Kyoto 606-8501, Japan
{yyoshida@lab2.,itohiro@}kuis.kyoto-u.ac.jp

Abstract. We present an algorithm for testing the k-vertex-connectivity of graphs with given maximum degree. The time complexity of the algorithm is independent of the number of vertices and edges of graphs. A graph G with n vertices and maximum degree at most d is called ϵ-far from k-vertex-connectivity when at least $\frac{\epsilon dn}{2}$ edges must be added to or removed from G to obtain a k-vertex-connected graph with maximum degree at most d. The algorithm always accepts every graph that is k-vertex-connected and rejects every graph that is ϵ-far from k-vertex-connectivity with a probability of at least 2/3. The algorithm runs in $O\left(d\left(\frac{c}{\epsilon d}\right)^k \log \frac{1}{\epsilon d}\right)$ time ($c > 1$ is a constant) for given $(k-1)$-vertex-connected graphs, and $O\left(d\left(\frac{ck}{\epsilon d}\right)^k \log \frac{k}{\epsilon d}\right)$ time ($c > 1$ is a constant) for given general graphs. It is the first constant-time k-vertex-connectivity testing algorithm for general $k \geq 4$.

1 Introduction

In this paper we present a constant-time k-vertex-connectivity testing algorithm for general constant $k \geq 4$. For such problems only $k \leq 3$ have been solved so far.

An n-vertex graph G is called *k-vertex-connected* (or simply, *k-connected*) if $n \geq k+1$ and deletion of any $k-1$ or fewer vertices leaves a connected graph. The maximum number k for which G is k-connected is called *vertex-connectivity* (or simply, *connectivity*) of G. A vertex set is called *vertex cut* if when removing them the resulting graph is not connected.

Property testing [14] is a relaxation of deciding problems that distinguish that an object (e.g. graph) has predetermined property $\mathcal{P}$ and that it has a large distance to having $\mathcal{P}$. The distance between an object and a property is parametrized by a positive real number $0 < \epsilon \leq 1$. An object is called ϵ-far from property $\mathcal{P}$ if it differs in an ϵ-fraction of description from any object having the property $\mathcal{P}$. Property testing algorithms with a given constant ϵ **accept** every object that satisfies the property $\mathcal{P}$ with a probability of at least $\frac{2}{3}$, and **reject** every object that is ϵ-far from the property $\mathcal{P}$ with a probability of at least $\frac{2}{3}$. It is known that various properties are testable in constant running time, i.e., the size of input objects does not matter. Excellent surveys in this area are available [5,6].

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 539–550, 2008.

Table 1. Summary of testing algorithm for k-connectivity

k	Connectivity of given graphs	Running time	Reference
1	any	$O\left(\frac{\log^2 1/(\epsilon d)}{\epsilon}\right)$	[7]
2	any	$\min\left\{O\left(\frac{\log(1/(\epsilon d))}{\epsilon^2 d}\right), O\left(\frac{2^d log^2(1/(\epsilon d))}{\epsilon}\right)\right\}$	[7]
3	any	$\min\left\{O\left(\frac{\log(1/(\epsilon d))}{\epsilon^3 d^2}\right), O\left(\frac{2^{2d} log(1/(\epsilon d))}{\epsilon^2 d}\right)\right\}$	[7]
any	$k-1$	$O\left(d\left(\frac{c}{\epsilon d}\right)^k \log\frac{1}{\epsilon d}\right)$ ($c>1$ is a constant)	This work
any	any	$O\left(d\left(\frac{ck}{\epsilon d}\right)^k \log\frac{k}{\epsilon d}\right)$ ($c>1$ is a constant)	This work

So far, some models have been proposed for graphs. The *adjacency matrix model* is a well-studied model for dense graphs. In this model, a graph is represented as an adjacency matrix that has n^2 entries. A testing algorithm is allowed to access any entry in constant time. A graph is called ϵ-far from property $\mathcal{P}$ if at least $\frac{\epsilon n^2}{2}$ edge modifications (additions and deletions) are needed to make G satisfy $\mathcal{P}$. However, when it comes to k-connectivity, any graph can be modified to a k-connected graph by adding at most $\frac{nk}{2}$ edges. Thus, for any graph, $\epsilon \leq \frac{k}{n}$ and it follows $n \leq \frac{k}{\epsilon}$. Hence, n can be regarded as a constant for fixed k and ϵ. Thus, it is possible to construct a testing algorithm that runs in constant time using the usual polynomial time algorithms such as max flow. This makes it meaningless to argue about testing k-connectivity in the adjacency matrix model.

A suitable model for testing k-connectivity of graphs is the *bounded degree model* introduced in [7]. In this model, a graph is represented by an adjacency list and the vertex degrees are bounded by a constant d. A testing algorithm has a constant-time access to any entry in the adjacency list by making a query to the ith neighbor of a vertex v. A graph is called ϵ-far from property $\mathcal{P}$, if it is necessary to modify at least ϵ-fraction of the adjacency list to make G satisfy property $\mathcal{P}$. An equivalent condition is that the minimum number of edges to be added or removed, preserving the maximum degree d, to make G having property $\mathcal{P}$ is at least $\frac{\epsilon dn}{2}$. Using this model, it is known that many properties are testable in constant time, such as k-edge-connectivity and cycle-freeness [7]. Also, various properties are testable in sublinear time, such as bipartiteness [8] and expansion [9,4].

Property testing on k-connectivity is considered in [7]. They developed an algorithm for $k = 1, 2, 3$ that runs in constant time. In this paper, we extended their work, and show an algorithm available for any k. Its running time is also constant, assuming k is constant. Previous results and our results are summarized in Table 1. It may seem strange that the running time decreases when d gets bigger because it appears that a graph can be tested more quickly for larger d. However, this is not true. Since the distance between an object and k-connectivity is measured as a fraction of dn, fixed ϵ, a graph that is ϵ-far from k-connectivity, is no longer ϵ-far for some larger d.

Property k-edge-connectivity is similar to k-connectivity. A graph G is called k-edge-connected if deletion of any $k-1$ or fewer edges leaves a connected graph. A testing algorithm for k-edge-connectivity is presented by Goldreich and Ron [7]. Its running time is $O\left(\frac{k^3 \log^2(1/\epsilon d)}{\epsilon^{3-\frac{2}{k}} d^{2-\frac{2}{k}}}\right)$ for general $k \geq 4$. They introduced an excellent idea such that if a graph G is ϵ-far from k-edge-connectivity, then G has sufficiently many small deficient sets. Basically, our algorithm uses this idea. However, the structure of k-vertex-connectivity is far more complicated than that of k-edge-connectivity. In particular, they use the fact that vertices can be decomposed into equivalent classes with respect to edge-connectivity, and the structure of equivalent classes can be represented as a cactus tree. Unfortunately, it is hard to define such equivalent classes with respect to vertex-connectivity because vertex-connectivity between two vertices does not satisfy transitivity, unlike edge-connectivity. Instead, we employ facts known in the area of *edge augmentation*. The edge augmentation problem for k-connectivity is to find the smallest set F of edges such that $G' = (V, E + F)$ is k-connected. This problem is well studied. See [12,13,10,11].

This paper is organized as follows. In Section 2, we present definitions and terminologies that will be used in the rest of paper. In Section 3, we present an algorithm for testing k-connectivity given $(k-1)$-connected graphs. In Section 4, we present an algorithm for testing k-connectivity given general graphs. We give our conclusions in Section 5.

2 Preliminaries

Let $G = (V, E)$ be a graph and n denote the number of vertices. Since we are concerned with vertex connectivity, we deal with *simple* graphs only. Let $\deg(v)$ for $v \in V$ denote the degree of v. For a vertex set $P \subseteq V$, we use $\Gamma(P)$ to denote the *neighbors* of P, i.e., $\Gamma(P) = \{v \in V \setminus P | \exists u \in P, uv \in E\}$. P is called a *fragment* if $P, V - P - \Gamma(P) \neq \emptyset$.

Let $m_k(G)$ denote the minimum number of edges that must be added to G to obtain a k-connected graph. Let $M_k^d(G)$ denote the minimum number of edges that must be added to or removed from G to obtain a k-connected graph with maximum degree d. When the context is clear, k and d may be omitted. Clearly, if G is ϵ-far from k-connectivity, $\frac{\epsilon dn}{2} \leq M_k^d(G)$ by the definition. A fragment P is called *deficient* if $|\Gamma(P)| < k$.

Let $G = (V, E)$ be a $(k-1)$-connected graph. A fragment P in G is called *tight* if $\Gamma(P) = k-1$. Tight sets are deficient. The maximum number of disjoint tight sets is denoted by $t(G)$. If a tight set P does not contain any other tight set, P is called a *minimal tight set*. In general, a minimal tight set may intersect other minimal tight sets. However, if $t(G)$ is large, the next proposition holds.

Proposition 1. [12] *Let $G = (V, E)$ be a $(k-1)$-connected graph. If $t(G) \geq k$, then all minimal tight sets in G are pairwise disjoint.* □

We say that $S \subseteq V$ is a *tight set cover* if $S \cap P \neq \emptyset$ for every tight set P. Let $\tau(G)$ be the size of the minimum tight set cover. For a set $K \subseteq V$, $b_K(G)$ denotes the number of components in $G - K$. Let $b(G) = \max\{b_K(G) | K \subseteq V, |K| = k-1\}$.

Two lower bounds for $m(G)$ are a function of $t(G)$ and $b(G)$. Since one edge can connect at most two disjoint tight sets, $m(G) \geq \left\lceil \frac{t(G)}{2} \right\rceil$. For all $K \subseteq V$ with size $k-1$, $G-K$ must be connected in the augmented graph. Thus $m(G) \geq b(G)-1$. It clearly follows that

$$m(G) \geq \max \left\{ \left\lceil \frac{t(G)}{2} \right\rceil, b(G)-1 \right\}. \tag{1}$$

3 Testing k-Vertex-Connectivity of $(k-1)$-Vertex-Connected Graphs

In this section, we argue about testing the k-connectivity of given $(k-1)$-connected graphs.

Our algorithm depends on the fact that a $(k-1)$-connected graph that is ϵ-far from k-connectivity has $\Omega(\epsilon dn)$ disjoint minimal tight sets. Thus, if a given graph is not k-connected, one of the uniformly selected $\Theta(\frac{1}{\epsilon d})$ vertices belongs to a tight set with high probability. If we can identify the tight set from a vertex in it, we can detect that the graph is not k-connected with high probability.

In the first half of this section, we show that there exist $\Omega(\epsilon dn)$ tight sets in a graph that is ϵ-far from k-connectivity. In the last half of this section, we show an algorithm for identifying tight sets and the entire algorithm for testing the k-connectivity of $(k-1)$-connected graphs.

3.1 The Number of Disjoint Minimal Tight Sets in Graphs That Are ϵ-Far from k-Connectivity

We discuss two cases with $d \geq k+1$ and $d = k$ separately. First, we consider the case with $d \geq k+1$. Let G be a graph that is ϵ-far from k-connectivity. At least $M(G) \geq \frac{\epsilon dn}{2}$ edges must be modified to obtain a k-connected graph with maximum degree d. To see the relation between ϵ (or, $M(G)$) and the number of disjoint minimal tight sets, we use $m(G)$, which is defined in the previous section, as an intermediate tool. After getting the relation between $M(G)$ and $m(G)$, we argue about the relation between $m(G)$ and $t(G)$.

Let G' be a k-connected graph that results from adding $m(G)$ edges to G. G' may have vertices with degree greater than d. By counting the number of edges that have to be modified to make the degree of all vertices be at most k, with preserving k-connectivity, we have a bound on $M(G)$ in terms of $m(G)$. To do this, we use the *splitting off* method. A splitting off is removing two edges su, sv and adding uv. An edge is called *critical* if removing the edge decreases the connectivity of the graph. Next the lemma is known.

Lemma 2. [1, 3] *Let $G = (V, E)$ be a k-connected graph with $n \geq 2k$ and let $r \in V$ be a vertex such that $deg(r) \geq k+2$ and every edge incident to r is critical with respect to k-connectivity. Then either:*

1. *there exists a pair of edges incident to r that can be split off preserving k-connectivity, or*
2. *for any edge pair ru, rv, there exists another edge pair sw, sz such that splitting off both edge pairs preserves k-connectivity,*

holds. □

Theorem 3. *Let $G = (V, E)$ be a graph with $n \geq 2k$ and maximum degree $d \geq k + 1$. Then $M(G) \leq 13m(G)$.*

Proof. Let G' be a k-connected graph that results from adding $m(G)$ edges to G. We define *excess* of G'(with respect to degree bound d) as

$$\sum_{v \in V} \max(\deg(v) - d, 0).$$

Since adding an edge to the graph increases the degree of exactly 2 vertices by 1, excess of G' is at most $2m(G)$. We show how to decrease excess of G' preserving k-connectivity. When the excess is 0, we have a k-connected graph with maximum degree d.

Let v be a vertex whose degree is greater than d. If some edge incident to v is not critical, we can remove the edge and decrease the excess by at least one. If all edges incident to v are critical, using Lemma 2, we can decrease the degree of v with at most two splitting off. Since one splitting off causes three edge modifications, we can decrease the excess by at least one with at most six edge modifications. Thus, in all cases six edge modifications are sufficient to decrease excess by one.

Since the initial excess of G' is at most $2m(G)$, $12m(G)$ edge modifications is sufficient to obtain a graph with maximum degree d. Hence, $M(G) \leq m(G) + 12m(G) = 13m(G)$. □

Next two lemmas connect $m(G)$ and $t(G)$.

Lemma 4. [13] *Let G be a $(k-1)$-connected graph. Then $m(G) \leq \tau(G) - 1$.* □

Lemma 5. [13] *If $t(G) \leq k$ for a $(k-1)$-connected graph G, then $\tau(G) \leq k$.* □

Suppose $t(G) \geq k$. Hence, minimal tight sets are pairwise disjoint from Proposition 1. From this, we can obtain $\tau(G) = t(G)$ easily. Using this fact and the above two lemmas, we have a lower bound on $t(G)$ in terms of $m(G)$ as follows.

Lemma 6. *Let G be a $(k-1)$-connected graph. Then $m(G) < \max\{k, t(G)\}$.*

Proof. By Lemma 4, $m(G) \leq \tau(G) - 1$. If $t(G) < k$, then $\tau(G) \leq k$ by Lemma 5. Thus, $m(G) < k$. If $t(G) \geq k$, every minimal tight sets is pairwise disjoint and $t(G) = \tau(G)$. Hence, $m(G) < t(G)$. □

From Theorem 3 and Lemma 6, the next theorem is easily obtained.

Theorem 7. *Let $G = (V, E)$ be a $(k-1)$-connected graph with $n \geq 2k$ and maximum degree $d \geq k + 1$. Then, $M(G) \leq 13 \max\{k, t(G)\}$.* □

Next, we show that similar theorems hold when $d = k$. We want to know how many edge modifications are sufficient to obtain a k-connected graph with maximum degree k. In this case, however, the splitting off method does not work anymore because Lemma 2 works for vertices having degrees greater than $k+1$. Thus, we must employ another method.

When $m(G)$ is large enough, $m(G)$ is equal to its lower bound (1) as shown in the following lemma.

Lemma 8. [11] *Let G be a $(k-1)$-connected graph with $n \geq k+1$ vertices. If $m(G) \geq 20k^3$ then, $m(G) = \max\left\{\left\lceil \frac{t(G)}{2} \right\rceil, b(G) - 1\right\}$.* □

Let K be a vertex cut of G with $|K| = k-1$ and C_K be a set of connected components of $G - K$. Since G is $(k-1)$-connected, each vertex in K must be incident to every component of C_K. (Otherwise, let v be a vertex in K that is not incident to some component of C_K, then $K - v$ separates G, contradicting $(k-1)$-connectivity of G.) Thus, $b(G) \leq k$ because the maximum degree of G is k. It follows that $m(G) = \left\lceil \frac{t(G)}{2} \right\rceil$ if $m(G) \geq 20k^3$.

Suppose that every minimal tight set in G is a singleton (i.e., one vertex of degree $k-1$). Added $m(G) = \left\lceil \frac{t(G)}{2} \right\rceil$ edges form parings of singletons. Thus, none of the vertices in the resulting graph has a degree greater than k.

When $m(G)$ is small, we remove edges and artificially increase $m(G)$. A graph is called *minimally k-connected* if it is k-connected but if removing any of the edges the resulting graph is no longer k-connected. The following lemma holds.

Lemma 9. [2] *A minimally $(k-1)$-connected graph has at least $\frac{(k-2)n+2}{2k-3}$ vertices of degree $k-1$.* □

A vertex of degree $k-1$ is a tight set itself. Thus, by removing edges we can increase the number of tight sets and $m(G)$ also increases. If n is large enough, we can remove edges until $m(G)$ gets sufficiently large.

Now, we can prove the next theorem.

Theorem 10. *Let $G = (V, E)$ be a $(k-1)$-connected graph with $n \geq 120k^3$ and maximum degree k, and n or k is even*[1]*. Then $M(G) \leq \max\{200k^3, 4t(G)\}$*

Proof. We consider two cases in terms of $m(G)$.

1. If $m(G) \geq 20k^3$: If $t(G) < k$, then by Lemmas 4 and 5, $m(G) \leq \tau(G) - 1 \leq k-1$. This is a contradiction. Thus $t(G) \geq k$. It follows that minimal tight sets are pairwise disjoint and no tight set intersects other minimal tight sets. Let T be a minimal tight set that is not a singleton (i.e., $|T| \geq 2$). The subgraph induced by T is connected, and hence some edge e exists in T. Since e does not cross the boundary of any (not only minimal) tight set, removing e from G preserves $(k-1)$-connectivity. Since one edge connects at most two disjoint tight sets, removing one edge increases the number of minimal tight sets by at most two. These two minimal tight sets are singletons because

[1] Note that there is no n-vertex k-regular graph if n and k are both odd.

original vertex degrees are bounded by $d = k$. Thus, the number of minimal tight sets that are not singletons must decrease. By removing at most $t(G)$ edges, we have a graph in which all minimal tight sets are singletons.
Let G' be the resulting graph made from G by removing an edge from each minimal tight set that is not a singleton. Here, $t(G) \leq t(G') \leq t(G)+t(G) = 2t(G)$. By Lemma 8 and the previous argument, $m(G') \geq \left\lceil \frac{t(G')}{2} \right\rceil \geq \left\lceil \frac{t(G)}{2} \right\rceil = m(G) \geq 20k^3$. It follows that $m(G') = \left\lceil \frac{t(G')}{2} \right\rceil$. Considering parity of degree sum,

$$(k-1)t(G') + k(n - t(G')) \equiv kn - t(G') \equiv t(G') \equiv 0 \pmod 2.$$

We use the fact that all tight sets are singletons and kn is even. The $m(G') = \frac{t(G')}{2}$ edges form pairings of singleton tight sets of G'. Hence, the resulting graph is a k-connected graph with maximum degree k. The number of modified edges in this process is at most $t(G)+\frac{t(G')}{2} \leq t(G)+\frac{2t(G)}{2} = 2t(G)$. Thus $M(G) \leq 2t(G)$.

2. If $m(G) < 20k^3$: Since $n \geq 120k^3$, $\frac{(k-2)n+2}{2k-3} \geq 40k^3$ holds. Thus by Lemma 9, removing at most $20k^3$ edges from G, we have a $(k-1)$-connected graph G'' having at least $40k^3$ vertices with degree $k-1$. That is, $m(G'') \geq 20k^3$ and $t(G'') \leq t(G) + 40k^3$. Note that this edge removal preserves $(k-1)$-connectivity of G. Using an argument similar to the previous case, $M(G) \leq 20k^3 + 2t(G'') \leq 100k^3 + 2t(G)$. Thus, $M(G) \leq \max\{200k^3, 4t(G)\}$.

Summarizing the two cases, the theorem follows. □

To simplify the problem, we combine Theorems 7 and 10. That a graph G is ϵ-far from k-connectivity means $\frac{\epsilon dn}{2} \leq M(G)$. Thus, we establish the next one.

Theorem 11. *Let G be a $(k-1)$-connected graph which is ϵ-far from k-connectivity with $n > \max\{120k^3, \frac{400k^3}{\epsilon d}\}$. Then, $\frac{\epsilon dn}{26} \leq t(G)$.* □

From this theorem we can easily show the next corollary.

Corollary 12. *Let G be a $(k-1)$-connected graph that is ϵ-far from k-connectivity with $n > \max\{120k^3, \frac{400k^3}{\epsilon d}\}$. Then, there exist at least $\frac{\epsilon dn}{52}$ disjoint minimal tight sets each containing at most $\frac{52}{\epsilon d}$ vertices.*

Proof. Suppose that there exist less than $\frac{\epsilon dn}{52}$ disjoint minimal tight sets with at most $\frac{52}{\epsilon d}$ vertices. Then, by Theorem 11, there exist more than $\frac{\epsilon dn}{52}$ disjoint minimal tight sets with more than $\frac{52}{\epsilon d}$ vertices. It follows that the number of vertices is more than n, a contradiction. □

3.2 An Algorithm for Testing k-Vertex-Connectivity of $(k-1)$-Vertex-Connected Graphs

In this subsection, we first present an algorithm for identifying a tight set T from a given vertex $s \in T$, and next describe the entire algorithm for testing k-connectivity given $(k-1)$-connected graphs.

```
Algorithm ExhaustSearch(s,k,u,G)
Input
    s: starting vertex
    k: the number of vertices adjacent to a tight set
    u: upper bound on size of tight sets.
    G: graph
Output
    true: if s is contained in some tight set with a size at most u
    false: otherwise
begin
    Perform BFS from s until (u + 1) vertices have been reached,
    and Let X be the set of reached vertices.
    if |X| < u + 1
        return true
    else if  k = 0
        return false
    else
        for each v ∈ X
            if ExhaustSearch(s, k − 1, u, G − v)
                return true
            end if
        end for
        return false
    end if
end
```

Fig. 1. Tight Set Identification Algorithm

Suppose that a vertex $s \in T$ and the upper bound u of the size of T is given. We want to identify T and $\Gamma(T)$. Since $|T| \leq u$, BFS from s until $u+1$ vertices have been reached out of $|T|$. Let X be the vertices found in this BFS. Then, X must have at least one vertex in $\Gamma(T)$. We certainly don't know which vertex is in $\Gamma(T)$. But considering u is a fixed number, we can adopt an exhaustive search as follows. We remove one vertex from X. If it is in $\Gamma(T)$ luckily, $|\Gamma(T)|$ decreases by one. To find another vertex of the rest of $\Gamma(T)$, we start a BFS again from s until $u+1$ vertices have been reached. This process is repeated $k-1$ times. If all removed $k-1$ vertices are luckily in $\Gamma(T)$, BFS from s cannot reach $u+1$ vertices because T is completely separated from $V-T$. Otherwise, BFS from s can still reach $u+1$ vertices. In this case, we change the candidates of $\Gamma(T)$ and repeat the above process for all possible combinations. Figure 1 describes this algorithm.

Clearly, the complexity of this algorithm is $O(du^k)$. Note that this algorithm always outputs false if the graph is k-connected.

Lemma 13. *Let G be a $(k-1)$-connected graph with more than u vertices. If G is k-connected, algorithm ExhaustSearch always outputs false. If G is not*

```
Algorithm ConnectivityTest
Output
    accept: if G is k-connected
    reject: otherwise
begin
    if n ≤ max{120k^3, 400k^3/εd}
        check connectivity of G using usual polynomial time algorithms.
    else
        l = ⌈log_2 52/εd⌉
        for i = 1 to l
            let X be a set of uniformly chosen ν_i = 208l/2^i εd vertices.
            for each v ∈ X
                if ExhaustSearch(v, k − 1, 2^i, G)
                    return reject
                end if
            end for
        end for
        return accept
    end if
end
```

Fig. 2. Testing k-Connectivity Algorithm

k-connected and s is a vertex in a tight set T, then ExhaustSearch always outputs true if $|T| \leq u$ and outputs false if $|T| > u$. □

We can construct a testing algorithm by using the above algorithm as follows. First, we randomly select $\Theta(\frac{1}{\epsilon d})$ vertices. If the graph is ϵ-far from k-connectivity, by Corollary 12, with high probability one of these vertices is contained in a tight set of a size at most $\frac{52}{\epsilon d}$. Hence, we can reject the graph using ExhaustSearch. If the graph is k-connected, we accept it with probability one because the tight set identification algorithm never outputs true in this case.

To make the algorithm run faster, we treat tight sets separately by their size. Since there are more than $\frac{52}{\epsilon d}$ tight sets with a size at most $\frac{52}{\epsilon d}$, there exists $i \leq l = \lceil \log_2 \frac{52}{\epsilon d} \rceil$ such that G has at least $\frac{52}{\epsilon d l}$ tight sets with size ranging between 2^{i-1} and $2^i - 1$.

The details are described in Fig. 2.

Theorem 14. *Let G be a $(k-1)$-connected graph. Algorithm ConnectivityTest always accepts if G is k-connected, and rejects with a probability of at least $\frac{2}{3}$ if G is ϵ-far from k-connectivity. The running time is $O\left(d\left(\frac{c}{\epsilon d}\right)^k \log \frac{1}{\epsilon d}\right)$ ($c > 1$ is a constant).*

Proof. If $n \leq \max\{120k^3, \frac{400k^3}{\epsilon d}\}$, n can be treated as a constant, and we are done.

If G is k-connected, algorithm ExhaustSearch always return false. Thus, the algorithm always outputs accept.

Suppose $n > \max\{120k^3, \frac{400k^3}{\epsilon d}\}$ and G is ϵ-far from k-connectivity. Let T_i be a set of disjoint minimal tight sets of a size ranging between 2^{i-1} and $2^i - 1$. Let $l = \lceil \log_2 \frac{52}{\epsilon d} \rceil$. By Corollary 12, $\sum_{i=1}^{l} |T_i| \geq \frac{\epsilon dn}{52}$. Hence, there exists an $i \leq l$ so that $|T_i| \geq \frac{\epsilon dn}{52l}$. Since the number of vertices residing in tight sets belonging to T_i is at least $2^{i-1}|T_i|$, the probability that a uniformly chosen vertex belongs to T_i is at least

$$\frac{2^{i-1}|T_i|}{n} \geq \frac{2^i \epsilon d}{104l}.$$

Thus, one of the uniformly chosen $\nu_i = \frac{104l}{2^{i-1}\epsilon d}$ vertices belongs to T_i with a probability of at least $1 - \left(1 - \frac{2}{\nu_i}\right)^{\nu_i} > 1 - e^{-2} > \frac{2}{3}$. If s is such a vertex, ExhaustSearch always returns true, and the graph will be rejected.

Overall running time is

$$\sum_{i=1}^{l} \nu_i d(2^i)^k = \sum_{i=1}^{l} \frac{208l}{2^i \epsilon d} d 2^{ik} = \frac{208l}{\epsilon} \sum_{i=1}^{l} (2^{k-1})^i$$
$$= O\left(d \left(\frac{52}{\epsilon d}\right)^k \log \frac{1}{\epsilon d}\right). \qquad \square$$

4 Testing k-Vertex-Connectivity of General Graphs

We can construct an algorithm for general graphs by using ConnectivityTest shown in the previous section.

Algorithm ConnectivityTest depends only on the number of disjoint deficient sets in graphs ϵ-far from k-connectivity. Thus, if enough deficient sets exist in a general graph that is ϵ-far from k-connectivity, a similar algorithm works.

First, we show that a property similar to Theorem 3 holds even if the maximum degree of graphs is k (i.e., $d = k$).

Theorem 15. *Let G be a graph with $n \geq 120k^3$ and maximum degree k. Then $M(G) \leq \max\{200k^3, 77m(G)\}$.*

Proof. Suppose that $M(G) > 200k^3$. We have a $(k-1)$-connected graph with maximum degree k by adding $M_{k-1}^k(G)$ edges to G. Note that $M_{k-1}^k(G) \leq 13m_{k-1}(G) \leq 13m_k(G)$ by Theorem 3. Let G' be the resulting graph. In this process, at most $8m_k(G)$ edges have been removed from the k-connected graph. Hence G' has at most $16m_k(G)$ disjoint minimal tight sets. By Theorem 10, $M_k^k(G') \leq 4t(G') \leq 64m_k(G)$. We have a k-connected graph with maximum degree k modifying at most $13m(G)+64m(G) = 77m(G)$ edges, and the theorem follows. $\square$

Combining Theorems 3 and 15, we have the next corollary easily.

Corollary 16. *Let G be a graph with $n \geq 120k^3$. Then $M(G) \leq \max\{200k^3, 77m(G)\}$.* □

Let G be a graph that is ϵ-far from k-connectivity with $n > \max\{120k^3, \frac{400k^3}{\epsilon d}\}$. Since $\frac{\epsilon dn}{2} \leq M(G)$, by the above corollary, $\frac{\epsilon dn}{154} \leq m(G)$. Let m_i be the minimum number of edges to be added for making G i-connected, and let G_i be the i-connected graph that results from adding such m_i edges to G. We define $m_0 = 0$ and $G_0 = G$. Since $m_k \geq \frac{\epsilon dn}{154}$, some i exists so that $m_i - m_{i-1} \geq \frac{\epsilon dn}{154k}$. In order to transform G_{i-1} to be i-connected, we must add at least $\frac{\epsilon dn}{154k}$ edges. Thus, by Lemma 6, $\frac{\epsilon dn}{154k} < k$ or $t(G_{i-1}) > \frac{\epsilon dn}{154k}$. In the former case, $n < \frac{154k^2}{\epsilon d}$ and it contradicts the assumption. Hence, $t(G_{i-1}) > \frac{\epsilon dn}{154k}$. Let $\epsilon' = \frac{13\epsilon}{77k}$, and apply ConnectivityTest on the G_{i-1} with parameter ϵ' replacing ϵ, then it would detect that G_{i-1} is not $k(> i)$ connected with a probability of at least $\frac{2}{3}$.

We next show that the detection probability of the testing algorithm directly applied on G is high. Note that ConnectivityTest only depends on the number of deficient fragments in a given graph that is ϵ-far from k-connectivity. In particular, the higher the number of deficient sets in the graph, the higher the probability that the algorithm decides the graph is not k-connected. Since G is a subgraph of G_{i-1}, the number of deficient sets in G is greater than or equal to those in G_{i-1}. Thus, just applying ConnectivityTest on G with parameter ϵ' decides that G is k-connected or not with a probability of at least $\frac{2}{3}$. By summarizing the above discussions, we establish the next theorem.

Theorem 17. *Let $\epsilon' = \frac{13\epsilon}{77k}$ and G be a graph. If G is k-connected, ConnectivityTest with parameter ϵ' replacing ϵ always accepts. If G is ϵ-far from k-connectivity, the algorithm rejects with a probability of at least $\frac{2}{3}$. The running time of the algorithm is $O\left(d\left(\frac{ck}{\epsilon d}\right)^k \log \frac{k}{\epsilon d}\right)$ ($c > 1$ is a constant).* □

5 Conclusions

We presented an algorithm for testing the k-connectivity of graphs with running time $O\left(d\left(\frac{c}{\epsilon d}\right)^k \log \frac{1}{\epsilon d}\right)$ for $(k-1)$-connected graphs and $O\left(d\left(\frac{ck}{\epsilon d}\right)^k \log \frac{k}{\epsilon d}\right)$ for general graphs. This is the first constant-time k-connectivity testing algorithm for general $k \geq 4$.

By employing a similar method, we have developed an algorithm for testing the k-edge-connectivity of digraphs. Note that only undirected graphs have been solved even for *edge*-connectivity. The running time is the same as the k-connectivity testing algorithm. The details will be reported in another article. So, the remaining problem is testing the k-vertex-connectivity of digraphs.

References

1. Bienstock, D., Brickell, E.F., Monma, C.L.: On the structure of minimum-weight k-connected spanning networks. SIAM J. Discret. Math. 3(3), 320–329 (1990)
2. Bollobas, B.: Extremal Graph Theory. Dover Publications (2004) (incorporated)

3. Cheriyan, J., Jordán, T., Nutov, Z.: On rooted node-connectivity problems. Algorithmica 30(3), 353–375 (2001)
4. Czumaj, A., Sohler, C.: Testing expansion in bounded-degree graphs. In: FOCS 2007: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 570–578 (2007)
5. Fischer, E.: The art of uninformed decisions: A primer to property testing. BEATCS: Bulletin of the European Association for Theoretical Computer Science 75 (2001)
6. Goldreich, O.: Combinatorial property testing (a survey) (1998)
7. Goldreich, O., Ron, D.: Property testing in bounded degree graphs. In: STOC 1997: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pp. 406–415. ACM, New York (1997)
8. Goldreich, O., Ron, D.: A sublinear bipartiteness tester for bounded degree graphs. In: STOC 1998: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pp. 289–298. ACM, New York (1998)
9. Goldreich, O., Ron, D.: On testing expansion in bounded-degree graphs. Electronic Colloquium on Computational Complexity (ECCC) (020) (2000)
10. Jackson, B., Jordán, T.: A near optimal algorithm for vertex connectivity augmentation. In: Lee, D.T., Teng, S.-H. (eds.) ISAAC 2000. LNCS, vol. 1969, pp. 312–325. Springer, Heidelberg (2000)
11. Jackson, B., Jordán, T.: Independence free graphs and vertex connectivity augmentation. J. Comb. Theory Ser. B 94(1), 31–77 (2005)
12. Jordán, T.: On the optimal vertex-connectivity augmentation. J. Comb. Theory Ser. B 63(1), 8–20 (1995)
13. Jordán, T.: A note on the vertex-connectivity augmentation problem. J. Comb. Theory Ser. B 71(2), 294–301 (1997)
14. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM J. Comput. 25(2), 252–271 (1996)

Almost 2-SAT Is Fixed-Parameter Tractable (Extended Abstract)

Igor Razgon and Barry O'Sullivan

Cork Constraint Computation Centre
Computer Science Department, University College Cork, Ireland
{i.razgon,b.osullivan}@cs.ucc.ie

Abstract. We consider the following problem. Given a 2-CNF formula, is it possible to remove at most k clauses so that the resulting 2-CNF formula is satisfiable? This problem is known to different research communities in theoretical computer science under the names Almost 2-SAT, All-but-k 2-SAT, 2-CNF deletion, and 2-SAT deletion. The status of the fixed-parameter tractability of this problem is a long-standing open question in the area of parameterized complexity. We resolve this open question by proposing an algorithm that solves this problem in $O(15^k * k * m^3)$ time showing that this problem is fixed-parameter tractable.

1 Introduction

We consider the following problem. Given a 2-CNF formula, is it possible to remove at most k clauses so that the resulting 2-CNF formula is satisfiable? This problem is known to different research communities in theoretical computer science under the names Almost 2-SAT, All-but-k 2-SAT, 2-CNF deletion, and 2-SAT deletion. The status of the fixed-parameter tractability of this problem is a long-standing open question in the area of parameterized complexity. The question regarding the fixed-parameter tractability of this problem was first raised in 1997 by Mahajan and Raman [11,12]. This question has been posed in the book of Niedermeier [15], being referred as one of central challenges for parameterized algorithms design. Finally, in July 2007, this question was included by Fellows in the list of open problems of the Dagstuhl seminar on Parameterized Complexity [5]. In this paper we resolve this open question by proposing an algorithm that solves this problem in $O(15^k * k * m^3)$ time. Thus we show that this problem is fixed-parameter tractable (FPT).

Regarding the name of this problem, we call *Almost 2-SAT* (2-ASAT) the optimization problem whose output is the smallest subset of clauses that have to be removed from the given 2-CNF formula so that the resulting formula is satisfiable. The *parameterized* 2-ASAT problem gets as additional input a parameter k, and the corresponding decision problem is to determine whether at most k clauses can be removed so that the resulting formula becomes satisfiable. The algorithm proposed in this paper solves the parameterized 2-ASAT problem.

Overview of the Algorithm. We define a variation of the 2-ASAT problem called the *Annotated* 2-ASAT *problem with a single literal* (2-ASLASAT). The

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 551–562, 2008.

input of this problem is a triple (F, L, l), where F is a 2-CNF formula, L is a set of literals such that F is *satisfiable with respect to* L (i.e. $F \wedge \bigwedge_{l' \in L} l'$ is satisfiable), l is a single literal. The task is to find a smallest subset of clauses of F such that after their removal the resulting formula is satisfiable with respect to $(L \cup \{l\})$. The description of the algorithm for the parameterized 2-ASAT problem is divided into two parts. In the first, and most important part we provide an $O(5^k * k * m^2)$ time algorithm that solves the parameterized 2-ASLASAT problem, where the parameter k is the maximum number of clauses to be removed and m is the number of clauses of F. In the second part we show that the parameterized 2-ASAT problem can be solved by $O(3^k * m)$ applications of the algorithm solving the parameterized 2-ASLASAT problem. The resulting runtime follows from the product of the last two complexity expressions. The transformation of the 2-ASAT problem into the 2-ASLASAT problem is based on *iterative compression* and can be seen as an adaptation of the method employed in [8] in order to solve the graph bipartization problem. In the following we overview the first part.

We introduce a *polynomially computable lower bound* on the solution size of the 2-ASLASAT problem for input (F, L, l). Then we prove that if a literal l^* is *neutral*, i.e. the lower bound on the solution size for $(F, L \cup \{l^*\}, l)$ is the same as for (F, L, l), then the solution size for $(F, L \cup \{l^*\}, l)$ and (F, L, l) is the same. This theorem allows us to introduce an algorithm that selects a clause C of F and applies the following branching rule. If C includes a neutral literal l^* then the algorithm applies itself recursively to $(F, L \cup \{l^*\}, l)$ *without any branching*. If not, the algorithm produces at most three branches. On one of them it removes C from F and decreases the parameter. On each of the other branches the algorithm adds one of the literals of C to L and applies itself recursively without changing the size of the parameter. The search tree produced by the algorithm is bounded because on each branch either the parameter is decreased or the lower bound on the solution size is increased (because the literals of the selected clause are *not neutral*). Thus on each branch *the gap between the parameter and the lower bound of the solution size is decreased* which ensures that the size of the search tree exponentially depends only on k and not on the size of F.

The lower bound mentioned in the previous paragraph is obtained by representing the 2-ASLASAT as a *separation problem*. In particular, we define the notion of a walk of a 2-CNF formula and show that, given an instance (F, L, l) of the 2-ASLASAT problem, F is insatisfiable with respect to $L \cup \{l\}$ if and only if there is a walk from $\neg L$ (i.e. from the set of negations of the literals of L) to $\neg l$ or a walk from $\neg l$ to $\neg l$. Thus the 2-ASLASAT problem can be viewed as a problem of finding the smallest set of clauses whose removal breaks all these walks. The considered lower bound on the solution size is the smallest number of clauses separating $\neg L$ from $\neg l$. We show that the size of this *separator* equals the largest number of clause-disjoint *paths* (i.e. walks without repeated clauses) from $\neg L$ to $\neg l$ and that it can be computed in a polynomial time by a Ford-Fulkerson-like procedure. For this proof it is essential that F is satisfiable with respect to L.

Related Work. As said above, the parameterized 2-ASAT problem has been introduced in [11]. In [10], this problem was shown to be a generalization of the

parameterized graph bipartization problem, which was also an open problem at that time. The latter problem was resolved in [16]. The additional contribution of [16] was introducing a method of iterative compression that has had a considerable impact on the design of parameterized algorithms. The most recent algorithms based on this method are currently the best one for the undirected Feedback Vertex Set [1] and the first parameterized algorithm for the famous Directed Feedback Vertex Set problem [3]. For earlier results based on iterative compression, we refer the reader to a survey article [9].

The study of parameterized graph separation problems was initiated in [13]. The technique introduced by the author allowed him to design fixed-parameter algorithms for the multiterminal cut problem and for a more general multicut problem. The latter assumed that the number of pairs of terminals to be separated was also a parameter. The latter result was extended in [7] where fixed-parameter algorithms for multicut problems on several classes of graphs were proposed. The first $O(c^k * poly(n))$ algorithm for the multiterminal cut problem was proposed in [2]. A reformulation of the main theorem of [2] is an essential part of the parameterized algorithm for the Directed FVS problem [3] mentioned in the previous paragraph. In the present paper, we applied the strategy of proof of this theorem in order to show that adding a *neutral* literal to the set of literals of the input does not increase the solution size. Along with computing the separators, the methods of computing disjoint paths have been investigated. The research led to intractability results [17] and parameterized approximability results [6].

The parameterized MAX-SAT problem (a complementary problem to the one considered in the present paper) where the goal is to satisfy at least k clauses of arbitrary sizes also received a considerable attention from the researchers. The best currently known algorithm for this problem runs in $O(1.37^k + |F|)$, where $|F|$ is the size of the given formula [4].

Structure of the Paper. In Section 2 we introduce the terminology which we use in the rest of the paper. In Section 3 we prove that the 2-ASLASAT problem is fixed-parameter tractable. In Section 4 we show that the 2-ASAT problem is fixed-parameter tractable. We conclude by mentioning a number of problems known to be FPT-equivalent to parameterized 2-ASAT and notice that the fixed-parameter tractability of these problems follows as a by-product of our main result. Due to space constraints, proofs are either omitted or replaced by sketches.[1]

2 Terminology

A CNF formula F is called a 2-CNF *formula* if each clause of F is of size at most 2. Throughout the paper we make two assumptions regarding the 2-CNF formulas we consider. Firstly, we assume that all the clauses are of size 2. If a formula has a clause (l) of size 1 then this clause is represented as $(l \vee l)$. Secondly,

[1] A manuscript available at http://arxiv.org/abs/0801.1300 contains a complete description of the result of the present paper with all the proofs and technical details.

everywhere except the very last theorem, we assume that all the clauses of any formula are pairwise distinct, i.e. no two clauses have the same set of literals. This assumption allows us to represent the operation of removing clauses from a formula in a set-theoretical manner. In particular, let S be a set of clauses. Then $F \setminus S$ is a 2-CNF formula that is the conjunction of clauses of F that are not contained in S. The result of removing a single clause C is denoted by $F \setminus C$ rather than $F \setminus \{C\}$.

Let F, S, C, L be a 2-CNF formula, a set of clauses, a single clause, and a set of literals, respectively. Then $Var(F)$, $Var(S)$, $Var(C)$, $Var(L)$ denote the set of variables whose literals appear in F, S, C, and L, respectively. For a single literal l, we denote by $Var(l)$ the variable of l. Also we denote by $Clauses(F)$ the set of clauses of F.

A set of literals L is called *non-contradictory* if it does not contain a literal and its negation. A literal l *satisfies* a clause $(l_1 \vee l_2)$ if $l = l_1$ or $l = l_2$. Given a 2-CNF formula F, a non-contradictory set of literals L such that $Var(F) = Var(L)$ and each clause of F is satisfied by at least one literal of L, we call L a *satisfying assignment* of F. F is *satisfiable* if it has at least one satisfying assignment. Given a set of literals L, we denote by $\neg L$ the set consisting of negations of all the literals of L. For example, if $L = \{l_1, l_2, \neg l_3\}$ then $\neg L = \{\neg l_1, \neg l_2, l_3\}$.

Let F be a 2-CNF formula and L be a set of literals. F is *satisfiable with respect to* L if $F \wedge \bigwedge_{l' \in L} l'$ is satisfiable. The notion of satisfiability of a 2-CNF formula with respect to the given set of literals will be very frequently used in the paper, hence, in order to save the space, we introduce a special notation for this notion. In particular, we say that $\textsc{swrt}(F, L)$ is true (false) if F is, respectively, satisfiable (not satisfiable) with respect to L. If L consists of a single literal l then we write $\textsc{swrt}(F, l)$ rather than $\textsc{swrt}(F, \{l\})$.

Definition 1 (Walk of a 2-CNF). *A* walk *of the given* 2-CNF *formula F is a non-empty sequence $w = (C_1, \ldots, C_q)$ of (not necessarily distinct) clauses of F having the following property. For each C_i one of its literals is specified as the* first literal *of C_i, the other literal is the* second literal*, and for any two consecutive clauses C_i and C_{i+1} the second literal of C_i is the negation of the first literal of C_{i+1}. The walk w is a* path *if all its clauses are pairwise distinct.*

Let $w = (C_1, \ldots, C_q)$ be a walk and let l' and l'' be the first literal of C_1 and the second literal of C_q, respectively. Then we say that l' is *the first literal of* w, that l'' is *the last literal of* w, and that w is *a walk from l' to l''*. Let L be a set of literals such that $l' \in L$. Then we say that w is a walk *from* L. Let $C = (l_1 \vee l_2)$ be a clause of w. Then l_1 is a first literal of C with respect to w if l_1 is the first literal of some C_i such that $C = C_i$. A second literal of a clause with respect to a walk is defined accordingly. In general a literal of a clause may be both a first and a second with respect to the given walk.

Definition 2 (Culprit Sets, 2-ASAT and 2-ASLASAT Problems)

- *A Culprit Set (CS) of a* 2-CNF *formula F is a subset S of $Clauses(F)$ such that $F \setminus S$ is satisfiable. We call the problem of finding a* Smallest CS *(SCS) of F the Almost 2-SAT Problem (*2-ASAT *problem).*

– *Let* (F, L, l) *be a triple where* F *is a* 2-CNF *formula,* L *is a non-contradictory set of literals such that* $\text{SWRT}(F, L)$ *is true and* l *is a literal such that* $Var(l) \notin Var(L)$*. A CS of* (F, L, l) *is a subset* S *of* $Clauses(F)$ *such that* $\text{SWRT}(F \setminus S, L \cup \{l\})$ *is true. We call the problem of finding a SCS of* (F, L, l) *the Annotated Almost 2-SAT problem with single literal (*2-ASLASAT *problem).*

In this paper we consider the parameterized versions of the 2-ASAT and 2-ASLASAT problems. In particular, the input of the *parameterized* 2-ASAT problem is (F, k), where F is a 2-CNF formula and k is a non-negative integer. The output is a CS of F of size at most k, if one exists. Otherwise, the output is 'NO'. The input of the *parameterized* 2-ASLASAT problem is (F, L, l, k) where (F, L, l) is as specified in Definition 3. The output is a CS of (F, L, l) of size at most k, if one exists. Otherwise, the output is 'NO'.

3 Parameterized Algorithm for the 2-ASLASAT Problem

We begin our analysis from the following Theorem.

Theorem 1. *Given a* 2-ASLASAT *problem instance* (F, L, l)*, then* $\text{SWRT}(F, L \cup \{l\})$ *is false if and only if* F *has a walk from* $\neg l$ *to* $\neg l$ *or a walk from* $\neg L$ *to* $\neg l$*.*

Theorem 1 allows us to view the 2-ASLASAT problem as a *separation* problem. In particular, a SCS of (F, L, l) can be viewed as the smallest number of clauses whose removal separates all walks from $\neg L$ to $\neg l$ and from $\neg l$ to $\neg l$. Our next step is to introduce a polynomially computable lower bound for the size of a SCS of (F, L, l).

Consider a smallest subset S of clauses such that $F \setminus S$ has no path from $\neg L$ to $\neg l$. We denote $|S|$ by $SepSize(F, \neg L, \neg l)$. From Theorem 1, $SepSize(F, \neg L, \neg l)$ is a lower bound on the size of an SCS of (F, L, l). In order to prove polynomial computability of $SepSize(F, \neg L, \neg l)$, we recall a well known notion of the *implication graph* of F. This is a digraph D whose set $V(D)$ of vertices corresponds to the set of literals of the variables of F and (l_1, l_2) is an arc in its set $A(D)$ of arcs if and only if $(\neg l_1 \vee l_2) \in Clauses(F)$. It is easy to establish a one-to-one correspondence between the walks of F and the walks of D. In particular, a walk $w = (l_1 \vee \neg l_2), (l_2 \vee \neg l_3), \ldots, (l_{t-1} \vee \neg l_t)$ of F from l_1 to $\neg l_t$ corresponds to the walk $w(D) = (\neg l_1, \neg l_2), (\neg l_2, \neg l_3), \ldots, (\neg l_{t-1}, \neg l_t)$ in D from $\neg l_1$ to $\neg l_t$. The opposite correspondence holds as well. Observe that in the above walk each clause $C_i = (l_i \vee \neg l_{i+1})$ is *represented* by arc $e_i = (\neg l_i, \neg l_{i+1})$ and the first and the second literals of C_i with respect to w correspond to the tail and the head of e_i, respectively.

This correspondence suggests that a Menger-like dependence in F might hold, i.e. the number of clause-disjoint paths from $\neg L$ to $\neg l$ equals $SepSize(F, \neg L, \neg l)$, and that $SepSize(F, \neg L, \neg l)$ might be computed by a Ford-Fulkerson-like procedure. This statement would immediately follow if one established a one-to-one correspondence between the sets of clause-disjoint paths of F and the sets of clause-disjoint paths of D. The subtle point is that this correspondence does not

hold in general. The reason is that a clause $(l_1 \vee l_2)$ where l_1 and l_2 are distinct is represented by two arcs of D: $(\neg l_1, l_2)$ and $(\neg l_2, l_1)$. It follows that a path of D may correspond to a walk of F which is not a path (i.e. has repeated clauses). Moreover, a set of arc-disjoint paths of D may correspond to a set of walks of F which are not clause-disjoint.

Fortunately, it is sufficient for us to establish the correspondence only between the paths from $\neg L$ in F and the paths from L in D. Taking into account that SWRT(F, L) is true, by definition of the 2-ASLASAT problem, this correspondence can be shown based on the following lemma.

Lemma 1. *Let F be a* 2-CNF *formula and let L be a set of literals such that* SWRT(F, L) *is true. Let $C = (l_1 \vee l_2)$ be a clause of F and let w be a walk of F from $\neg L$ containing C and assume that l_1 is a first literal of C with respect to w. Then l_1 is* not *a second literal of C with respect to any walk from $\neg L$.*

It immediately follows from this lemma that there are no two paths p_1 and p_2 of D starting at L such that $(\neg l_1, l_2)$ participates in p_1, while $(\neg l_2, l_1)$ participates in p_2 because it would mean that in the corresponding walks in F, l_1 is the first literal of C with respect to one of them and the second literal with respect to the other. Thus Lemma 1 eliminates the above obstacle to establishing the desired correspondence. Formalizing this argument allows us to prove the following theorem.

Theorem 2. *Given a* 2-ASLASAT *problem instance (F, L, l), $SepSize(F, \neg L, \neg l)$ equals the largest number of clause-disjoint paths from $\neg L$ to $\neg l$ in F as well as the largest number of arc-disjoint paths from L to $\neg l$ in D.*

Based on Theorems 1 and 2, we can give an informal outline of a parameterized algorithm for the 2-ASLASAT problem. For the main case of this algorithm we have an instance (F, L, l) of the problem where L is non-empty and select a clause $C = (l_1 \vee l_2)$ such that $l_1 \in \neg L$. We branch on the removal/non-removal of this clause. If this clause is not removed then any satisfying assignment with respect to L must contain $\neg l_1$ and, hence, also must contain l_2 in order to satisfy C. Therefore, we can say that we branch on the removal of this clause, or the addition of l_2 to L. The first branch decreases the parameter but the second branch does not. Hence the second branch is problematic for the design of the parameterized algorithm. One fortuitous case occurs if $SepSize(F, \neg(L \cup \{l_2\}), \neg l) > SepSize(F, \neg L, \neg l)$. According to the combination of Theorem 1 and Theorem 2, this condition means that adding l_2 to L increases a polynomially computable lower bound on the size of a SCS of (F, L, l). Therefore if C satisfies this fortuitious case then both branches *decrease* the gap between parameter and the lower bound: one by decreasing the parameter, the other by increasing the lower bound. It can be shown that if only such fortuitious clauses are selected then the algorithm terminates in $O^*(c^k)$. However what about the case where adding l_2 to L does not increase the lower bound? The following Theorem proves that in this case the size of a SCS of (F, L, l) is *not increased* as well and, hence,

the non-removal decision can be safely made regarding C. That is, the branching is applied only for the fortuitious case, which leads to a parameterized algorithm.

Definition 3 (Neutral Literal). *Let (F, L, l) be an instance of the* 2-ASLASAT *problem. A literal l^* is a* neutral literal *of (F, L, l) if $(F, L \cup \{l^*\}, l)$ is an instance of the* 2-ASLASAT *problem and $SepSize(F, \neg L, \neg l) = SepSize(F, \neg(L \cup \{l^*\}), \neg l)$.*

Theorem 3. *Let (F, L, l) be an instance of the 2-ASALSAT problem and let l^* be a neutral literal of (F, L, l). Then there is a CS of $(F, L \cup \{l^*\}, l)$ of size smaller than or equal to the size of an SCS of (F, L, l).*

Proof. (Sketch) We begin by introducing a number of sets. Let X be a SCS of (F, L, l), SP be a set of clauses of F such that $F \setminus SP$ has no path from $\neg(L \cup \{l^*\})$ to $\neg l$ and $|SP| = SepSize(F\neg(L \cup \{l^*\}), \neg l)$ (due to the neutrality of l^*, $|SP| = SepSize(F, \neg L, \neg l)$). Let R be the subset of all clauses C of $F \setminus SP$ such that $F \setminus SP$ has a walk $w_R(C)$ from $\neg L$ ending by C. We denote the rest of the clauses of $F \setminus SP$ by NR. We denote $X \cap R$ by XR. Finally, we denote by Y the set of all clauses C of $SP \setminus X$ such that there is a walk $w(C)$ having the following properties. The first clause of $w(C)$ is C, the last literal of $w(C)$ is $\neg l$, all the literals of $w(C)$ except C belong to $NR \setminus X$, and there is a walk from $\neg(L \cup \{l^*\})$ to $\neg l$ having $w(C)$ as a suffix. We are going to prove that $X^* = (X \setminus XR) \cup Y$ is a CS of $(F, L \cup \{l^*\}, l)$ and that $|Y| \leq |XR|$. The present theorem immediately follows from the combination of these two facts.

By Theorem 1, to prove that X^* is a CS of $(F, L \cup \{l^*\}, l)$ all we need to show is that $F \setminus X^*$ has no walk from $\neg(L \cup \{l^*\})$ to $\neg l$ and from $\neg l$ to $\neg l$. We show here only the former. Assume that $F \setminus X^*$ has a walk w^* from $\neg(L \cup \{l^*\})$ to $\neg l$ in contradiction with the considered statement. Then w^* intersects with SP by definition. Fix the *last* clause C of w^* such that $C \in SP$. We claim that $C \in Y$, which leads to a contradiction with $Y \subseteq X^*$ confirming the statement. To show this we assume the opposite and fix an entry $C' \in R$ of w^* following C. According to Lemma 1, C' has *the same* 'orientation' in both w^* and $w_R(C')$, hence we may replace the prefix of w^* ending with C' by $w_R(C')$. As a result we get a new walk w'' from $\neg L$ to $\neg l$ in $F \setminus X^*$ which meets SP *after* C'. That is, w^* meets SP after C' and hence after C in contradiction to the definition of C. This shows that $C \in Y$.

To show that $|Y| \leq |XR|$, we take a set $\mathbf{P}$ of $|SP|$ clause-disjoint paths of F from $\neg L$ to $\neg l$ guaranteed to exist according to Theorem 2. We observe that every path of $\mathbf{P}$ contains exactly one clause of SP and every clause of SP is contained in exactly one path of $\mathbf{P}$. Let $p \in \mathbf{P}$ be the path containing a clause $C \in Y$. We claim that C is preceded in C by a clause of XR. Otherwise, applying Lemma 1, we can replace the suffix of p starting from C by $w(C)$. As a result we get a walk from $\neg L$ to $\neg l$ that does not intersect with X, i.e. a walk from $\neg L$ to $\neg l$ in $F \setminus X$ which is impossible according to Theorem 1. Since $\mathbf{P}$ has $|Y|$ *clause-disjoint* paths containing the clauses of Y, $|XR| \geq |Y|$ as required. □

We provide the formal description of the algorithm below.

$FindCS(F, L, l, k)$
Input: An instance (F, L, l, k) of the parameterized 2-ASLASAT problem.
Output: A CS of (F, L, l) of size at most k if one exists. Otherwise 'NO' is returned.

1. **if** SWRT$(F, L \cup \{l\})$ is true **then** return $\emptyset$
2. **if** $k = 0$ **then** Return 'NO'
3. **if** $k \geq |Clauses(F)|$ **then** return $Clauses(F)$
4. **if** $SepSize(F, \neg L, \neg l) > k$ **then** return 'NO'
5. **if** F has a walk from $\neg L$ to $\neg l$ **then**
   Let $C = (l_1 \vee l_2)$ be a clause such that $l_1 \in \neg L$ and $Var(l_2) \notin Var(L)$
6. **else** Let $C = (l_1 \vee l_2)$ be a clause which belongs to a walk of F from $\neg l$ to $\neg l$ and SWRT$(F, \{l_1, l_2\})$ is true [2]
7. **if** Both l_1 and l_2 belong to $\neg(L \cup \{l\})$ **then**
   7.1 $S \leftarrow$ FINDCS$(F \setminus C, L, l, k-1)$
   7.2 **if** S is not 'NO' **then** Return $S \cup \{C\}$
   7.3 Return 'NO'
8. **if** Both l_1 and l_2 do not belong to $\neg(L \cup \{l\})$ **then**
   8.1 $S_1 \leftarrow$ FINDCS$(F, L \cup \{l_1\}, l, k)$
   8.2 **if** S_1 is not 'NO' **then** Return S_1
   8.3 $S_2 \leftarrow$ FINDCS$(F, L \cup \{l_2\}, l, k)$
   8.4 **if** S_2 is not 'NO' **then** Return S_2
   8.5 $S_3 \leftarrow$ FINDCS$(F \setminus C, L, l, k-1)$
   8.6 **if** S_3 is not 'NO' **then** Return $S_3 \cup \{C\}$
   8.7 Return 'NO'
   (In the rest of the algorithm we consider the cases where exactly one literal of C belongs to $\neg(L \cup \{l\})$. W.l.o.g. we assume that this literal is l_1)
9. **if** l_2 is not neutral in (F, L, l) **then**
   9.1 $S_2 \leftarrow$ FINDCS$(F, L \cup \{l_2\}, l, k)$
   9.2 **if** S_2 is not 'NO' **then** Return S_2
   9.3 $S_3 \leftarrow$ FINDCS$(F \setminus C, L, l, k-1)$
   9.4 **if** S_3 is not 'NO' **then** Return $S_3 \cup \{C\}$
   9.5 Return 'NO'
10. Return FINDCS$(F, L \cup \{l_2\}, l, k)$

The algorithm is presented as a function $FindCS(F, L, l, k)$. The first part of the algorithm (lines 1-4) is processing the stopping conditions. Lines 1-3 are trivial. Line 4 is correct because $SepSize(F\neg L, \neg l)$ is a lower bound on the size of a SCS of (F, L, l) according to Theorem 1. The second part of the algorithm is selecting the clause C to be considered during the branching process. The selection procedure is designed so that it guarantees that if a literal l' is added to the set L then $Var(l') \notin Var(L)$. This ensures that on any path from the root of the search tree to the leaves there may be at most n nodes that add literals to L, where $n = |Vars(F)|$. Taking into account that along a path in a search tree at most k nodes that remove clauses can occur, we derive that the height of the search tree is at most $n + k$.

The remaining part of the algorithm describes the process of applying an appropriate branching rule depending on the literals of clause C. The correctness

[2] Doing the analysis, we will prove that on Steps 5 and 6 F has at least one clause with the required property.

of the branching rules is based on the observation that if C does not belong to the CS S of (F, L, l) being constructed, then any satisfying assignment of $F \setminus S$, including the one that does not intersects with $\neg(L \cup \{l\})$, has to satisfy C. It follows that on the branches where C is not removed, at least one literal of C is added to L. There are two cases where branching is not performed at all. The first case occurs if the condition of line 7 is satisfied: in this case C itself is not satisfiable with respect to $L \cup \{l\}$. Consequently, C belongs to any CS of (F, L, l). The second case occurs in line 10. The correctness of this step follows from Theorem 3 by taking into account that l_2 is a neutral literal with respect to (F, L, l).

The key part of the runtime analysis is to prove that as a result of adding a literal l' of clause C to L on lines 8.1, 8.3, and 9.1, $SepSize(F, \neg(L \cup \{l'\}), \neg l) > SepSize(F, \neg L, \neg l)$. Regarding line 9.1, the algorithm explicitly states that $l' = l_2$ is not neutral and hence the required property follows from Definition 3. Regarding lines 8.1 and 8.3 our proof uses the following intuitive argument. A clause with both literals outside $\neg(L \cup \{l\})$ can be selected only on line 6 i.e in the case where $SepSize(F, \neg L, \neg l) = 0$. The selected clause $(l_1 \vee l_2)$ belongs to a walk w from $\neg l$ to $\neg l$ in F. We also show that $Var(l_1) \neq Var(l)$ and that $Var(l_2) \neq Var(l)$. Then we derive from the combination of these facts that there are subwalks of w that are walks from $\neg l_1$ to $\neg l$ and from $\neg l_2$ to $\neg l$. Consequently, F has a walk from $\neg(L \cup \{l_i\})$ to $\neg l$ for $i = 1, 2$. It can be shown that in this case, the respective paths also exist which implies by Theorem 2 that $SepSize(F\neg(L \cup \{l_i\}), \neg l) > 0 = SepSize(F, \neg L, \neg l)$.

To prove that the exponential part of the runtime of $FindCS$ depends only on k we show that the number of leaves of the search tree depends only on k. In order to do this we introduce a measure on (F, L, l, k) which is bounded from above by a function of k and which is decreased by each branch whenever a branching rule with 2 or 3 branches is applied. A measure $\beta = \beta(F, L, l, k) = 2k - SepSize(F, \neg L, \neg l)$ satisfies these requirements.[3] Indeed, if we add a non-neutral literal to L then the second item increases and hence the whole measure decreases. If we remove a clause from F (thus decreasing k) then the first item decreases by 2 while the second item decreases by at most 1, thus the whole measure decreases again. Taking into account that $\beta \leq 2k$, we obtain that on each path from the root of the search tree to a leaf at most $2k$ nodes with 2 or 3 outgoing branches. Since each node of the search node has at most 3 outgoing branches, the number of leaves of the search tree is at most $3^{2k} = 9^k$. This already implies the fixed-parameter tractability of the 2-ASLASAT problem. Using a more careful assessment we reduce the upper bound on the number of leaves to 5^k. We omit the details here due to the lack of space. As we noticed above, the height of the search tree is at most $n + k$, hence the number of nodes of the search tree is at most $(n + k)5^k$. It is also not hard to show that $(n + k) = O(m)$, where $m = |Clauses(F)|$. Therefore, the number of nodes of the search tree is bounded by $O(5^k m)$.

[3] In fact, we use the measure $max(0, 2k - SepSize(F, \neg L, \neg l))$. We demonstrate our argument on a simplified measure to make it more intuitive.

In the remaining part of the analysis we notice that the heaviest operations performed by $FindCS$ per node of the search tree are checking on line 4 whether the lower bound is exceeded or not and neutrality checking on line 9. According to Theorem 2, these operations can be performed by $O(k)$ iterations of the Ford-Fulkerson algorithm applied to the implication graph of F. The runtime of each iteration is $O(m + |L|)$, where the additional item $|L|$ takes into account the literals whose variables do not belong to $Vars(F)$. Combining this with the bound on the number of nodes obtained in the previous paragraph, we get the following theorem.

Theorem 4. *The* 2-ASLASAT *problem is fixed-parameter tractable. It particular it can be solved in* $O(5^k km(m + |L|)$ *time.*

4 Algorithm for the 2-ASAT Problem

In the final part of our proof of the fixed-parameter tractability of the 2-ASAT problem we show that it can be solved by $O(3^k m)$ calls to a procedure solving the 2-ASLASAT problem. First we get rid of the repeated clauses by associating with each clause $C = (l' \vee l'')$ of the given formula a unique literal l_i and replacing C by a pair of clauses $(l' \vee l_i)$ and $(\neg l_i \vee l'')$. Note that the number of clauses of the resulting formula remains $O(m)$. Thus we may assume that in the given instance (F, k) of the 2-ASAT problem F has no repeated clauses. Next we observe that the 2-ASAT problem can be solved by $O(m)$ calls to a problem with input (F_1, S_1, k), where F_1 is a 2-CNF formula with $Clauses(F_1) \subseteq Clauses(F)$, and S_1 is a CS of F_1 of size $k + 1$. This transformation is known to the parameterized complexity community under the name *iterative compression* [9].

Next we observe that F_1 has a CS of size at most k if and only if there a set $I \subset S_1$ such that there is a subset S_2 of $Clauses(F_1 \setminus I)$ such that $S_2 \cap S_1 = \emptyset$, $|S_2| \leq k - |I|$, and S_2 is a CS of $F_1 \setminus I$. Thus in order to find the required CS of F_1, we explore 2^{k+1} subsets of S_1 and for each subset I we solve the problem with input (F_2, S_2, k_2), where $F_2 = F_1 \setminus I$, $S_2 = S_1 \setminus I$, $k_2 = k - |I|$. The output of the problem is a CS of F_2 having size at most k_2 and disjoint with S_2.

Set $F_3 = F_2 \setminus S_2$ and observe that the set Y is the required CS of F_2 if and only if $|Y| \leq k_2$ and there is a non-contradictory set of literals satisfying $F_3 \setminus Y$ and all the clauses of S_2. The last condition can be reformulated as follows: there is a non-contradictory set L_3 of literals satisfying the clauses of S_2 such that $F_3 \setminus Y$ is satisfiable with respect to L_3. Our next transformation is based on this view. In particular, we explore all possible non-contradictory sets of literals obtained by taking one literal from each clause of S_2 (there may be at most 2^{k_2+1} such combinations) and for each considered set L_3 we solve a problem with input (F_3, L_3, k_2) whose output is a set such that $Y \subseteq Clauses(F_3)$, $|Y| \leq k_2$, and $\text{SWRT}(F_3 \setminus Y, L_3)$ is true. Note that F_3 is satisfiable because $F_3 = F_2 \setminus S_2 = F \setminus S$ while S is a CS of F.

In the last stage, based on the satisfiability of F_3, we guess a satisfying assignment P_3 of F_3. If P_3 does not intersect with $\neg L_3$ we return the empty set. Otherwise we partition L_3 into the subsets L'_3 and L''_3 such that P_3 does not

intersect with L'_3, while $\neg L''_3 \subseteq P_3$. Then we introduce two new literals l^*_1 and l^*_2 and transform F_3 into a 2-CNF F^* by replacing the literals of L'_3 by l^*_1, the literals of $\neg L'_3$ by $\neg l^*_1$, the literals of L''_3 by l^*_2, and the literals of $\neg L''_3$ by $\neg l^*_2$. We observe that the set of literals P^* obtained from P_3 by the analogous replacement is a satisfying assignment of F^* that does not contain $\neg l^*_1$, from which we conclude that SWRT(F^*, l^*_1) is true. It follows that $(F^*, \{l^*_1\}, l^*_2)$ is a valid instance of the 2-ASLASAT problem. Moreover, we show that $(F^*, \{l^*_1\}, l^*_2)$ has a CS Y^* of size at most k_2 if and only if there is a set Y, $|Y| \leq k_2$ such that SWRT$(F_3 \setminus Y, L)$ is true and show a transformation from Y^* to Y. Taking into account that the 2-ASLASAT problem is known to be FPT according to Theorem 4, it follows that the 2-ASAT problem is FPT as well.

It follows from the above description that 2-ASAT problem can be solved by $O(4^k m)$ calls to an algorithm solving the 2-ASLASAT problem. A simple combinatorial argument decreases the upper bound to $O(3^k m)$. Combining this with Theorem 4 we obtain the following theorem.

Theorem 5. *The* 2-ASAT *problem is fixed-parameter tractable. In particular, it can be solved in* $O(15^k km^3)$ *time.*

5 Concluding Remarks

We conclude by noting a number of consequences of our main result. It was noticed in [5] that the parameterized 2-ASAT problem is FPT-equivalent to the following problem: given a graph G having a perfect matching, find whether G has a vertex cover of size at most $n/2 + k$. This problem is called vertex cover problem parameterized above the perfect matching (VC-PM). It is shown [14] that the VC-PM problem is FPT-equivalent to the vertex cover problem parameterized above the size of a maximum matching and that the latter problem is FPT-equivalent to a problem of finding whether at most k vertices can be removed from the given graph so that the size of the minimum vertex cover of the resulting graph equals the size of its maximum matching. It follows from Theorem 5 that all these problems are fixed parameter tractable.

Acknowledgements

We thank Venkatesh Raman for pointing out to several relevant references, Somnath Sikdar for his help in fixing a bug in an earlier version of the archived manuscript, and an anonymous reviewer for suggestions of improvement the presentation of the extended abstract. Finally, we acknowledge the Science Foundation Ireland for supporting our research through grant 05/IN/I886.

References

1. Chen, J., Fomin, F., Liu, Y., Lu, S., Villanger, Y.: Improved algorithms for the feedback vertex set problems. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 422–433. Springer, Heidelberg (2007)

2. Chen, J., Liu, Y., Lu, S.: An improved parameterized algorithm for the minimum node multiway cut problem. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 495–506. Springer, Heidelberg (2007)
3. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. In: STOC 2008 (to appear, 2008)
4. Chen, J., Kanj, I.A.: Improved exact algorithms for max-sat. Discrete Applied Mathematics 142(1-3), 17–27 (2004)
5. Demaine, E., Gutin, G., Marx, D., Stege, U.: Open problems from dagstuhl seminar 07281 (2007), http://drops.dagstuhl.de/opus/volltexte/2007/1254/pdf/07281.SWM.Paper.1254.pdf
6. Grohe, M., Grüber, M.: Parameterized approximability of the disjoint cycle problem. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 363–374. Springer, Heidelberg (2007)
7. Guo, J., Hüffner, F., Kenar, E., Niedermeier, R., Uhlmann, J.: Complexity and exact algorithms for multicut. In: SOFSEM, pp. 303–312 (2006)
8. Hüffner, F.: Algorithm engineering for optimal graph bipartization. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 240–252. Springer, Heidelberg (2005)
9. Hüffner, F., Niedermeier, R., Wernicke, S.: Techniques for practical fixed-parameter algorithms. The Computer Journal 51(1), 7–25 (2008)
10. Khot, S., Raman, V.: Parameterized complexity of finding subgraphs with hereditary properties. Theoretical Computer Science 289(2), 997–1008 (2002)
11. Mahajan, M., Raman, V.: Parametrizing above guaranteed values: Maxsat and maxcut. Electronic Colloquium on Computational Complexity (ECCC) 4(33) (1997)
12. Mahajan, M., Raman, V.: Parameterizing above guaranteed values: Maxsat and maxcut. Journal of Algorithms 31(2), 335–354 (1999)
13. Marx, D.: Parameterized graph separation problems. Theoretical Computer Science 351(3), 394–406 (2006)
14. Mishra, S., Raman, V., Saurabh, S., Sikdar, S., Subramanian, C.: The complexity of finding subgraphs whose matching number equals the vertex cover number. In: ISAAC, pp. 268–279 (2007)
15. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford Lecture Series in Mathematics and Its Applications, vol. 31 (2006)
16. Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Operations Research Letters 32(4), 299–301 (2004)
17. Slivkins, A.: Parameterized tractability of edge-disjoint paths on directed acyclic graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 482–493. Springer, Heidelberg (2003)

On Problems without Polynomial Kernels (Extended Abstract)

Hans L. Bodlaender[1], Rodney G. Downey[2,⋆], Michael R. Fellows[3,⋆⋆], and Danny Hermelin[4,⋆⋆⋆]

[1] Department of Information and Computing Sciences, Utrecht University, 3508 TB Utrecht 80.089 - Netherlands
hansb@cs.uu.nl
[2] School of Mathematics, Statistics and Computer Science, Victoria University of Wellington, Wellington 600 - New Zealand
rod.downey@vuw.ac.nz
[3] The University of Newcastle, Callaghan NSW 2308 - Australia
michael.fellows@newcastle.edu.au
[4] The University of Haifa, Haifa 31905 - Israel
danny@cri.haifa.ac.il

Abstract. Kernelization is a central technique used in parameterized algorithms, and in other approaches for coping with NP-hard problems. In this paper, we introduce a new method which allows us to show that many problems do not have polynomial size kernels under reasonable complexity-theoretic assumptions. These problems include k-Path, k-Cycle, k-Exact Cycle, k-Short Cheap Tour, k-Graph Minor Order Test, k-Cutwidth, k-Search Number, k-Pathwidth, k-Treewidth, k-Branchwidth, and several optimization problems parameterized by treewidth or cliquewidth.

1 Introduction

Parameterized complexity extends classical complexity theory in a way that allows a refined categorization of tractable and intractable computational problems. This is done by a two-dimensional analysis of problems instances – one dimension used as usual for measuring the input-length, and the other used for measuring other structural-properties of the input, *e.g.* its witness size. A problem is considered tractable, if there is an algorithm solving it with any super-polynomial running-time confined strictly to the parameter. As an example, consider the k-Vertex Cover problem: Given a graph G and a parameter $k \in \mathbb{N}$, determine whether G has a vertex cover of size k. When viewed classically, this problem is NP-complete. However, its parameterized variant can be solved

⋆ Research supported by the Marsden Fund of New Zealand.
⋆⋆ Research supported by the Australian Research Council Center of Excellence in Bioinformatics.
⋆⋆⋆ Supported by the Adams Fellowship of the Israel Academy of Sciences and Humanities.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 563–574, 2008.

in $O(2^k n)$ time [12] (see [23] for improvements), which is practical for instances with small parameter values, and in general is far better than the $O(n^{k+1})$ running time of the brute-force algorithm. More generally, a problem is said to be *fixed-parameter tractable* if it has an algorithm running in time $f(k)p(n)$ (FPT-time), where f is any computable function solely in the parameter k, and $p(n)$ is a polynomial in the total input length n [12]. The class of all fixed-parameter tractable problems is denoted by FPT. The first class of *fixed-parameter intractable* problems is W[1], and it is known that if FPT = W[1] then n variable 3-SAT can be solved in $2^{o(n)}$ time [10].

A fundamental and very powerful technique in designing FPT algorithms is *kernelization*. In a nutshell, a kernelization algorithm for a parameterized problem is a *polynomial-time transformation* that transforms any given instance to an equivalent instance of the same problem, with size and parameter bounded by a function of the parameter in the input. Typically this is done using so-called *reduction rules*, which allow the safe reduction of the instance to an equivalent "smaller" instance. In this sense, kernelization can be viewed as polynomial-time preprocessing which has universal applicability, not only in the design of efficient FPT algorithms, but also in the design of approximation and heuristic algorithms [22].

It is clear that any (decidable) language which has a kernelization algorithm is in FPT. Somewhat more surprising, but still very simple to show, is that all problems in FPT have kernelization algorithms [9]. This is seen by considering the two cases $f(k) \geq n$ and $f(k) < n$ separately, where $f(k)$ is the parameter-dependent time-bound of the algorithm solving the given problem. Since every FPT problem has a kernelization algorithm, it is interesting to study problems that are kernelizable in a stricter sense - for example, problems which allow kernelization algorithms that reduce instances to a size which is *polynomially* bounded by the parameter. Such problems are said to have a polynomial kernelization algorithm, or a *polynomial kernel*. For instance, the classical kernelization algorithm of Buss for k-Vertex Cover is a polynomial kernel (see *e.g.* [12]), and so k-Vertex Cover has a polynomial kernel. Other problems known to have polynomial kernels include k-Leaf Spanning Tree [6], k-Feedback Vertex Set [5,8], k-Planar Dominating Set [1], k-Cluster Editing [21], k-Hitting Set for Sets of Bounded Size [28], and many more.

On the other hand, there are also several problems for which no polynomial kernel has yet been found. These clearly include all problems known to be W[1]-hard, as the existence of a kernel for such a problem would imply W[1] = FPT. So we focus on parameterized problems known to be in FPT. Many examples of such problems can be found among the problems shown to be in FPT using heavy machinery such as *color-coding* [2], the *graph minor technique* [14], or *tree-decomposition dynamic programming* [3]. In many cases, the algorithms given by these frameworks are impractical in practice. For instance, consider the k-Path problem: Given a graph G and a parameter $k \in \mathbb{N}$, determine whether G has a simple path of length k.

This problem can be solved in $O(2^{O(k)}n^2 \lg n)$ time using the color-coding technique of Alon, Yuster, and Zwick [2]. This time complexity might seem similar to the complexity of the algorithm for k-Vertex Cover mentioned above, however the hidden constant in the $O(k)$ exponent is quite large, ruling-out any possibility for practical usefulness[1]. Nevertheless, an efficient polynomial kernel could be a promising path in making this algorithm practical. Does k-Path have a polynomial kernel? k-Minor Order Test and k-Treewidth are other good examples, as both serve as highly time-consuming subroutines in most algorithms deploying the graph minor technique or tree-decomposition dynamic programming. Do k-Minor Order Test and k-Treewidth have polynomial kernels?

In this paper, we introduce a new method which allows us to show that many problems do not have polynomial kernels under reasonable complexity-theoretic assumptions. We believe that this material is significant and will have wide applications. For instance, learning of our material, three other teams of authors, namely, Fortnow and Santhanam [19], Chen *et al.* [11], and Buhrman [7] have applied the concepts in this paper to other arenas.

Questions such as these are the motivating starting point of this paper. Clearly, if P = NP then all parameterized problems based on NP-complete problems have constant size kernels. Thus, any method we generate to show that a problem is unlikely to have a polynomial kernel will entail a complexity-theoretic hypothesis. For developing such a hypotheses, we introduce the notion of a *distillation algorithm*. Intuitively speaking, a distillation algorithm for a given problem functions like a Boolean *OR* gate of problem-instances – it receives as input a sequence of instances, and outputs yes-instance iff at least one of the instances in the sequence is also a yes-instance. The algorithm is allowed to run in time polynomial in the total length of the sequence, but must output an instance whose size is polynomially bounded by the size of the maximum-size instance in its input sequence. We remark that independently and somewhat earlier, a similar notion had been formulated by Harnik and Naor [24] in relation to compression-related cryptographic problems. Our paper, as well as the subsequent papers mentioned above, show that the notion of distillation is of central importance in complexity considerations.

We study the possibility of the existence of distillation algorithms for NP-complete problems, and conjecture that this is highly implausible. It is clear that if any NP-complete problem has a distillation algorithm, then they all do. This seems very unlikely. Intuitively, large amounts of information cannot be coalesced into a single small instance. This notion seems rather similar to the notion of P-selectivity which collapses the polynomial hierarchy [26]. This same intuition suggests that the existence of distillation algorithms for NP-complete problems might lead to a similar collapse. After correspondence about this issue, Fortnow and Santhanam verified a conjecture of ours proving that the existence a of distillation algorithm for any NP-complete problem would imply the collapse

[1] It was brought to our attention that there are recent improvements to the k-Path algorithm mentioned above which have rather practical running-times [25,27].

of the polynomial hierarchy to the third level [19]. This allows us to prove, via a parameterized form of distillation, the unlikelihood of polynomial kernels for FPT problems such as k-Path, k-Minor Order Test and others. In particular, our study gives rise to the following theorem.

Theorem 1. *Unless all* NP*-complete problems have distillation algorithms, none of the following* FPT *problems have polynomial kernels:* k-Path, k-Cycle, k-Exact Cycle, k-Short Cheap Tour, k-Graph Minor Order Test, k-Bounded Treewidth Subgraph Test, k-Planar Graph Subgraph Test, k-Planar Graph Induced Subgraph Test, w-Independent Set, w-Clique, *and* w-Dominating Set.

Here, w-Independent Set, w-Clique, and w-Dominating Set denote the classical Independent Set, Clique, and Dominating Set problems parameterized by the treewidth of their given graphs. These are given as mere examples. Many other graph-theoretic problems parameterized by the treewidth of the graph could have been used in the theorem.

We next turn to study distillation of coNP-complete problems. Although we are unable to relate the existence of distillation algorithms for coNP-complete problems to any known complexity conjecture, we can still show that polynomial kernels for some important FPT problems not captured by Theorem 1, imply distillation algorithms for coNP-complete problems.

Theorem 2. *Unless all* coNP*-complete problems have distillation algorithms, none of the following* FPT *problems have polynomial kernels:* k-Cutwidth, k-Modified Cutwidth, k-Search Number, k-Pathwidth, k-Treewidth, k-Branchwidth, k-Gate Matrix Layout, k-Front Size, w-3-Coloring *and* w-3-Domatic Number.

We remark that in unpublished work, Buhrman [7] has shown that there are oracles relative to which no coNP-complete problem has a distillation algorithm. We believe that the same information-theoretic intuition applies here, and that no coNP-complete problem can have a distillation algorithm.

In the full version of this paper, we also study *sub-exponential kernels*, *i.e* kernelization algorithms that reduce instances to a size which is *sub-exponentially* bounded by the parameter. In particular, we prove that there are problems solvable in $O(2^k n)$ time which (unconditionally) do not admit a kernel of size $2^{o(k)}$. This relates our material to the work of Flum, Grohe, and Weyer [18] who introduced the notion of "bounded fixed-parameter tractability" as an attempt to provide a theory for feasible FPT algorithms. They argued that for an FPT algorithm to be useful in practice, it should most likely have a running time of $2^{O(k)}n^{O(1)}$ or perhaps $2^{k^{O(1)}}n^{O(1)}$. We show that the notion of small kernel and small running-time are quite different.

2 Preliminaries

Throughout the paper, we let Σ denote a finite alphabet, and $\mathbb{N}$ the set of natural numbers. A (classical) problem L is a subset of Σ^*, where Σ^* is the set of all

finite length strings over Σ. In natural cases, the strings in L will be an encoding of some combinatorial object, *e.g.* graphs. We will call strings $x \in \Sigma^*$ which are proper encodings, *input* of L, regardless of whether $x \in L$. We will often not distinguish between a combinatorial object and its string encoding, using for example G to denote both a graph and a string in Σ^*.

A *parameterized problem* is a subset $L \subseteq \Sigma^* \times \mathbb{N}$. In this way, an input (x, k) to a parameterized language consists of two parts, where the second part k is the *parameter*. A parameterized problem L is *fixed-parameter tractable* if there exists an algorithm which on a given $(x, k) \in \Sigma^* \times \mathbb{N}$, decides whether $(x, k) \in L$ in $f(k)p(n)$ time, where f is an arbitrary computable function solely in k, and p is a polynomial in the total input length (including the unary encoding of the parameter) $n = |x| + k$. Such an algorithm is said to run in FPT-*time*, and FPT is the class of all parameterized problems that can be solved by an FPT-time algorithm (*i.e.* all problems which are fixed-parameter tractable). For more background on parameterized complexity, the reader is referred to [5,12,17].

To relate notions from parameterized complexity and notions from classic complexity theory with each other, we use a natural way of mapping parameterized problems to classical problems. The mapping of parameterized problems is done by mapping (x, k) to the string $x\#1^k$, where $\# \notin \Sigma$ denotes the blank letter and 1 is an arbitrary letter in Σ. In this way, the *unparameterized version* of a parameterized problem L is the langauge $\widetilde{L} = \{x\#1^k \mid (x, k) \in L\}$. We next give a formal definition for the central notion of this paper:

Definition 1 (Kernelization). *A* kernelization algorithm*, or in short, a* kernel *for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in $p(|x| + k)$ time a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that*

- $(x, k) \in L \Leftrightarrow (x', k') \in L$,
- $|x'|, k' \leq f(k)$,

where f is an arbitrary computable function, and p a polynomial. Any function f as above is referred to as the size *of the kernel.*

That is, if we have a kernel for L, then for any $(x, k) \in \Sigma \times \mathbb{N}$, we can obtain in polynomial time an equivalent instance with respect to L whose size is bounded by a function of the parameter. If the size of the kernel is polynomial, we say that the parameterized langauge L has a *polynomial kernel*.

There is also a more general definition for kernelization than the one given above which sometimes appears in practice. This definition allows a kernelization algorithm for a parameterized problem L to map an instance of L to an instance of another problem L'. We remark that all results in this paper easily follow for most cases of the more general definition. Nevertheless, we will present these results with Definition 1 for the sake of clarity and simplicity.

3 A Generic Lower-Bounds Engine

In the following we develop the main engine for proving Theorems 1 and 2. This engine evolves around the notion of *distillation algorithms* for NP-complete problems.

We first introduce this notion, and then define a parameterized analog that we call a composition algorithm. Following this, we show that if a compositional parameterized problem has a polynomial kernel, then its unparameterized counterpart has a distillation algorithm.

Definition 2 (Distillation). *A* distillation algorithm *for a classical problem* $L \subseteq \Sigma^*$ *is an algorithm that receives as input a sequence* $(x_1, \ldots, x_t)$*, with* $x_i \in \Sigma^*$ *for each* $1 \leq i \leq t$*, uses time polynomial in* $\sum_{i=1}^{t} |x_i|$*, and outputs a string* $y \in \Sigma^*$ *with*

1. $y \in L \iff x_i \in L$ *for some* $1 \leq i \leq t$.
2. $|y|$ *is polynomial in* $\max_{1 \leq i \leq t} |x_i|$.

That is, given a sequence of t instances of L, a distillation algorithm gives an output that is equivalent to the sequence of instances, in the sense that a collection with at least one yes-instance (*i.e.* instance belonging to L) is mapped to a yes-instance, and a collection with only no-instances is mapped to a no-instance. (In a certain sense, this functions like a Boolean *OR* operator.) The algorithm is allowed to use polynomial-time in the total size of all instances. The crux is that its output must be bounded by a polynomial in the size of the largest of the instances from the sequence, rather than in the total length of the instances in the sequence. We next introduce the notion of a composition algorithm for parameterized problems. In some sense, one can view a composition algorithm as the parameterized analog of a distillation algorithm.

Definition 3 (Composition). *A* composition algorithm *for a parameterized problem* $L \subseteq \Sigma^* \times \mathbb{N}$ *is an algorithm that receives as input a sequence* $((x_1, k), \ldots, (x_t, k))$*, with* $(x_i, k) \in \Sigma^* \times \mathbb{N}^+$ *for each* $1 \leq i \leq t$*, uses time polynomial in* $\sum_{i=1}^{t} |x_i| + k$*, and outputs* $(y, k') \in \Sigma^* \times \mathbb{N}^+$ *with*

1. $(y, k') \in L \iff (x_i, k) \in L$ *for some* $1 \leq i \leq t$.
2. k' *is polynomial in* k.

Hence, given a sequence of instances for L, a composition-algorithm outputs an equivalent instance to this sequence in the same sense as a distillation algorithm, except that now the parameter of the instance is required to be polynomially-bounded by parameter appearing in all instances of the sequence, rather than the size of the instance bounded by the maximum size of of all instances.

We call classical problems with distillation algorithms *distillable problems*, and parameterized problems with composition algorithms *compositional problems*. Despite the similarities between the two definitions, as we shall soon see, the existence of composition algorithms for some parameterized problems is much more plausible than the existence of distillations for their unparameterized counterparts. Nevertheless, there is still a deep connection between distillation and composition, obtained via polynomial kernelization. In particular, in the following lemma we prove that combining a composition algorithm for a parameterized problem L, with a polynomial kernel for it, admits a distillation algorithm for the unparameterized counterpart of L.

Lemma 1. *Let L be a compositional parameterized problem whose unparameterized version $\widetilde{L}$ is* NP*-complete. If L has a polynomial kernel, then $\widetilde{L}$ is also distillable.*

Proof. Let $\widetilde{x}_1, \ldots, \widetilde{x}_t \in \Sigma^*$ be instances of $\widetilde{L}$, and let $(x_i, k_i) \in \Sigma^* \times \mathbb{N}^+$ denote the instance of L derived from $\widetilde{x}_i$, for all $1 \leq i \leq t$. Since $\widetilde{L}$ is NP-complete, there exist two polynomial-time transformations $\Phi : \widetilde{L} \to \text{SAT}$ and $\Psi : \text{SAT} \to \widetilde{L}$, where SAT is the problem of deciding whether a given boolean formula is satisfiable. We use the composition and polynomial kernelization algorithms of L, along with Φ and Ψ, to obtain a distillation algorithm for $\widetilde{L}$. The distillation algorithm proceeds in three steps.

Set $k = \max_{1 \leq i \leq t} k_i$. In the first step, we take the subsequence in $((x_1, k_1), \ldots, (x_t, k_t))$ of instances whose parameter equals ℓ, for each $1 \leq \ell \leq k$. We apply the composition algorithm on each one of these subsequence separately, and obtain a new sequence $((y_1, k'_1), \ldots, (y_r, k'_r))$, where (y_i, k'_i), $1 \leq i \leq r$, is the instance obtained by composing all instances with parameters equaling the i'th parameter value in $\{k_1, \ldots, k_t\}$. In the second step, we apply the polynomial kernel on each instance of the sequence $((y_1, k'_1), \ldots, (y_r, k'_r))$, to obtain a new sequence $((z_1, k''_1), \ldots, (z_r, k''_r))$, with (z_i, k''_i) the instance obtained from (y_i, k'_i), for each $1 \leq i \leq r$. Finally, in the last step, we transform each $\widetilde{z}_i$, the unparameterized instance of $\widetilde{L}$ derived from (z_i, k''_i), to a Boolean formula $\Phi(\widetilde{z}_i)$. We output the instance of $\widetilde{L}$ for which Ψ maps the disjunction of these formulas to, *i.e.* $\Psi(\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i))$.

We argue that this algorithm distills the sequence $(\widetilde{x}_1, \ldots, \widetilde{x}_t)$ in polynomial time, and therefore is a distillation algorithm for $\widetilde{L}$. First, by the correctness of the composition and kernelization algorithms of L, and by the correctness of Φ and Ψ, it is not difficult to verify that $\Psi(\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i)) \in \widetilde{L} \iff \widetilde{x}_i \in \widetilde{L}$ for some i, $1 \leq i \leq t$. Furthermore, the total running-time of our algorithm is polynomial in $\sum_{i=1}^{t} |\widetilde{x}_i|$. To complete the proof, we show that the final output returned by our algorithm is polynomially bounded in $n = \max_{1 \leq i \leq t} |\widetilde{x}_i|$. The first observation is that since each $\widetilde{x}_i$ is derived from the instance (x_i, k_i), $1 \leq i \leq t$, we have $r \leq k = \max_{1 \leq i \leq t} k_i \leq \max_{1 \leq i \leq t} |\widetilde{x}_i| = n$. Therefore, there are at most n instances in the sequence $((y_1, k'_1), \ldots, (y_r, k'_r))$ obtained in the first step of the algorithm. Furthermore, as each (y_i, k'_i), $1 \leq i \leq r$, is obtained via composition, we know that k'_i is bounded by some polynomial in $\ell \leq k \leq n$. Hence, since for each $1 \leq i \leq r$, the instance (z_i, k''_i) is the output of a polynomial kernelization on (y_i, k'_i), we also know that (z_i, k''_i) and $\widetilde{z}_i$ have size polynomially-bounded in n. It follows that $\sum_{i=1}^{r} |\widetilde{z}_i|$ is polynomial in n, and since both Φ and Ψ are polynomial-time, so is $\Psi(\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i))$. □

We conclude this section by stating a lemma proven by by Fortnow and Santhanam [19], which verifies our initial intuition that NP-complete problems are unlikely to have distillation algorithms. It is clear from Definition 2, that if any NP-complete problem were distillable, then they all would be – we can use the polynomial-time reductions provided for any NP-complete problem $\widetilde{L}$ to and from our presumed distillable NP-complete problem to distill $\widetilde{L}$. Fortnow and

Santhanam proved that a distillation algorithm for any NP-complete problem would imply coNP $\subseteq$ NP/poly, which on its turn implies that the polynomial hierarchy collapses to at most three levels [31], a hierarchy generally believed to be proper.

Lemma 2 ([19]). *If any* NP*-complete problem has a distillation algorithm then* coNP $\subseteq$ NP/poly*.*

4 Applications

Lemmas 1 and 2 that together form our lower bound engine together imply that any compositional parameterized problem whose unparameterized counterpart is NP-complete cannot have a polynomial kernel, unless the polynomial hierarchy collapses. In the following we show the strength of our lower bound engine by giving several examples of compositional FPT problems that are based on unparameterized classical NP-complete problems. We focus only on natural examples, and in particular, we complete the proof of Theorem 1.

Let us call a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ a *parameterized graph problem*, if for any $(x, k) \in L$, x is an encoding of a graph.

Lemma 3. *Let L be a parameterized graph problem such that for any pair of graphs G_1 and G_2, and any integer $k \in \mathbb{N}$, we have $(G_1, k) \in L \vee (G_2, k) \in L \iff (G_1 \cup G_2, k) \in L$, where $G_1 \cup G_2$ is the disjoint union of G_1 and G_2. Then L is compositional.*

As an immediate corollary of the simple lemma above, we get that our case-study problem k-Path is compositional, and thus is unlikely to have a polynomial kernel. Indeed, the disjoint union of two graphs has a k-path iff one of the graphs has a k-path. Two other similar examples are the k-Cycle and k-Exact Cycle problems, which respectively ask to determine whether a given graph has a (not necessarily induced) subgraph which is isomorphic to a cycle with at least k vertices and a cycle with exactly k vertices. Both these problems are also in FPT by the color-coding technique of Alon *et al.* [2], and are compositional by the lemma above. Another example is k-Short Cheap Tour, which given an edge-weighted graph, asks whether there is a tour of length at least k in the graph with total weight not more than some given threshold. This problem is in FPT due to [29], and is again compositional according to Lemma 3.

In fact, Lemma 3 implies that any parameterized problem which asks to determine whether a specific graph H (*e.g.* a k-clique) is a "subgraph of some kind" of an input graph G, for almost any natural notion of subgraph, is compositional when parameterized by H (or more precisely, by the *numeric encoding of* H, the position of H in some canonical ordering of simple graphs). For example, consider the k-Minor Order Test problem, famously in FPT due to Robertson and Seymour's celebrated Graph Minor Theorem. This problem asks to decide whether a given graph H is a minor of another given graph G, and the parameter k is H. Clearly, if we slightly relax the problem and require H to be

connected, the disjoint union construction of Lemma 3 above gives a composition algorithm for this problem. If H is not connected, then we can connect it by adding a new *global* vertex adjacent to all other vertices in H, and then add such a global vertex to each G_i, $1 \leq i \leq t$. By similar arguments we can also show that k-Bounded Treewidth Subgraph Test – the problem of determining whether a given bounded treewidth graph occurs as a subgraph in another given graph (in FPT again via color-coding [2]) – is also compositional. Two other good examples are k-Planar Graph Subgraph Test and k-Planar Graph Induced Subgraph Test, both in FPT due to [13].

We now turn to proving the last item of Theorem 1. In particular, we show that many natural NP-complete problems parameterized by treewidth are unlikely to have a polynomial kernel. We illustrate the technique with one example, and then state the general result that can be obtained using the same way. Consider the w-Independent Set problem: Given a graph G, a tree-decomposition $\mathcal{T}$ of G of width $w \in \mathbb{N}^+$, and an integer $k \in \mathbb{N}^+$, determine whether G has an independent set of size k. Note that the parameter here is w and not k. We call the unparameterized variant of w-Independent Set the Independent Set with Treewidth problem. Clearly, Independent Set with Treewidth is NP-complete by the straightforward reduction from Independent Set which appends a trivial tree-decomposition to the given instance of Independent Set.

To show that w-Independent Set is compositional, we will work with a 'guarantee' version, the w-Independent Set Refinement problem: given a graph G, a tree-decomposition $\mathcal{T}$ of G, and an independent set I in G, determine whether G has an independent set of size $|I| + 1$. The parameter is the width w of $\mathcal{T}$. The unparameterized variant of w-Independent Set Refinement is Independent Set Refinement with Treewidth. It is easy to see that this problem is NP-complete by the following reduction from Independent Set with Treewidth – Given an instance $(G, \mathcal{T}, k)$, construct the instance $(G', \mathcal{T}', I)$, where G' is the graph obtained by adding $k - 1$ new pairwise non-adjacent vertices I to G which are connected to all the old vertices, and $\mathcal{T}'$ is the tree-decomposition obtained by adding I to each node in $\mathcal{T}$.

Lemma 4. *w-Independent Set Refinement is compositional, and furthermore, if w-Independent Set has a polynomial kernel then so does w-Independent Set Refinement.*

The proof of the lemma above (which we omit due to space constraints) implies that to fit a natural NP-complete graph problem parameterized by treewidth into the context of our lower-bound framework, one has to basically show two things: First, that the refinement variant of the problem is compositional, and second, that the unparameterized version of the refinement variant is NP-complete. In fact, this technique is not necessarily limited to treewidth, but can be used with almost any other structural parameter such as cliquewidth, maximum degree, minimum vertex cover, and so forth. To complete the proof of Theorem 1, what is left to prove is that Dominating Set Refinement with Treewidth is NP-complete; Clique Refinement with Treewidth can be seen to be

NP-complete by a similar construction used in the proof of Lemma 4 above. We omit the details.

5 Extensions

We next extend the framework presented in the previous section so that it captures other important FPT problems not captured by Theorem 1. In particular, we discuss the proof for Theorem 2. The main observation is that an AND-variant of a composition algorithm for a parameterized problem L, yields a composition algorithm for $\overline{L}$, the complement of L. This observation is useful since a lot of problems have natural AND-compositions rather than regular compositions (*i.e.* OR-compositions). As any FPT problem has a polynomial kernel iff its complement also has one, showing that a coFPT problem is compositional is just as good for our purposes as showing that its complement in FPT is compositional.

Lemma 5. *Let L be a parameterized graph problem such that for any pair of graphs G_1 and G_2, and any integer $k \in \mathbb{N}$, we have $(G_1, k) \in L \wedge (G_2, k) \in L \iff (G_1 \cup G_2, k) \in L$, where $G_1 \cup G_2$ is the disjoint union of G_1 and G_2. Then $\overline{L}$, the complement of L, is compositional.*

There are many FPT problem with a natural composition as above. These include the classical "width problems" k-Pathwidth, k-Treewidth, and k-Branchwidth (see [4] for formal definitions and FPT algorithms for these problems). Three closely related relatives of these problems are k-Search Number [15], k-Front Size [4], and k-Gate Matrix Layout [16], which all have AND-composition by the lemma above. Lemma 5 also implies that two other famous FPT "width problems" are AND-compositional, namely, k-Cutwidth and k-Modified Cutwidth [15].

We prove the last item of Theorem 2 by using refinement variants as done for the treewidth parameterized problems in Theorem 1. In this context, it is worth mentioning that partitioning problems seem more adaptable to AND-compositions, as opposed to subset problems which are better suited for regular composition. Recall that w-3-Chromatic Number is the problem of determining, given a graph G and a tree-decomposition $\mathcal{T}$ of G, whether there exists a partitioning (or *coloring*) Π of $V(G)$ into three classes, where each class induces an independent set in G. The parameter is the width of $\mathcal{T}$. The w-3-Domatic Number problem is defined similarly, except that here the goal is to partition (or *domatic-color*) $V(G)$, again into three classes, with each class inducing a dominating set of G. Indeed, we selected w-3-Chromatic Number and w-3-Domatic Number for Theorem 2 as they are two of the more well-known graph partitioning problems. Many other natural partitioning problems could have been selected as well.

The refinement variants of these two problems, w-3-Chromatic Number Refinement and w-3-Domatic Number Refinement, are defined by adding to the input an appropriate vertex-partitioning Π (with respect to the problem definition), of cardinality four for w-3-Chromatic Number Refinement and

two for w-3-Domatic Number Refinement. It is easy to see that the unparameterized versions of these two problems are NP-complete by recalling that one can color planar graphs with four colors in polynomial-time (see *e.g.* [30]), while it is NP-complete to decide whether a planar graph is 3-colorable, and by recalling that every graph without an isolated vertex can be domatic-colored with two colors in polynomial-time (see *e.g.* [20]). Furthermore, it is easy to see that the standard disjoint union algorithm is an AND-composition for these two problems. Thus, by similar arguments used in Section 4, we can conclude that a polynomial-kernel for either w-3-Chromatic Number or w-3-Domatic Number implies that all coNP-complete problems are distillable.

Acknowledgements

We would like to thank Lance Fortnow, Raul Santhanam, and Harry Buhrman for many fruitful discussions. In particular, Lance and Raul provided the proof for Lemma 2 of this paper. The fourth author would also like to thank Moritz Müller for reviewing several preliminary versions, and especially for the countless (and sometimes endless) debates on related topics.

References

1. Alber, J., Fellows, M.R., Niedermeier, R.: Efficient data reduction for DOMINATING SET: A linear problem kernel for the planar case. In: Penttonen, M., Schmidt, E.M. (eds.) SWAT 2002. LNCS, vol. 2368, pp. 150–159. Springer, Heidelberg (2002)
2. Alon, N., Yuster, R., Zwick, U.: Color coding. Journal of the ACM 42(4), 844–856 (1995)
3. Arnborg, S.: Efficient algorithms for combinatorial problems on graphs with bounded decomposability. A survey. BIT Numerical Mathematics 25(1), 2–23 (1985)
4. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)
5. Bodlaender, H.L.: A cubic kernel for feedback vertex set. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 320–331. Springer, Heidelberg (2007)
6. Bonsma, P.S., Brüggemann, T., Woeginger, G.J.: A faster FPT algorithm for finding spanning trees with many leaves. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 259–268. Springer, Heidelberg (2003)
7. Buhrman, H.: Private communication (2007)
8. Burrage, K., Estivill-Castro, V., Fellows, M.R., Langston, M.A., Mac, S., Rosamond, F.A.: The undirected feedback vertex set problem has a Poly(k) kernel. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 192–202. Springer, Heidelberg (2006)
9. Cai, L., Chen, J., Downey, R.G., Fellows, M.R.: Advice classes of paramterized tractability. Annals of Pure and Applied Logic 84(1), 119–138 (1997)
10. Cai, L., Juedes, D.W.: Subexponential parameterized algorithms collapse the W-hierarchy. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 273–284. Springer, Heidelberg (2001)

11. Chen, Y., Flum, J., Müller, M.: Lower Bounds for Kernelizations – (manuscript, 2007)
12. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
13. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. In: Proceedings of the 6th annual ACM/SIAM Symposium on Discrete Algorithms (SODA), pp. 632–640 (1995)
14. Fellows, M.R., Langston, M.A.: Nonconstructive proofs of polynomial-time complexity. Information Processing Letters 26, 157–162 (1988)
15. Fellows, M.R., Langston, M.A.: On well-partial-order theory and its application to combinatorial problems of VLSI design. SIAM Journal of Discrete Math. 5(1), 117–126 (1992)
16. Fernau, H.: Parameterized algorithms: A graph-theoretic approach. PhD thesis, University of Tübingen (2005)
17. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
18. Flum, J., Grohe, M., Weyer, M.: Bounded fixed-parameter tractability and $\log^2 n$ nondeterministic bits. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 555–567. Springer, Heidelberg (2004)
19. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. In: Proceedings of the 40th Symposium on the Theory of Computing (STOC) (to appear, 2008)
20. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
21. Gramm, J., Guo, J., Huffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. Mathematical Systems Theory 38, 373–392 (2005)
22. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
23. Guo, J., Niedermeier, R., Wernicke, S.: Parameterized complexity of generalized vertex cover problems. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 36–48. Springer, Heidelberg (2005)
24. Harnik, D., Naor, M.: On the compressibility of NP instances and cryptographic applications. In: Proceedings of 47th annual IEEE symposium on Foundations Of Computer Science (FOCS), pp. 719–728 (2006)
25. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Divide-and-color. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 58–67. Springer, Heidelberg (2006)
26. Ko, K.: On self-reducibility and weak P-selectivity. Journal of Computer and System Sciences 26(2), 209–221 (1983)
27. Liu, Y., Lu, S., Chen, J., Sze, S.: Greedy localization and color-coding: Improved matching and packing algorithms. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 84–95. Springer, Heidelberg (2006)
28. Niedermeier, R., Rossmanith, P.: An efficient fixed-parameter algorithm for 3-Hitting Set. Journal of Discrete Algorithms 1, 89–102 (2003)
29. Plehn, J., Voigt, B.: Finding minimally weighted subgraphs. In: Möhring, R.H. (ed.) WG 1990. LNCS, vol. 484, pp. 18–29. Springer, Heidelberg (1991)
30. Robertson, N., Sanders, D.P., Seymour, P., Thomas, R.: Efficiently four-coloring planar graphs. In: Proceedings of the 28th annual ACM Symposium on the Theory Of Computing (STOC), pp. 571–575 (1996)
31. Stockmeyer, L.J.: The polynomial-time hierarchy. Theoretical Computer Science 3, 1–22 (1977)

Faster Algebraic Algorithms for Path and Packing Problems*

Ioannis Koutis

Carnegie Mellon University
Computer Science Department
Pittsburgh, PA 15213
ioannis.koutis@cs.cmu.edu

Abstract. We study the problem of deciding whether an n-variate polynomial, presented as an arithmetic circuit G, contains a degree k square-free term with an odd coefficient. We show that if G can be evaluated over the integers modulo 2^{k+1} in time t and space s, the problem can be decided with constant probability in $O((kn+t)2^k)$ time and $O(kn+s)$ space. Based on this, we present new and faster algorithms for two well studied problems: (i) an $O^*(2^{mk})$ algorithm for the m-set k-packing problem and (ii) an $O^*(2^{3k/2})$ algorithm for the simple k-path problem, or an $O^*(2^k)$ algorithm if the graph has an induced k-subgraph with an odd number of Hamiltonian paths. Our algorithms use $poly(n)$ random bits, comparing to the $2^{O(k)}$ random bits required in prior algorithms, while having similar low space requirements.

1 Introduction

This paper presents new and faster randomized algorithms for the well studied parameterized simple path and set packing problems.

The k-path problem asks for a simple path of length k in a graph of n vertices V and m edges E. In the general case where k is not considered a parameter, the longest path and the Hamiltonian path problems are well known to be NP-hard. Papadimitriou and Yannakakis conjectured that the $O(\log n)$-path problem can be solved in polynomial time [9]. Alon et. al., introducing color-coding, confirmed the conjecture by describing an $O^*(5.44^k)$ [1] randomized algorithm and an $O^*(c^k)$ deterministic algorithm for some constant $c > 8000$ [1]. The space complexity of both algorithms is $O^*(2^k)$. A big step towards closing the gap between the randomized and the deterministic complexity was taken by Kneis et. al. who presented an $O^*(16^k)$ deterministic algorithm [6]. The deterministic complexity was further reduced by Chen et. al. [2], who presented more efficient color-coding schemes and gave an $O^*(12.8^k)$ time deterministic algorithm. A new

* This work was partially supported by the National Science Foundation under grant number CCF-0635257.

[1] Following the parameterized complexity notation, we use $O^*(f(k))$ to hide a $poly(n)$ factor, but we will also use $O^*(f(k)poly(n))$ to hide factors of lower order.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 575–586, 2008.

randomized divide-and-conquer algorithm -the first to deviate from the color-coding approach- was given in [6] and independently in [2]. Its time complexity is $O^*(4^k)$ and its space complexity $O(nk \log k + m)$.

The m-set k-packing problem gives a universe of n elements and a collection C of N sets each consisting of m elements; it asks for a sub-collection of k pairwise disjoint sets from C. In the general case where k is not considered a parameter, the set packing problems for all $m \geq 3$ are well known to be NP-hard. For the special case $m = 3$, the problem was first considered in [3] where a deterministic $O^*(2^{O(k)}(3k)!)$ algorithm was given. The bound was subsequently improved to $O^*(5.7k)^k$ in [5]. Later, it was realized that the m-set packing problem is amenable to a dynamic programming/color-coding approach; this led to an $O^*(5.44^{3k})$ randomized algorithm in [7], and to an $O^*(12.7D)^{3k}$ for $D \geq 10.4$ deterministic algorithm in [4]. In both cases the space complexity is $O^*(2^{3k})$. The deterministic time complexity was improved to $O^*(4.68^{3k})$ in [8]. The randomized divide-and-conquer algorithm of [2] improved the randomized time complexity to $O^*(2.52^{3k})$ and the space complexity to $O(nk \log k)$.

While the recent algorithmic progress in the path and packing problems has been impressive, both the color-coding and the randomized divide-and-conquer approaches seem to have an inherent time complexity limit, namely the $O^*(2^k)$ bound for the k-path and the $O^*(2^{3k})$ bound for the 3-set k-packing. On the other hand, the Hamiltonian path and the general m-set (n/m)-packing problem have fairly easy $O^*(2^n)$ algorithms. In view of this, a question presents itself: is there a $O^*(2^k)$ algorithm for the k-path and an $O^*(2^{mk})$ algorithm for the m-set k-packing?

1.1 Our Approach and Contributions

We answer the question in the affirmative for the m-set k-packing. We describe an algorithm that decides the problem in $O^*(2^{km}(km)^2N)$ expected time, and based on that, an algorithm that finds a packing in $O^*(2^{km}(km)^2N^2)$ expected time. Both algorithms use $O(kn)$ space. The answer is "almost" positive for the k-path problem as well. In the case the graph contains an odd number of k-paths (or generally when any subgraph induced by k vertices has an odd number of Hamiltonian paths), we describe an algorithm that finds a k-path in $O^*(2^k k^2 m(n + \min(k^2, m)))$ expected time and $O(kn + m)$ space. In the general case, we describe an algorithm that decides the problem in $O^*(2^{3k/2}k^2m)$ expected time and finds a k-path in $O^*(2^{3k/2}k^2m(n + \min(k^2, m)))$ expected time and $O(k^2m)$ space. We should note that we have been working on a slight extension of the techniques presented in this paper and we believe that the general time complexity can be reduced to $O^*(2^k)$; the details will be included in the full version of the paper. Our algorithms use $poly(n)$ random bits, improving the $2^{O(k)}poly(n)$ random bits required in prior algorithms. As we will discuss in more detail later, the possibility that our algorithm can be derandomized with only a polynomial slowdown, doesn't look very remote.

Our approach is based on reductions (mentioned or presented in [11] and [7]) to the problem of detecting a square-free term of degree k in polynomials

represented as arithmetic circuits. We describe a fast and space efficient algorithm for detecting square-free terms with odd coefficients. In Section 2 we show that if the polynomial P can be evaluated over the integers modulo 2^{k+1} in time t and space s, the existence of the "odd" square-free term in P can be decided with constant probability in $O((kn+t)2^k)$ time and $O(kn+s)$ space. In Section 3 we describe a randomized reduction of the m-set k-packing to the odd k-term problem. In Section 4 we describe a deterministic reduction of the k-path problem to the odd $3k/2$-term problem. Finally in Section 5 we discuss some related open questions.

2 Detecting Square-Free Terms with Odd Coefficients

Let $X = x_1, \ldots, x_n$ and let $K[X]$ be the commutative ring of polynomials with coefficients from the field K. Any non-zero polynomial $\mathbb{Z}_2[X]$ is by definition a sum (or equivalently a set) of monomials. A monomial is called square-free or multilinear if it is linear in all its variables. The total degree of a monomial is the sum of the degrees of its variables. In general, any polynomial $P \in \mathbb{Z}_2[X]$ can be represented as an arithmetic circuit which is a directed acyclic graph with addition and multiplication gates, and terminals corresponding to the variables.

Definition 1. *The ODD MULTILINEAR k-TERM problem: Given an arithmetic circuit G decide whether the polynomial $P(X) \in \mathbb{Z}_2[X]$ represented by G contains a multilinear term of total degree k or less.*

The main idea of our algorithm for this problem is the evaluation of the given polynomial over a suitably selected *commutative algebra*. In a commutative algebra the addition and multiplication operators behave in the same way as in the common algebra of integers or reals. This enables looking at the polynomial in two equivalent ways; its circuit representation allows its fast evaluation, whereas its expanded form as a sum of monomials allows us reasoning about its value in terms of the individual evaluations of its monomials. The slightly counterintuitive fact is that a commutative algebra may contain elements whose square is 0. We will exploit this to annihilate non-multilinear terms in the evaluation of P. It turns out that this can be done using commutative *group algebras* of $\mathbb{Z}_2^k$. We refer the reader to the Appendix for definitions and facts. We now give an algorithm for the ODD MULTILINEAR k-TERM problem, that works with the assumption that P contains only multilinear terms of degree exactly k. We will then see how this restriction can be easily removed.

decide-multilinear: Given an instance of the ODD MULTILINEAR k-TERM problem: **(i)** For each $x_i \in X$, independently pick a random vector $v_i \in \mathbb{Z}_2^k$ and assign to it the value $(v_0 + v_i) \in \mathbb{Z}_2[\mathbb{Z}_2^k]$, where v_0 is the k-dimensional zero vector; Let $\bar{X}$ denote the assignment $x_i \leftarrow v_0 + v_i$. **(ii) -[option 1]:** If the coefficient of v_0 in $P(\bar{X})$ is equal to 1, then return "yes" otherwise return "no". **(ii) -[option 2]:** Let b_t denote the k dimensional vector containing the binary form of t, $\bar{\Lambda}_t$ denote the assignment $x_i \leftarrow 1 + (-1)^{v_i^T b_t}$ and $Z = \sum_{t=0}^{2^k-1} P(\bar{\Lambda}_t)$. If Z is equal to $2^k \ mod \ 2^{k+1}$ then return "yes", otherwise return "no".

It may appear that given the two options for step (ii), **decide-multilinear** gives two algorithms. However, in Theorem 2 we will prove that option 2 is equivalent to option 1 and thus it just provides an alternative implementation; from this we will derive our complexity claims. We will prove the soundness of the algorithm in Theorem 1, using option 1. In order to proceed with the proofs we need to introduce some notation. For simplicity let $\mathcal{A}$ denote $\mathbb{Z}_2[\mathbb{Z}_2^k]$. If $S \subset \mathbb{Z}_2^k$ is a set of vectors, we denote by $\pi(S)$ their product in $\mathcal{A}$. By convention, for all sets $V \subseteq \mathbb{Z}_2^k$ we will consider the empty set as a subset of S and we will let $\pi(\emptyset) = v_0$. Note that $\pi(A)\pi(B) = v_0$ if and only if $\pi(A) = \pi(B)$. We let J denote the element of A which is the sum of all vectors in $\mathbb{Z}_2^k$, that is $J = \sum_{v \in \mathbb{Z}_2^k} v$. We also say that an element w of $\mathcal{A}$ is *split* if it is the sum of exactly 2^{k-1} distinct vectors.

By construction, each monomial evaluates to an element of the form $\Pi(V) = \prod_{v \in V}(v_0 + v)$, where $V \subset \mathbb{Z}_2^k$. Using the fact that for all $v \in \mathbb{Z}_2^k$ we have $v_0 v = v$, we can expand $\Pi(V)$ into a sum, to get

$$\Pi(V) = \prod_{v \in V}(v_0 + v) = \sum_{S \subseteq V} \pi(S). \tag{1}$$

Lemma 1. *If the vectors in $V \subseteq \mathbb{Z}_2^k$ are linearly dependent over $\mathbb{Z}_2$, $\Pi(V)$ evaluates to 0. If the vectors in V are linearly independent, $\Pi(V)$ is a sum of $2^{|V|}$ distinct vectors (including v_0).*

Proof. By definition, when the vectors in V are linearly dependent, there is $V' \subseteq V$ such that $\pi(V') = v_0$. Then for all $S \subseteq V'$ we have $\pi(S)\pi(V' - S) = v_0$, which implies that $\pi(S) = \pi(V' - S)$. Hence, every term in the sum expansion (equality 1) of $\Pi(V')$ is generated an even number of times, which gives us $\Pi(V') = 0$. This in turn implies $\Pi(V) = 0$, because $\Pi(V) = \Pi(V)\Pi(V - V')$. This shows the first part of the Lemma. For the second part of the Lemma we observe that for all $S_a \neq S_b \subseteq V$ we have $\pi(S_a) \neq \pi(S_b)$. To see why, note that if $\pi(S_a) = \pi(S_b)$, then $\pi(S_a)\pi(S_b) = \pi(S_a S_b) = v_0$, which implies that the vectors in $(S_a \cup S_b - S_a \cap S_b)$ are linearly dependent, a contradiction. Therefore, since there are $2^{|V|}$ possible subsets of V (including $\emptyset$), $\Pi(V)$ is a sum of $2^{|V|}$ distinct vectors (including v_0). □

Lemma 2. *Let $P_{k-1} \in \mathbb{Z}_2[X]$ be a sum of multilinear monomials of degree exactly $k-1$.* **[a]** *For all assignments $\bar{X}$ of the form $x_i \leftarrow (v_0 + v_i)$, $P_{k-1}(\bar{X})$ is either split, or equal to 0, or equal to J.* **[b]** *In the case $P_{k-1}(\bar{X})$ is split, we have $Prob_{v \in \mathbb{Z}_2^k}((v_0 + v)P_{k-1}(\bar{X}) = J) = Prob_{v \in \mathbb{Z}_2^k}((v_0 + v)P_{k-1}(\bar{X}) = 0) = 1/2$.*

Proof. Let $P_{k-1} = \sum_j M_j$ where each M_j is a monomial of degree $k-1$. We will derive the Lemma by looking at $I = (v_0 + v)P_{k-1}(\bar{X})$ for a proper vector v. Note that $\mathbb{Z}_2^k$ contains 2^k vectors. Lemma 1 then implies that for all $v \in \mathbb{Z}_2^k$ and all M_j, we have $(v_0 + v)M_j(\bar{X}) = 0$ or $(v_0 + v)M_j(\bar{X}) = J$. Therefore, we have $I = 0$ or $I = J$, so the coefficient of any vector in I completely determines the value of I.

Clearly, this allows the possibilities $P_{k-1}(\bar{X}) = 0$ and $P_{k-1}(\bar{X}) = J$. Now assume that $P_{k-1}(\bar{X})$ is a sum of t distinct vectors, where $1 < t < 2^k$. We have

$$I = (v_0 + v)P_{k-1}(\bar{X}) = P_{k-1}(\bar{X}) + vP_{k-1}(\bar{X}).$$

Since $vv_1 = vv_2$ implies $v_1 = v_2$, it must be that $vP_{k-1}(\bar{X})$ is a sum of t distinct vectors in $\mathbb{Z}_2^k$. Hence, every vector in the expansion of I is generated $0, 1$ or 2 times. If some vector w is generated two times, its coefficient in I will be 0, and hence $I = 0$.

Now pick a vector v such that $vP_{k-1}(\bar{X})$ contains a vector w which is not in $P_{k-1}(\bar{X})$; this is clearly always possible. In that case the coefficient of w in I is 1, thus $I = J$. In addition I is a sum of at most $2t$ vectors, and since J is the sum of 2^k vectors, we must have $t \geq 2^{k-1}$. If $t > 2^{k-1}$, a simple pigeonhole argument shows that there must be a vector w' which is generated two times in I, implying that $I = 0$. This is a contradiction, so we must have $t = 2^{k-1}$. The **[b]** claim follows from the fact that $t = 2^{k-1}$ and the observation that I contains the vector v_0 with probability $1/2$, with respect to the choice of v. □

We are ready to prove the soundness of the algorithm.

Theorem 1. *If P does not contain a multilinear term, the algorithm* **decide-multilinear** *returns "no". Otherwise it returns "yes" with probability greater than 1/4.*

Proof. For the first claim, note that every monomial q which is not multilinear can be written as $x_i^2 q'$ for some variable x_i and monomial q'. Now observe that $\bar{x}_i^2 = (v_0 + v_i)^2 = 0$. Hence $\bar{x}_i^2 q' = 0$. So, if P does not contain multilinear terms, all its terms evaluate to 0. Thus, $P(\bar{X}) = 0$, and the algorithms returns "no". We will show the other direction using induction on the number of multilinear terms in P. Recall our assumption that all these terms have degree exactly k.

Base case: Let $V = \{v_1, \ldots, v_k\}$ be k random vectors drawn independently from $\mathbb{Z}_2^k$. The probability that a multilinear monomial of degree k evaluates to J is by construction equal to $Pr(\prod_{i=1}^{k}(v_0 + v_i) = J)$. By Lemma 1, this is equal to the probability that the vectors in V are linearly independent. By standard linear algebra facts, given that $\{v_1, \ldots, v_{j-1}\}$ are linearly independent, the vectors $\{v_1, \ldots, v_j\}$ are independent if and only if v_j is not in the vector space $\mathcal{S}$ generated by $\{v_1, \ldots, v_{j-1}\}$. In Lemma 1 we showed that there are exactly 2^{j-1} distinct linear combinations of the $j-1$ vectors, so there are $2^k - 2^{j-1}$ vectors that are not in $\mathcal{S}$. Hence,

$$Prob(\prod_{i=1}^{j}(v_0 + v_i) \neq 0) = (1 - \frac{2^{j-1}}{2^k})Prob(\prod_{i=1}^{j-1}(v_0 + v_i) \neq 0) \geq \prod_{i=1}^{k}(1 - \frac{1}{2^i}).$$

The last product is lower bounded by $(1/2)\prod_{i=2}^{k}(1-1/i^2) = (k+1)/(4k) > 1/4$.

Inductive argument: If P is not a single monomial, then there is a variable x such that $P = xP_{k-1} + P'$ where x does not appear in $P' \neq 0$, and P_{k-1} is a

sum of multilinear terms of degree $k-1$. Using Lemma 2**[a]** we can consider all possible cases under which P evaluates to J, to get

$$\begin{aligned}
&Prob(xP_{k-1}+P'=J) &=\\
&Prob(P'=J)Prob(P_{k-1}=0 \text{ or } P_{k-1}=J/P'=J)+\\
&Prob(P'=J)Prob(P_{k-1} \text{ is split}/P'=J)Prob(xP_{k-1}=0/P_{k-1} \text{ is split})+\\
&Prob(P'=0)Prob(P_{k-1} \text{ is split}/P'=0)Prob(xP_{k-1}=J/P_{k-1} \text{ is split}) &\geq\\
&(1/2)Prob(P_{k-1} \text{ is split})=Prob(xP_{k-1}=J).
\end{aligned}$$

The inequality came from dropping the first term and -after applying Lemma 2**[b]**- combining the remaining two. The probabilities are taken with respect to the random assignment $\bar{X}$. The polynomial xP_{k-1} contains less monomials than $xP_{k-1}+P'$, hence the inductive hypothesis applies. □

Let A be a commutative algebra and $\bar{Y}$ be an assignment $x_i \leftarrow \bar{y_i} \in A$, for $i=1,\ldots,n$. We denote by $P_A(\bar{Y})$ the evaluation of P at $\bar{Y}$ over A.

Theorem 2. *Options 1 and 2 for step (ii) of* **decide-multilinear** *are equivalent. Furthermore, if the input circuit G can be evaluated over the integers modulo 2^{k+1} in time t and space s, option 2 can be performed in $O((nk+t)2^k)$ time and $O(nk+s)$ space.*

Proof. Let G be an arithmetic circuit with n variables X and $P \in \mathbb{Z}[X]$ be the polynomial represented by G. Also, let $\bar{X}$ be the assignment $x_i \leftarrow (v_0+v_i)$, as defined in **decide-multilinear**. Let $\rho(u)$ denote the matrix representation of $u \in \mathbb{Z}[\mathbb{Z}_2^k]$ and $trace(M)$ denote the sum of the diagonal elements of the matrix M. Observe that

$$P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})=\sum_{g\in\mathbb{Z}_2^k} a_g g \Rightarrow P_{\mathbb{Z}_2[\mathbb{Z}_2^k]}(\bar{X})=\sum_{g\in\mathbb{Z}_2^k}(a_g \bmod 2)g. \tag{2}$$

Thus, it is enough to consider the parity of the coefficient of v_0 in $P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})$. Moving to $\mathbb{Z}[\mathbb{Z}_2^k]$ allows us working with the matrix representation of $P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})$, which is given by

$$\rho(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X}))=\sum_{g\in\mathbb{Z}_2^k} a_g\rho(g).$$

For each $g \in \mathbb{Z}_2^k$, $\rho(g)$ is a permutation matrix of dimension 2^k with zeros in the diagonal, with the exception of the identity v_0 of $\mathbb{Z}_2^k$, for which $\rho(v_0)$ is the identity matrix. Hence all the diagonal entries of $\rho(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X}))$ are equal. This, in combination with equality 2 implies that

$$\begin{aligned}
P_{\mathbb{Z}_2[\mathbb{Z}_2^k]}(\bar{X})=0 &\Rightarrow trace(\rho(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})))=0 \bmod 2^{k+1}\\
P_{\mathbb{Z}_2[\mathbb{Z}_2^k]}(\bar{X})=J &\Rightarrow trace(\rho(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X})))=2^k \bmod 2^{k+1}.
\end{aligned}$$

Now instead of evaluating P at $x_i \leftarrow (v_0+v_i)$ over $\mathbb{Z}[\mathbb{Z}_2^k]$, we can equivalently evaluate it at $x_i \leftarrow \rho(v_0+v_i)$ over the isomorphic matrix algebra $\mathcal{M}=\rho(\mathbb{Z}[\mathbb{Z}_2^k])$, and then compute (modulo 2^{k+1})

$$trace(P_{\mathcal{M}}(\rho(\bar{X}))) = trace(\rho(P_{\mathbb{Z}[\mathbb{Z}_2^k]}(\bar{X}))).$$

By the representation theory of $\mathbb{Z}_2^k$, there is a matrix U of dimension 2^k such that for all $v \in \mathbb{Z}_2^k$, $\rho(V) = U\Lambda_v U^{-1}$, where Λ_v is a diagonal matrix with the eigenvalues of $\rho(v)$. Let Λ_i denote the diagonal matrix containing the eigenvalues of $\rho(v_0 + v_i)$ and $\bar{\Lambda}$ denote the assignment $x_i \leftarrow \Lambda_i$. Let $\Lambda_{i,j}$ denote the j^{th} diagonal entry of Λ_i, and $\bar{\Lambda}_j$ denote the assignment $x_i \leftarrow \Lambda_{i,j}$. Using the well known relationship of the trace with the eigenvalues, we have (taking all quantities modulo 2^{k+1})

$$trace(P_{\mathcal{M}}(\rho(\bar{X}))) = trace(UP_{\mathcal{M}}(\bar{\Lambda})U^{-1}) = trace(P_{\mathcal{M}}(\bar{\Lambda})) = \sum_{j=1}^{2^k} P_{\mathbb{Z}_{2^{k+1}}}(\bar{\Lambda}_j).$$

If b_j is the k-bit binary form of j, we can fix a U so that $\Lambda_{i,j} = 1 + (-1)^{v_i^T b_{j-1}}$ [10]. This completes the proof for the equivalence of options 1 and 2.

We have reduced the original problem to 2^k evaluations of P and the summation of the outputs over $\mathbb{Z}_{2^{k+1}}$. The 2^k evaluations can be performed sequentially, re-using the space, while the output sum is updated. The algorithm needs to maintain in the memory the assignment $\bar{X}$ which takes space $O(kn)$. For each j, the algorithm computes the input $\bar{\Lambda}_j$ in $O(nk)$ time. The evaluation of $P(\bar{\Lambda}_j)$ can be done in time $O(t)$, and space $O(nk + s)$, by assumption. Hence the total time is $O((nk + t)2^k)$ and the space requirement is $O(nk + s)$. □

Remark. If the smallest multilinear term in P has degree $k - j$, we can consider $P_j = (y_1 \dots y_j)P$. By Lemma 1, any term of degree greater than k always evaluates to 0, so running **decide-multilinear** on P' has the same effect as running it with the assumed restriction. We omit the details to the full paper.

3 Reducing m-Set k-Packing to Multilinear mk-Term

Let us start with a formal definition of the set packing problem.

Definition 2. *The m-SET k-PACKING problem (decision and search): Given a collection C of N sets, each containing m elements from a universe U of n elements, decide whether there is a collection $C' \subseteq C$ of k mutually-disjoint sets. If yes, find such a collection C'.*

The main result of this Section is the construction of a family $P_{\mathcal{A}}(X)$ of $2^{|\mathcal{A}|}$ packing-encoding polynomials, parameterized by a set $\mathcal{A}$ of variables taking binary values.

path-encoding polynomials: Given an instance I of the m-SET k- PACKING problem: (i) assign variables $X = \{x_i\}$, $i = 1, \dots, n$ to the elements of U, (ii) assign to the set $S_i \in C$ the degree m *set-monomial* Y_i, defined as the product of the variables corresponding to the elements of S_i. (iii) Let $\mathcal{A} = \{a_{i,j} : i \in [1, k], j \in [1, N]\}$ and define $P_{\mathcal{A}}(X) = \prod_{i=1}^{k} \left(\sum_{j=1}^{N} a_{i,j} Y_j\right)$.

Theorem 3. *Let $P \in \mathbb{Z}_2[X]$ be a polynomial picked uniformly at random from $P_{\mathcal{A}}(X)$ by letting $Prob(a_{i,j} = 0) = Prob(a_{i,j} = 1) = 1/2$, independently for all i, j. If I is a "yes" instance of the m-set k-packing problem, the polynomial $P \in \mathbb{Z}_2[X]$ has a multilinear term of degree mk with probability at least $1/4$. Otherwise, P has no multilinear terms.*

Proof. It is clear that P can be expanded to a sum of what we will call *set-products*, each of which is a product of k set-variables Y_i. Each set-product is itself a monomial and has total degree mk. If a set-product is a multiple of Y_iY_j for two intersecting sets S_i and S_j, then by construction it is not multilinear. It follows that if I does not contain a k-set packing, P has no multilinear terms.

Claim A: The coefficient $cf(Y_1 \dots Y_k)$ (and thus of any product of k distinct set variables) in P is odd with probability at least $1/4$, with respect to the random assignment to the coefficients $a_{i,j}$.

To prove Claim A consider the $k \times k$ matrix $M_{i,j} = a_{i,j}$. Let S_k denote the set of all permutations of $1, \dots, k$, and $perm(M), det(M)$ denote the permanent and the determinant of M. We have

$$cf(Y_1 \dots Y_k) = \sum_{\pi \in S_k} \prod_{i=1}^{k} a_{i,\pi(i)} = perm(M) = det(M) \ (mod\ 2).$$

By standard linear algebra facts $det(M)\ mod\ 2 = 0$ if and only if the columns of M are linearly dependent over $\mathbb{Z}_2$. However, the columns of M are random vectors picked independently and uniformly from $\mathbb{Z}_2^k$. In the proof of Theorem 1 (base case of induction), we showed that the probability that they are linearly independent is at least $1/4$. This finishes the proof of Claim A •

Assume now that I has at least one k-set packing, and fix one. Clearly its set-product is a multilinear term T of degree mk. The same multilinear term T can be generated by a collection C_T of distinct set-products, corresponding to different k-set packings that cover the same subset of U. It is then enough to show that the probability (with respect to the random assignment to the coefficients $a_{i,j}$) that C_T generates an odd number of copies of T is at least $1/4$. Let us denote this probability by $Prob(C_T \rightsquigarrow 1)$.

Claim B: For all $C' \subseteq C_T$, there is a $C'' \subset C'$ such that

$$Prob(C'' \rightsquigarrow 1) \leq Prob(C' \rightsquigarrow 1).$$

Let $\mathcal{Y} = \{Y_1, \dots, Y_N\}$ be the set of set-variables. Let C_Z be a collection of set-products that generate the same multilinear term T and share the common factor $Z \in \mathcal{Y}$. Let $\mathcal{A}_S$ denote the set of assignments to the coefficients $a_{i,j}$ corresponding to the variables of $S \subseteq \mathcal{Y}$ in the factors of P. Now fix an assignment $A \in \mathcal{A}_{\mathcal{Y}-Z}$ and let $a_{i,Z}$ denote the coefficients that multiply Z in the factors of P. Considering P as a function of $a_{i,j}$, its partial evaluation at A is always of the form

$$P(A) = R + T\left(\sum_{i=1}^{k} a_{i,Z} b_i\right)$$

where R is a sum of terms different than T, and $b_i = 0$ or $b_i = 1$ depending only on A. It can be seen that there are two cases: (a) for all i, we have $b_i = 0$ in which case for all assignments in $\mathcal{A}_Z$ the polynomial $P(A)$ does not contain T, (b) there is at least one j for which $b_j = 1$. In this case we say that C_Z is *Z-dependent* under the assignment A. Then, it is not hard to see that

$$Prob_{\mathcal{A}_Z}(C_Z \rightsquigarrow 1/C_Z \textit{ is } Z\textit{-dependent}) = 1/2. \tag{3}$$

where as indicated by the notation the probability is taken with respect to the assignments in $\mathcal{A}_Z$. Using the same notation, it follows that

$$Prob_{\mathcal{A}}(C_Z \rightsquigarrow 1) = Prob_{\mathcal{A}}(C_Z \rightsquigarrow 0) = (1/2)Prob_{\mathcal{A}_{Y-Z}}(C_Z \textit{ is } Z\textit{-dependent}) \tag{4}$$

We are now ready to move to the main part of the proof of Claim B. In the following, the probability subscripts are implied by the context, and we will drop them for simplicity. Let C' be an arbitrary subset of C_T. Clearly, unless $|C'| = 1$, there is $Z \in \mathcal{Y}$ such that $C' = C_Z \cup C_{\bar{Z}}$ where C_Z contains all set-products in C' that are multiple of a common factor of Z and $C_{\bar{Z}} = C' - C_Z \neq \emptyset$. Considering all possibilities, and appropriately using equalities 3 and 4, we have

$$\begin{aligned} Prob(C' \rightsquigarrow 1) = {} & Prob(C_{\bar{Z}} \rightsquigarrow 1)Prob(C_Z \textit{ is not } Z\textit{-dependent}/C_{\bar{Z}} \rightsquigarrow 1) + \\ & Prob(C_{\bar{Z}} \rightsquigarrow 1)Prob(C_Z \textit{ is } Z\textit{-dependent}/C_{\bar{Z}} \rightsquigarrow 1)(1/2) + \\ & Prob(C_{\bar{Z}} \rightsquigarrow 0)Prob(C_Z \textit{ is } Z\textit{-dependent}/C_{\bar{Z}} \rightsquigarrow 0)(1/2) \\ \geq {} & (1/2)Prob(C_Z \textit{ is } Z\textit{-dependent}) = Prob(C_Z \rightsquigarrow 1). \end{aligned}$$

The inequality came from dropping the first term and combining the remaining two. The set C_Z is the set C'' stated in Claim B. This completes the proof for Claim B •

Finally we observe that Claim B can be used repeatedly to show that there is a chain $C_T \supset C_1 \ldots \supset C_\nu$, where $|C\nu| = 1$, such that $Prob(C_T \rightsquigarrow 1) \geq Prob(C_1 \rightsquigarrow 1) \geq \ldots \geq Prob(C_\nu \rightsquigarrow 1)$. Claim A gives that $Prob(C_\nu \rightsquigarrow 1) \geq 1/4$ and the proof is completed. □

Theorem 4. *There is an $O^*(2^{km}(km)^2N)$ time algorithm such that, on a instance I of the m-set k-packing problem, its output is "yes" with probability at least $1/16$ only if I contains a k-packing. In a "yes" instance, a k-packing can be found with constant probability by $O(N \log N)$ calls to the decision algorithm. Both the decision and the search algorithms use $O(kn)$ space.*

Proof. A random polynomial P from the family of packing-encoding polynomials $P_{\mathcal{A}}(X)$ can be constructed in $O(kN)$ time. The polynomial P can be evaluated over $\mathbb{Z}_2^{km+1}$ with $O(kmN)$ addition and multiplication operations involving $(km+1)$-bits numbers, each of which takes $O^*(km)$ time. It is clear that $O(kn)$ space is required to store an assignment over $\mathbb{Z}_2^k$ to the variables, and it is not hard to see that P can be evaluated in $O(kn)$ space over $\mathbb{Z}_2^k$ because apart from the assignment to the variables, only three values must be kept around at any given time. The proof follows by applying Theorem 2. The details of the search

algorithm can be found in [7]. The key observation is that if $C' \subseteq C$ contains a k-packing, then $O(\log N)$ calls of the decision algorithm on $C' - Z$ decide with probability $1-1/N$ whether $Z \in C'$ is contained in all k-packings of C' or $C'-Z$ contains a k-packing. Starting from C, the algorithm applies this procedure at most N times to find the k-packing with probability $(1-1/N)^N > 1/4$. □

4 Reducing k-Simple Path to Multilinear $3k/2$-Term

We start with a definition of the problem. We will work with directed graphs; if the graph is undirected we can see it as a directed graph in the obvious way.

Definition 3. The k-PATH problem (decision and search): *Given a graph G with n vertices V and m directed edges E, decide whether the graph contains a path of length $k-1$ which connects k distinct vertices. If yes, find such a path.*

Let $\mathcal{P}_{t,v}$ denote the set of directed paths of length t ending in $v \in V$. Let $X = \{x_{v1}, \ldots, x_{v_n}\}$ be a set of n variables corresponding to the vertices of G, and $Y = \cup_{t=1}^{k} \cup_{(v_i,v_j)\in E} y_{v_i,v_j,t}$ be a set of nk variables where each (directed) edge corresponds to k variables; the k different copies are intended to encode the position of the edge along a path of length k. If $p = (v_1, v_2, \ldots, v_t)$ is a path of length $t-1$ we define its encoding to be

$$enc(p) = \left(\prod_{i=1}^{t} x_{v_i}\right)\left(\prod_{j=1}^{\lfloor t/2 \rfloor} y_{v_{2j-1},v_{2j},2j-1}\right)$$

and the t-path encoding polynomial for $v \in V$ as $P(t,v) = \sum_{p\in\mathcal{P}_{t,v}} enc(p)$.

Lemma 3. *For a path p, the encoding $enc(p)$ is multilinear if and only if p is simple. If p_1, p_2 are two distinct simple paths then $enc(p_1) \neq enc(p_2)$. Hence, the path encoding polynomial $P_k = \sum_{v\in V} P(k,v)$ contains a multilinear term of degree $k + \lfloor k/2 \rfloor$ if and only if G contains a simple path k.*

Proof. If p is not a simple path then clearly the first factor in $enc(p)$ contains a squared variable. If p is simple, then all the vertices and edges it uses are distinct so $enc(p)$ is multilinear. Now notice that a directed path p is completely determined by: (i) its sink vertex, (ii) the list of the directed edges whose source is at even distance from the first vertex on the path, along with their positions in p. This list is completely determined by the second factor in $p(e)$, while if the sink vertex is missing in the list (when p has an even number of edges) it can be recovered from the first factor in $p(e)$. Hence every simple k-path ending in v is mapped to a distinct (thus with a unit coefficient) multilinear monomial in P_k, and the monomial has degree $k + \lfloor k/2 \rfloor$. □

We now describe a circuit for the computation of P_k. By convention we define $P(0,v) = x_v$. Assume we have constructed a circuit for $P(t-1,v)$ for all $v \in V$. It is not hard to see that the circuit for $P(t,v)$ can be constructed as follows:

$$P(t,v) = \sum_{(u,v)\in E} x_v (y_{u,v,t})^{(t \bmod 2)} P(t-1,u).$$

Theorem 5. *There is an $O^*(2^{3k/2}k^2m)$ time algorithm such that, on a graph G, it returns "yes" with probability at least $1/4$ only if the graph contains a simple k-path. In a "yes" instance, a simple k-path can be found with $O^*(n+\min(k^2,m))$ applications of the decision algorithm. Both the decision and the search algorithm use $O(k^2m)$ space.*

Proof. The algorithm runs **decide-multilinear** on $P_k = \sum_{v\in V} P(k,v)$. Given the values for $P(t-1,v)$, the values $P(t,v)$ can be computed with $O(m)$ additions and multiplications in $\mathbb{Z}_2^{k+1}$. Hence the polynomial $P(k,v)$ can be evaluated over $\mathbb{Z}_2^{k+1}$ with $O(km)$ operations taking $O^*(k)$ time each. For all t the circuit needs to remember $((k/2)m+n)$ variable values and the n values of $P(t-1,v)$, taking space $O(k^2m)$. The claim follows by applying Theorem 2. The algorithm and proof for the search version of the problem is similar to those for the search version of the m-set k-packing. Roughly, the algorithm will keep removing vertices until it is left with an induced k-subgraph which still contains a k-path, and then it will remove up to $O(k^2)$ edges until it is left with a k-path. □

Remark 1. Assume that there is a subset $V' \subseteq V$ with $|V'| = k$ such that the number $n_{V'}$ of (directed) Hamiltonian paths in the graph induced by V' is odd. If we set the edge variables to $Y = 1$ in the definitions of the path encodings, it is not hard to see that the coefficient of $\prod_{v\in V'} x_v$ in $P_k(Y=1)$ is equal to $n_{V'}$.This observation can be used to reduce the time complexity to $O^*(2^k k^2 m)$ and the space to $O(kn+m)$.

5 Open Questions

It is a possibility that the algorithms in this paper can be derandomized. A basic step seems to be the construction of a family of assignments $\mathcal{F} : X \to \mathbb{Z}_2^k$, such that for each k-subset X' of X, there is an assignment in $\mathcal{F}$ under which the vectors in X' are linearly independent over $\mathbb{Z}_2$. Since a random assignment satisfies this with probability at least $1/4$, it is a plausible conjecture that there is a polynomial size $\mathcal{F}$ with the required property.

References

1. Alon, N., Yuster, R., Zwick, U.: Color coding. Journal of the ACM 42(4), 844–856 (1995)
2. Chen, J., Lu, S., Sze, S.-H., Zhang, F.: Improved algorithms for path, matching, and packing problems. In: SODA 2007: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics, pp. 298–307 (2007)
3. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
4. Fellows, M.R., Knauer, C., Nishimura, N., Ragde, P., Rosamond, F.A., Stege, U., Thilikos, D.M., Whitesides, S.: Faster fixed-parameter tractable algorithms for matching and packing problems. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 311–322. Springer, Heidelberg (2004)

5. Jia, W., Zhang, C., Chen, J.: An efficient parameterized algorithm for m-set packing. J. Algorithms 50(1), 106–117 (2004)
6. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Divide-and-color. In: WG: Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, pp. 58–67 (2006)
7. Koutis, I.: A faster parameterized algorithm for set packing. Information Processing Letters 94(1), 4–7 (2005)
8. Liu, Y., Lu, S., Chen, J., Sze, S.-H.: Greedy localization and color-coding: Improved matching and packing algorithms. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 84–95. Springer, Heidelberg (2006)
9. Papadimitriou, C.H., Yannakakis, M.: On limited nondeterminism and the complexity of the V-C dimension. J. Comput. Syst. Sci. 53(2), 161–170 (1996)
10. Terras, A.: Fourier Analysis on Finite Groups and Applications. Cambridge University, Cambridge (1999)
11. Valiant, L.G.: Why is boolean complexity difficult? In: Boolean Function Complexity. Lond. Math. Soc. Lecure Note Ser, vol. 169

Appendix: Group Algebras of $\mathbb{Z}_2^k$ and Their Representation

Let $\mathbb{Z}_2^k$ be the group consisting of k-dimensional 0-1 vectors with the group multiplication being entry-wise addition modulo 2. The group algebra $K[\mathbb{Z}_2^k]$, where K is a field, is the set of all linear combinations of the form $\sum_{v\in\mathbb{Z}_2^k} a_v v$ where $a_v \in K$. The addition operator of $K[\mathbb{Z}_2^k]$ is defined by $\sum_{v\in\mathbb{Z}_2^k} a_v v + \sum_{v\in\mathbb{Z}_2^k} b_v v = \sum_{v\in\mathbb{Z}_2^k}(a_v + b_v)v$. Multiplication by a scalar $\alpha \in K$ is defined by $\alpha\sum_{v\in\mathbb{Z}_2^k} a_v v = \sum_{v\in\mathbb{Z}_2^k}(\alpha a_v)v$. The multiplication operator of $K[\mathbb{Z}_2^k]$ is defined by $\left(\sum_{v\in\mathbb{Z}_2^k} a_v v\right)\left(\sum_{u\in\mathbb{Z}_2^k} b_u u\right) = \sum_{v,u\in\mathbb{Z}_2^k}(a_v b_u)(uv)$. It can be verified that $K[\mathbb{Z}_2^k]$ is commutative. In the particular case of $\mathbb{Z}_2[\mathbb{Z}_2^k]$, an element can also be seen as a subset of $\mathbb{Z}_2^k$.

Matrices of dimension d with integer entries form a group $M^{d\times d}$ with matrix multiplication and an algebra $\mathcal{M}^{d\times d}$ with matrix addition, multiplication by a scalar, and matrix multiplication. It is well known ([10], p. 172), that there is a one-to-one map $\rho : \mathbb{Z}_2^k \to M^{2^k\times 2^k}$ such that $\rho(uv) = \rho(u)\rho(v)$. The map ρ is an isomorphism. The map ρ can be extended to a one-to-one map of $\mathbb{Z}[\mathbb{Z}_2^k]$ to $\mathcal{M}^{2^k\times 2^k}$, as follows: $\rho(\sum_{v\in\mathbb{Z}_2^k} a_v v) = \sum_{v\in\mathbb{Z}_2^k} a_v\rho(v)$. It can be verified that if w_1, w_2 are elements of $\mathbb{Z}[\mathbb{Z}_2^k]$ and $\alpha \in \mathbb{Z}$, we have $\rho(w_1 + w_2) = \rho(w_1) + \rho(w_2)$, $\rho(w_1 w_2) = \rho(w_1)\rho(w_2)$ and $\rho(\alpha w_1) = \alpha\rho(w_1)$. Hence, the map ρ defines an isomorphic matrix algebra which we will denote by $\rho(\mathbb{Z}[\mathbb{Z}_2^k])$.

The matrices $\rho(\mathbb{Z}_2^k)$ are simultaneously diagonalizable, i.e. there is a matrix U such that for all $v \in \mathbb{Z}_2^k$, we have $\rho(v) = U^{-1}\Lambda_v U$, where Λ_v are the eigenvalues of $\rho(v)$, also known as the characters of v. If $b(i)$ is the vector containing the k-bit binary form of i, the i^{th} eigenvalue of $\rho(v)$ is given by $(-1)^{v^T b(i)}$ [10].

Understanding the Complexity of Induced Subgraph Isomorphisms

Yijia Chen[1], Marc Thurley[2], and Mark Weyer[2]

[1] BASICS, Department of Computer Science, Shanghai Jiaotong University, 200030 Shanghai, China
[2] Institut für Informatik, Humboldt-Universität zu Berlin, 10099 Berlin, Germany

Abstract. We study left-hand side restrictions of the *induced subgraph isomorphism* problem: Fixing a class $\mathcal{C}$, for given graphs $G \in \mathcal{C}$ and arbitrary H we ask for induced subgraphs of H isomorphic to G.

For the homomorphism problem this kind of restriction has been studied by Grohe and Dalmau, Kolaitis and Vardi for the decision problem and by Dalmau and Jonsson for its counting variant.

We give a dichotomy result for both variants of the induced subgraph isomorphism problem. Under some assumption from parameterized complexity theory, these problems are solvable in polynomial time if and only if $\mathcal{C}$ contains no arbitrarily large graphs.

All classifications are given by means of parameterized complexity. The results are presented for arbitrary structures of bounded arity which implies, for example, analogous results for directed graphs.

Furthermore, we show that no such dichotomy is possible in the sense of classical complexity. That is, if P $\neq$ NP there are classes $\mathcal{C}$ such that the induced subgraph isomorphism problem on $\mathcal{C}$ is neither in P nor NP-complete. This argument may be of independent interest, because it is applicable to various parameterized problems.

1 Introduction

Given graphs G and H, the *induced subgraph isomorphism problem* asks for the existence of induced subgraphs of H isomorphic to G. A wide variety of graph theoretic problems can be formulated in this way, as is the case for the induced path or induced cycle problem. By the fact that the independent set problem is also an induced subgraph isomorphism problem, the latter is obviously NP-complete.

This inherent intractability is highly unsatisfactory, as it does not give any insight into the complexity of more restricted subproblems such as e.g. the induced path problem, necessitating separate investigation [3]. To uniformly study the complexity of such subproblems of the induced subgraph isomorphism problem, we therefore consider restrictions of this problem in the following way. Fixing a class $\mathcal{C}$ of graphs, we consider only inputs $G \in \mathcal{C}$ whereas H is still an arbitrary graph.

For the related homomorphism problem, the complexity of this kind of restrictions has been described in [5,10] in terms of a dichotomy: If $\mathcal{C}$ has bounded

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 587–596, 2008.

treewidth up to homomorphic equivalence, the homomorphism problem is in polynomial time, otherwise it is intractable in the sense of parameterized complexity. A similar dichotomy from [4] states that for the counting version polynomial time is equivalent to bounded treewidth.

In this paper we settle the complexity of the restricted induced subgraph isomorphism problem by giving a further dichotomy. For both the decision and the counting variant, we show that the problem is computable in polynomial time if and only if there is an absolute bound on the size of the graphs in the class $\mathcal{C}$. Otherwise, the problem is intractable in terms of parameterized complexity.

We give two different proofs for the two versions of this problem. We prove the dichotomy for the decision problem based on the result from [10]. For the counting version, we give a more direct proof using an inclusion-exclusion style argument. This cannot be applied to the decision case.

Furthermore, all proofs will be given for arbitrary structures of bounded arity. Therefore, our results are not only applicable to graphs. They also extend to analogous results for the induced subgraph isomorphism problems on e.g. directed graphs, on coloured graphs, and on hypergraphs with bounded edge-size.

The fact that our hardness results rely on parameterized complexity theory raises the question of whether a similar dichotomy in terms of classical complexity could possibly be established. Using a Ladner-style argument (compare [11]) we show that this is not the case, unless P = NP (or FP = #P for the counting problem, resp.). More precisely, there are classes $\mathcal{C}$ such that the restricted induced subgraph isomorphism problem is neither in P nor NP-complete. This result is presented in a universal way which enables us to derive analogs for e.g. the homomorphism problem and the corresponding counting problems. Note that, for the homomorphism problem itself, a similar result has also been shown independently by [2].

Due to space limitations we have to defer some proofs to the full version of the paper.

2 Preliminaries

Structures. We only consider relational vocabularies. Hence, a *vocabulary* is a set of relational symbols, each having an arity in $\mathbb{N}$. The arity of the symbol R is denoted $\mathrm{ar}(R)$. The arity of a vocabulary is the maximal arity of its symbols. Let τ be a vocabulary. A *structure* $\mathfrak{A}$ of vocabulary τ, or *τ-structure* for short, is a tuple $(A, (R^{\mathfrak{A}})_{R\in\tau})$, where the *universe* A of $\mathfrak{A}$ is some set and $R^{\mathfrak{A}} \subseteq A^{\mathrm{ar}(R)}$ for all $R \in \tau$. As we have done here, whenever we denote a structure by a German type letter, its universe is implicitly denoted by the corresponding Roman type letter. For algorithmic purposes, all vocabularies and universes are finite. The arity of a structure is the arity of its vocabulary.

Let $\mathfrak{A}$ and $\mathfrak{B}$ be structures of the same vocabulary τ. $\mathfrak{A}$ is a *substructure* of $\mathfrak{B}$, if $A \subseteq B$ and $R^{\mathfrak{A}} \subseteq R^{\mathfrak{B}}$ for all $R \in \tau$. $\mathfrak{A}$ is an *induced substructure* of $\mathfrak{B}$, if furthermore $R^{\mathfrak{A}} = R^{\mathfrak{B}} \cap A^{\mathrm{ar}(R)}$ for all $R \in \tau$. A *homomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ is a function $f : A \to B$ such that for all $R \in \tau$, we have $f(R^{\mathfrak{A}}) \subseteq R^{\mathfrak{B}}$.

An *embedding* is a homomorphism that is injective. A *strong embedding* is an embedding f such that $f(R^{\mathfrak{A}}) = R^{\mathfrak{B}} \cap f(A)^{\mathrm{ar}(R)}$ for all $R \in \tau$. Note that (strong) embeddings coincide with isomorphisms to (induced) substructures.

For a structure $\mathfrak{A}$, say of vocabulary τ, the *Gaifman graph* $G(\mathfrak{A})$ of $\mathfrak{A}$ is the graph with vertex set A such that there is an edge between a and a' for $a \neq a'$ if and only if there is some $R \in \tau$, say of arity r, some $(a_1, \ldots, a_r) \in R^{\mathfrak{A}}$, and some $1 \leq i, j \leq r$ such that $a = a_i$ and $a' = a_j$.

Parameterized Complexity. Let Σ be a finite alphabet. A *parameterization of* Σ is a polynomial time computable mapping $\kappa : \Sigma^* \to \mathbb{N}$. A *parameterized decision problem* is a pair (P, κ) with $P \subseteq \Sigma^*$ and κ a parameterization. Similarly, a *parameterized counting problem* is a pair (F, κ) with $F : \Sigma^* \to \mathbb{N}$ and κ a parameterization. For problem instances $x \in \Sigma^*$, the value $\kappa(x)$ is called the *parameter* of x.

An algorithm is a *fixed-parameter algorithm*, if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that for all $x \in \Sigma^*$ the algorithm stops after at most $f(\kappa(x)) \cdot |x|^c$ steps. A parameterized decision problem (P, κ) is *fixed-parameter tractable*, if there is a fixed-parameter algorithm which, for all $x \in \Sigma^*$, decides if $x \in P$. The class of all such problems is denoted by FPT. *Fixed-parameter tractable parameterized counting problems* are defined analogously, with fixed-parameter algorithms computing $F(x)$ and FFPT being the class of all of these problems.

In all well-behaved cases, our hardness results hold for the usual strongly uniform reductions. In general, however, we need nonuniform reductions for technical reasons. Therefore we define both versions, starting with the nonuniform variants. Note that FPT and FFPT are not closed under these, nor do we need them to be.

An FPT *many-one reduction* from a parameterized problem (P, κ) to a parameterized problem (P', κ') with $P \subseteq \Sigma^*$ and $P' \subseteq (\Sigma')^*$ is a family $(f_i : \Sigma^* \to (\Sigma')^*)_{i \in \mathbb{N}}$ with the following properties: There is a $c \in \mathbb{N}$ and some $h : \mathbb{N} \to \mathbb{N}$ such that on inputs of length n, every f_i is computable in time $h(i) \cdot n^c$. Furthermore there is some $g : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$ and $y := f_{\kappa(x)}(x)$, we have $x \in P$ if and only if $y \in P'$ and $\kappa'(y) \leq g(\kappa(x))$.

The corresponding reductions for counting problems are defined similarly. Let (F, κ) and (F', κ') be parameterized counting problems with $F : \Sigma^* \to \mathbb{N}$ and $F' : (\Sigma')^* \to \mathbb{N}$. An FPT *parsimonious reduction* from (F, κ) to (F', κ') is a family $(f_i : \Sigma^* \to (\Sigma')^*)_{i \in \mathbb{N}}$ such that for a fixed $c \in \mathbb{N}$ and $h : \mathbb{N} \to \mathbb{N}$, on inputs of length n, every f_i is computable in time $h(i) \cdot n^c$. Furthermore there is some $g : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$ and $y := f_{\kappa(x)}(x)$ we have $F(x) = F'(y)$ and $\kappa'(y) \leq g(\kappa(x))$.

To develop our results, we need a second notion of reductions between counting problems. With (F, κ) and (F', κ') as above, an FPT *Turing reduction* from (F, κ) to (F', κ') is a family of functions $(f_i : \Sigma^* \to \mathbb{N})_{i \in \mathbb{N}}$ with the following properties. For all $x \in \Sigma^*$ we have $F(x) = f_{\kappa(x)}(x)$. Furthermore, there are $g, h : \mathbb{N} \to \mathbb{N}$ and $c \in \mathbb{N}$ such that on inputs on length n, every f_i can be

computed in time $h(i) \cdot n^c$ by an algorithm with oracle access to F' such that every oracle query $F'(y)$ satisfies $\kappa'(y) \leq g(i)$.

The above reductions become *strongly uniform* if we further stipulate that the given families and the mappings $g, h : \mathbb{N} \to \mathbb{N}$ be computable.

Downey and Fellows [6] defined a hierarchy of complexity classes of decision problems FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \ldots$, conjecturing that all of the given inclusions are strict. Analogs of this hierarchy in terms of counting problems have been proposed in [12] and, in slightly different form, in [8]. The differences between these definitions do not affect the aim of our paper. For our purposes, we rely on [8] which define a hierarchy FFPT $\subseteq$ #W[1] $\subseteq$ #W[2] $\subseteq \ldots$ with the same conjecture as before, that all inclusions are strict. We will be concerned only with the first level W[1] (#W[1], respectively) of these hierarchies.

Usually, these classes are defined such that they are closed under strongly uniform reductions. Nonuniform versions are immediate, however. For ease of presentation, we will denote both versions by W[1] respectively #W[1]. We will rely on two results, namely Theorem 7 explained below and the following.

Theorem 1 (Flum and Grohe [8]). *p*-#Clique *is* #W[1]*-complete under strongly uniform* FPT *parsimonious reductions, where p-*#Clique *is the classical* Clique *problem parameterized by the size of the clique we are looking for.*

As each strongly uniform reduction is nonuniform, this gives hardness for both variants of #W[1].

For a class $\mathcal{C}$ of structures, the problem Hom($\mathcal{C}$) asks, when given a structure $\mathfrak{A} \in \mathcal{C}$ and an arbitrary structure $\mathfrak{B}$, whether there is a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. Similarly, we define restrictions of the problems Emb and StrEmb asking, whether a structure $\mathfrak{A}$ is isomorphic to a substructure, respectively an induced substructure, of a structure $\mathfrak{B}$. In some contexts these problems are called the *embedding* and *strong embedding* problem – hence the abbreviations Emb and StrEmb.

Further, #Hom, #Emb, #StrEmb are their counting analogs, which ask for the number of such homomorphisms or isomorphisms, and *p*-Hom, *p*-Emb, *p*-StrEmb, *p*-#Hom, *p*-#Emb, and *p*-#StrEmb are the parameterized versions, where the parameter is $|\mathfrak{A}|$. Note that the membership of all of these problems in W[1] (#W[1], respectively) is well-known. This follows, for example, from [7] and [8].

3 The Dichotomies

Throughout this paper we assume that $\mathcal{C}$ is a class of structures of bounded arity, i.e., there is a bound r_0 such that no structure in $\mathcal{C}$ has arity beyond r_0.

Furthermore, we say that $\mathcal{C}$ is *meagre*, if there is some $n_0 \in \mathbb{N}$ such that for all $\mathfrak{A} \in \mathcal{C}$ of arity at least 2 we have $|A| \leq n_0$.

Theorem 2 (*p*-StrEmb($\cdot$) Dichotomy). *Let $\mathcal{C}$ be a class of structures of bounded arity.*

If $\mathcal{C}$ is meagre, then $\mathrm{StrEmb}(\mathcal{C}) \in \mathrm{P}$. *Otherwise, p-*$\mathrm{StrEmb}(\mathcal{C})$ *is complete for* $\mathrm{W}[1]$ *using nonuniform* FPT *many-one reductions.*

If $\mathcal{C}$ is recursively enumerable, then $\mathrm{W}[1]$*-completeness holds even for strongly uniform* FPT *many-one reductions.*

Theorem 3 (*p*-#StrEmb$(\cdot)$ Dichotomy). *Let $\mathcal{C}$ be a class of structures of bounded arity.*

If $\mathcal{C}$ is meagre, then $\#\mathrm{StrEmb}(\mathcal{C}) \in \mathrm{FP}$. *Otherwise, p-*$\#\mathrm{StrEmb}(\mathcal{C})$ *is complete for* $\#\mathrm{W}[1]$ *using nonuniform* FPT *Turing reductions.*

If $\mathcal{C}$ is recursively enumerable, then $\#\mathrm{W}[1]$*-completeness holds even for strongly uniform* FPT *Turing reductions.*

For membership in P or FP, we view our problems as promise problems, unless $\mathcal{C}$ happens to be decidable in polynomial time. When we consider classes of graphs instead of classes of arbitrary structures, then arity 2 is guaranteed. Hence meagreness just means bounded size and the theorems read as follows:

Corollary 4. *Let $\mathcal{C}$ be a class of graphs.*

If the graphs in $\mathcal{C}$ have bounded size, then $\mathrm{StrEmb}(\mathcal{C}) \in \mathrm{P}$. *Otherwise, p-*$\mathrm{StrEmb}(\mathcal{C})$ *is complete for* $\mathrm{W}[1]$ *under FPT many-one reductions.*

The analogue holds for the counting problem.

The first parts of Theorem 2 and Theorem 3 are easy:

Lemma 5. *If $\mathcal{C}$ is meagre, then* $\#\mathrm{StrEmb}(\mathcal{C}) \in \mathrm{FP}$ *and* $\mathrm{StrEmb}(\mathcal{C}) \in \mathrm{P}$.

Hence, in the following we may assume that $\mathcal{C}$ contains arbitrarily large structures of arity at least 2.

Roughly speaking, we want to find structures in $\mathcal{C}$ which exhibit large cliques. We can do this only up to taking complements. So, for a τ-structure $\mathfrak{A}$, define the complement $\mathfrak{A}^{\mathrm{comp}}$ of $\mathfrak{A}$ as the following τ-structure: The universe is again A, and for each relational symbol $R \in \tau$, say of arity r, we have $R^{\mathfrak{A}^{\mathrm{comp}}} = A^r \setminus R^{\mathfrak{A}}$. For a class $\mathcal{C}$ of structures, let $\mathcal{C}^{\mathrm{comp}} := \{\mathfrak{A}^{\mathrm{comp}} \mid \mathfrak{A} \in \mathcal{C}\}$ be the class of complements of structures in $\mathcal{C}$. Note that this is not the complement of $\mathcal{C}$.

The following lemma is immediate.

Lemma 6. *p-*$\mathrm{StrEmb}(\mathcal{C}) \equiv^{\mathrm{FPT}}$ *p-*$\mathrm{StrEmb}(\mathcal{C}^{\mathrm{comp}})$ *by parsimonious reductions.*

For a structure $\mathfrak{A}$ and a symbol R in its vocabulary, say of arity $r \geq 2$, let

$$D(\mathfrak{A}, R) := (A, \{(a, b) \in A^2 \mid a \neq b, (a, \ldots, a, b) \in R^{\mathfrak{A}}\})$$

be the digraph associated with $\mathfrak{A}$ and R. For any given $k \in \mathbb{N}$ and sufficiently large A, Ramsey's Theorem guarantees that $D(\mathfrak{A}, R)$ contains a clique or a tournament or an independent set of size k. Then, at least one of $D(\mathfrak{A}, R)$ and $D(\mathfrak{A}^{\mathrm{comp}}, R)$ contains a clique or a tournament of size k. Hence, for at least one of $\mathcal{C}$ and $\mathcal{C}^{\mathrm{comp}}$ we can find arbitrarily large cliques or tournaments in the digraphs associated with its structures. Using Lemma 6, we can assume without loss of generality that this is the case for $\mathcal{C}$. Then, in particular, the Gaifman graphs of structures in $\mathcal{C}$ contain arbitrarily large cliques.

From this point on, the proofs for Theorem 2 and Theorem 3 diverge.

3.1 Hardness of Deciding

First to the proof of Theorem 2. We use $\leq^{\text{FPT}}$ to denote nonuniform FPT many-one reducibility in the general case. If $\mathcal{C}$ is recursively enumerable, we instead intend it to denote strongly uniform FPT many-one reducibility.

We base the hardness part of Theorem 2 on the following result:

Theorem 7 (Grohe [10]). *If $\mathcal{C}$ is a class of structures with* cores *of unbounded* treewidth*, then p-Hom($\mathcal{C}$) is hard for* W[1].

Some explanations are in order. Hardness uses, just as we need it, nonuniform FPT many-one reductions, which are strongly uniform in case $\mathcal{C}$ is recursively enumerable. As to the notions of cores and treewidth, let us omit the definitions and just state the two facts we actually need:

1. The core of a structure $\mathfrak{A}$ is some particular homomorphic image of $\mathfrak{A}$ in $\mathfrak{A}$.
2. If $G(\mathfrak{A})$ contains a clique of size k, then the treewidth of $\mathfrak{A}$ is at least $k-1$.

A relation $R \subseteq A^r$ is *antireflexive*, if for all $a \in A$ we have $(a,\ldots,a) \notin R$. A structure is *antireflexive*, if all its relations are. For a given structure $\mathfrak{A}$, the *antireflexive part* $\mathfrak{A}^{\text{antiref}}$ of $\mathfrak{A}$ is obtained from $\mathfrak{A}$ by deleting all tuples of the form $(a,\ldots,a)$ from all relations of $\mathfrak{A}$. Further, let $\mathcal{C}^{\text{antiref}} := \{\mathfrak{A}^{\text{antiref}} \mid \mathfrak{A} \in \mathcal{C}\}$. If $\mathcal{C}$ is recursively enumerable, then so is $\mathcal{C}^{\text{antiref}}$.

Lemma 8. p-Hom($\mathcal{C}^{\text{antiref}}$) $\leq^{\text{FPT}}$ p-StrEmb($\mathcal{C}$).

Proof: Assume given an input $(\mathfrak{A}', \mathfrak{B})$ to the reduction. Let $\mathfrak{A}$ be such, that $\mathfrak{A}' = \mathfrak{A}^{\text{antiref}}$. In case $\mathcal{C}$ is recursively enumerable, such an $\mathfrak{A}$ can be found effectively, otherwise there is no need for effectiveness because we use nonuniform reductions.

Define the structure $\mathfrak{C}$ as the following variant of $\mathfrak{A} \otimes \mathfrak{B}$: The universe is $C = A \times B$ and for each symbol R, say of arity r, let

$$R^{\mathfrak{C}} := \{((a_1,b_1),\ldots,(a_r,b_r) \mid (a_1,\ldots,a_r) \in R^{\mathfrak{A}}, (b_1,\ldots,b_r) \in R^{\mathfrak{B}}\}$$
$$\cup\, \{((a,b),\ldots,(a,b)) \mid (a,\ldots,a) \in R^{\mathfrak{A}}, b \in B\}.$$

Now if $f : A \to B$ is a homomorphism from $\mathfrak{A}'$ to $\mathfrak{B}$, then $g : A \to C$ defined by $g(a) = (a, f(a))$ is a strong embedding of $\mathfrak{A}$ in $\mathfrak{C}$. Conversely, if $g : A \to C$ is a strong embedding of $\mathfrak{A}$ in $\mathfrak{C}$, then the projection of g to B is a homomorphism from $\mathfrak{A}'$ to $\mathfrak{B}$.

Hence, the reduction outputs $(\mathfrak{A}, \mathfrak{C})$. □

Proof of Theorem 2: Lemma 5 already covered the lower part of the dichotomy. For the upper part, membership in W[1] is widely known (see e.g. [9].

For hardness, let $\mathcal{C}$ contain arbitrarily large structures of arity at least 2. We have already seen that without loss of generality, the digraphs associated with structures from $\mathcal{C}$ contain arbitrarily large cliques or tournaments. For ease of presentation, let us assume only the latter. Say, $\mathfrak{A} \in \mathcal{C}$ and R satisfy that

$D(\mathfrak{A}, R)$ contains a tournament of size k. Then $D(\mathfrak{A}^{\text{antiref}}, R)$ still contains the tournament. Every homomorphic image of this tournament into an antireflexive structure is necessarily injective. As $\mathfrak{A}^{\text{antiref}}$ itself is antireflexive, it follows for the core $\mathfrak{A}'$ of $\mathfrak{A}^{\text{antiref}}$, that $D(\mathfrak{A}', R)$ contains a tournament as a subdigraph. Then, $G(\mathfrak{A}')$ contains a clique of size k, so $\mathfrak{A}'$ has treewidth at least $k-1$. As k is arbitrary, the cores of structures from $\mathcal{C}^{\text{antiref}}$ have unbounded treewidth.

Using Theorem 7, we conclude that p-Hom$(\mathcal{C}^{\text{antiref}})$ is W[1]-hard. Lemma 8 then implies the W[1]-hardness of p-StrEmb$(\mathcal{C})$. □

3.2 Hardness of Counting

We now turn to the proof for the counting problems. Generally, we use $\leq^{\text{FPT-T}}$ to denote nonuniform FPT Turing reducibility. If $\mathcal{C}$ is recursively enumerable, we instead intend it to denote strongly uniform FPT Turing reducibility.

Proof of Theorem 3: Membership of p-#StrEmb$(\mathcal{C})$ in #W[1] is well-known [8] and if $\mathcal{C}$ is meagre, then Lemma 5 implies membership in FP.

So we may assume that $\mathcal{C}$ contains arbitrarily large structures of arity at least 2. By the above considerations, we can accordingly assume that the Gaifman graphs of structures in $\mathcal{C}$ contain arbitrarily large cliques. We show hardness, by giving an FPT Turing reduction from p-#Clique to p-#StrEmb$(\mathcal{C})$. Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. First we find a structure $\mathfrak{A} \in \mathcal{C}$ such that $G(\mathfrak{A})$ contains a k-clique. We assume $A = [k']$ with $k' \geq k$ and $G(\mathfrak{A}) \restriction [k]$ is a k-clique. Let τ be the vocabulary of $\mathfrak{A}$. We define a τ-structure $\mathfrak{B} = \mathfrak{B}(\mathfrak{A}, G, k)$ with universe

$$B := \big(V \times [k]\big) \dot\cup\, [k+1, k'].$$

To define the relations of $\mathfrak{B}$, we need two projections $\pi_1 : B \to V \dot\cup \{\bot\}$ and $\pi_2 : B \to A$ defined by

$$\pi_1(b) := \begin{cases} u, & \text{if } b = (u,i) \text{ for some} \\ & u \in V \text{ and } i \in [k] \\ \bot, & \text{if } b \in [k+1, k'], \end{cases} \qquad \pi_2(b) := \begin{cases} i, & \text{if } b = (u,i) \text{ for some} \\ & u \in V \text{ and } i \in [k] \\ b, & \text{if } b \in [k+1, k']. \end{cases}$$

Now for every $R \in \tau$ with arity r we let

$$R^{\mathfrak{B}} := \big\{(b_1, \ldots, b_r) \in B^r \mid (\pi_2(b_1), \ldots, \pi_2(b_r)) \in R^{\mathfrak{A}} \text{ and } \{\pi_1(b_1), \ldots, \pi_1(b_r)\} \setminus \{\bot\} \text{ is a clique in } G\big\}. \tag{1}$$

By our bounded arity assumption, we always have $r \leq r_0$ here, hence $|\mathfrak{B}|$ is polynomial in $|\mathfrak{A}|$ and k.

Let h be a strong embedding from $\mathfrak{A}$ to $\mathfrak{B}$. We call h *good* if

$$\pi_2(h(A)) = [k'] \tag{2}$$

Note that this implies that π_2 is bijective on $h(A)$. Then we can establish:

Claim 1. For every good h, if we let

$$\big\{(v_i, i)\big\} := h(A) \cap \big(V \times \{i\}\big)$$

for every $i \in [k]$, then the set $\{v_1, \ldots, v_k\}$ is a k-clique in G. ⊣

The proof of the next claim is straightforward.

Claim 2. Let $\bar{u} := (u_1, \ldots, u_k) \in V^k$ such that $\{u_1, \ldots, u_k\}$ is a k-clique in G. Then the mapping $h_{\bar{u}} : A \to B$ with

$$h_{\bar{u}}(i) := \begin{cases} (u_i, i), & \text{if } i \in [k], \\ i, & \text{if } i \in [k+1, k'] \end{cases}$$

is a good strong embedding from $\mathfrak{A}$ to $\mathfrak{B}$. ⊣

Let η be the number of good strong embeddings from $\mathfrak{A}$ to $\mathfrak{B}$, $\alpha := \big|\mathrm{Aut}(\mathfrak{A})\big|$ the number of automorphisms of $\mathfrak{A}$, and κ the number of k-cliques in G. Then:

Claim 3

$$\kappa = \frac{\eta}{\alpha \cdot k!}$$ ⊣

Theorem 3 now follows, if we can show how to compute the number η of good strong embeddings from $\mathfrak{A}$ to $\mathfrak{B}$. This is done using the principle of inclusion and exclusion:

For a τ-structure $\mathfrak{B}$ and a set $X \subseteq B$ let $\mathfrak{B}[X]$ denote the *induced* substructure of $\mathfrak{B}$ defined by X, i.e. $\mathfrak{B}[X] = (X, (R^{\mathfrak{B}} \cap X^{ar(R)})_{R \in \tau})$. Define for every set $I \subseteq [k']$ the structure $\mathfrak{B}_I := \mathfrak{B}[\pi_2^{-1}(I)]$. Let $\mathrm{StrEmb}(\mathfrak{A}, \mathfrak{B}_I)$ denote the set of strong embeddings from $\mathfrak{A}$ to $\mathfrak{B}_I$ and let b_I be the value returned by the p-#StrEmb($\mathcal{C}$) oracle on input $(\mathfrak{A}, \mathfrak{B}_I)$, i. e. $b_I = |\mathrm{StrEmb}(\mathfrak{A}, \mathfrak{B}_I)|$. Furthermore, define C_I as the set of strong embeddings $f : \mathfrak{A} \to \mathfrak{B}$ satisfying $\pi_2(f(A)) = I$. Let $c_I := |C_I|$. The definition of C_I immediately implies that $C_{[k']}$ is the set of all good strong embeddings of $\mathfrak{A}$ into $\mathfrak{B}$.

Obviously, $\mathrm{StrEmb}(\mathfrak{A}, \mathfrak{B}_I) = \dot{\bigcup}_{I' \subseteq I} C_{I'}$ for all $I \subseteq [k']$. Hence

$$c_I = b_I - \sum_{I' \subsetneq I} c_{I'}.$$

Then, by recursion on $|I|$, we can compute the $2^{k'}$ values c_I from our knowledge of the values b_I. As k' is bounded in terms of the parameter k, we can compute all of the $2^{k'}$ values by $2^{k'}$ oracle calls within the time bounds of an FPT Turing reduction. □

4 The Nondichotomies

For a parameterized (decision, counting, or otherwise) problem (Q, κ) and $A \subseteq \mathbb{N}$, the *restriction* of (Q, κ) to A, denoted $(Q, \kappa) \restriction A$, is the classical problem

Q, restricted to inputs x such that $\kappa(x) \in A$. We consider the case that A is decidable in polynomial time ($A \in \mathrm{P}$), when numbers are encoded in unary.

Let $\leq$ denote the reducibility for Q, e.g. polynomial time many-one reducibility for decision problems and polynomial time Turing reducibility for counting problems. If $A_1, A_2 \subseteq \mathbb{N}$ are in P and $A_1 \subseteq A_2$, and if Q is nontrivial, then, clearly, $(Q,\kappa) \restriction A_1 \leq (Q,\kappa) \restriction A_2$. Hence the lattice of polynomial time decidable subsets of $\mathbb{N}$ induces a partial order of degrees witnessed by restrictions of (Q,κ). We now establish a dense linear suborder.

Theorem 9. *Let (Q,κ) be a parameterized problem. Assume that Q is not solvable in polynomial time, but that (Q,κ) is solvable in* XP *time, i. e. on input x in time $|x|^{g(\kappa(x))}$ for some function $g : \mathbb{N} \to \mathbb{N}$.*

Then there is a dense linear order $\mathcal{O}$ of polynomial time decidable subsets of $\mathbb{N}$ such that for all $A_1, A_2 \in \mathcal{O}$ with $A_1 \subsetneq A_2$ we have $(Q,\kappa) \restriction A_2 \not\leq (Q,\kappa) \restriction A_1$.

The proof follows the lines of Ladner's classical argument.

Corollary 10. *If* $\mathrm{P} \neq \mathrm{NP}$*, respectively* $\mathrm{P} \neq \#\mathrm{P}$*, then the complexities of problems of the form* $\mathrm{StrEmb}(\mathcal{C})$*, respectively* $\#\mathrm{StrEmb}(\mathcal{C})$*, with $\mathcal{C}$ being some polynomial time decidable class of graphs, contain a dense linear order between* P *and* NP*, respectively between* P *and* #P*.*

The same holds for the homomorphism and embedding problems, and for structures instead of graphs.

The order's denseness implies that there is no finite classification of the unparameterized complexities of problems of the form $\mathrm{StrEmb}(\mathcal{C})$. Contrast this to our dichotomies.

As noted in the introduction, [2] contains an independently obtained proof of the fact that there is a polynomial time decidable class $\mathcal{C}$ of structures such that $\mathrm{Hom}(\mathcal{C})$ is neither in P nor complete for NP.

5 Conclusion and Open Problems

We give dichotomy results for the complexity of the restricted induced subgraph isomorphism problem. The upper parts of both our dichotomies are parameterized hardness, while the lower parts are classical tractability. Strong evidence is given that classifications of these problems cannot be given by classical complexity theory alone.

We were not able to classify the restricted subgraph isomorphism problem p-$\mathrm{Emb}(\mathcal{C})$. The decision problem is known to be fixed-parameter tractable if $\mathcal{C}$ is of bounded tree-width [1]. If $\mathcal{C}$ is of unbounded treewidth modulo homomorphic equivalence, then the problem is easily seen to be W[1]-hard. A natural example of the remaining cases is $\mathcal{C} = \{K_{k,k} \mid k \in \mathbb{N}\}$ for which p-$\mathrm{Emb}(\mathcal{C})$ coincides with the complete bipartite subgraph problem ([9], p. 355).

A classification of the counting problem p-$\#\mathrm{Emb}(\mathcal{C})$ is wide open as well. Clearly, this problem is hard if $\mathcal{C}$ is of unbounded treewidth. However, treewidth

is not the measure of choice here, as p-#Emb($\mathcal{C}$) is hard even if $\mathcal{C}$ is the class of all paths [8]. A natural example of the unknown cases is the parameterized problem of counting matchings [8].

Acknowledgement

Lemma 8 and its proof were suggested by an anonymous referee. The authors' original proof of Theorem 2 was much more complicated. Further thanks are due to Jörg Flum, Martin Grohe and Berit Grußien for comments on earlier versions of this paper. The first author also acknowledges the support of the National Nature Science Foundation of China (60673049).

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM 42(4), 844–856 (1995)
2. Bodirski, M., Grohe, M.: Non-dichotomies in Constraint Satisfaction Complexity. In: Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008, Track B) (to appear, 2008)
3. Chen, Y., Flum, J.: On parameterized path and chordless path problems. In: Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity, pp. 250–263. IEEE Computer Society, Los Alamitos (2007)
4. Dalmau, V., Jonsson, P.: The complexity of counting homomorphisms seen from the other side. Theor. Comput. Sci. 329(1-3), 315–323 (2004)
5. Dalmau, V., Kolaitis, P.G., Vardi, M.Y.: Constraint satisfaction, bounded treewidth, and finite-variable logics. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, CP 2002, pp. 310–326 (2002)
6. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness I: Basic results. SIAM J. Comput. 24(4), 873–921 (1995)
7. Flum, J., Grohe, M.: Fixed-parameter tractability, definability, and model-checking. SIAM J. Comput. 31(1), 113–145 (2001)
8. Flum, J., Grohe, M.: The parameterized complexity of counting problems. SIAM J. Comput. 33(4), 892–922 (2004)
9. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
10. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. J. ACM 54(1) (2007)
11. Ladner, R.E.: On the structure of polynomial time reducibility. J. ACM 22(1), 155–171 (1975)
12. McCartin, C.: Parameterized counting problems. Ann. Pure Appl. Logic 138(1-3), 147–182 (2006)

Spanners in Sparse Graphs

Feodor F. Dragan[1], Fedor V. Fomin[2], and Petr A. Golovach[2]

[1] Department of Computer Science, Kent State University, Kent, Ohio 44242, USA
dragan@cs.kent.edu*
[2] Department of Informatics, University of Bergen, PB 7803, 5020 Bergen, Norway
fedor.fomin,petr.golovach@ii.uib.no**

Abstract. A *t-spanner* of a graph G is a spanning subgraph S in which the distance between every pair of vertices is at most t times their distance in G. If S is required to be a tree then S is called a *tree t-spanner* of G. In 1998, Fekete and Kremer showed that on unweighted planar graphs the *tree t-spanner problem* (the problem to decide whether G admits a tree t-spanner) is polynomial time solvable for $t \leq 3$ and is NP-complete as long as t is part of the input. They also left as an open problem whether the tree t-spanner problem is polynomial time solvable for every fixed $t \geq 4$. In this work we resolve this open problem and extend the solution in several directions. We show that for every fixed t, it is possible in polynomial time not only to decide if a planar graph G has a tree t-spanner, but also to decide if G has a t-spanner of *bounded treewidth*. Moreover, for every fixed values of t and k, the problem, for a given planar graph G to decide if G has a t-spanner of treewidth at most k, is not only polynomial time solvable, but is *fixed parameter tractable* (with k and t being the parameters). In particular, the running time of our algorithm is linear with respect to the size of G. We extend this result from planar to a much more general class of sparse graphs containing graphs of bounded genus. An *apex graph* is a graph obtained from a planar graph G by adding a vertex and making it adjacent to some vertices of G. We show that the problem of finding a t-spanner of treewidth k is fixed parameter tractable on graphs that do not contain some fixed apex graph as a minor, i.e. on *apex-minor-free graphs*. Graphs of bounded treewidth are sparse graphs and our technique can be used to settle the complexity of the parameterized version of the *sparse t-spanner problem*, where for given t and m one asks if a given n-vertex graph has a t-spanner with at most $n - 1 + m$ edges. Our results imply that the sparse t-spanner problem is fixed parameter tractable on apex-minor-free graphs with t and m being the parameters. Finally we show that the tractability border of the t-spanner problem cannot be extended beyond the class of apex-minor-free graphs. In particular, we prove that for every $t \geq 4$, the problem of finding a tree t-spanner is NP-complete on *K_6-minor-free graphs*. Thus our results are tight, in a sense that the restriction of input graph being apex-minor-free cannot be replaced by H-minor-free for some non-apex fixed graph H.

* This work was partially done while the first author was visiting the Department of Informatics of University of Bergen.

** Supported by Norwegian Research Council.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 597–608, 2008.

1 Introduction

One of the basic questions in the design of routing schemes for communication networks is to construct a spanning network which has two (often conflicting) properties: it should have simple structure and nicely approximate distances of the network. This problem fits in a larger framework of combinatorial and algorithmic problems that are concerned with distances in a finite metric space induced by a graph. An arbitrary metric space (in particular a finite metric defined by a graph) might not have enough structure to exploit algorithmically. A powerful technique that has been successfully used recently in this context is to embed the given metric space in a simpler metric space such that the distances are approximately preserved in the embedding. New and improved algorithms have resulted from this idea for several important problems (see, e.g., [2,7,25]). Tree metrics are a very natural class of simple metric spaces since many algorithmic problems become tractable on them.

Peleg and Ullman [30] suggested the following parameter to measure the quality of a spanner. The spanner S of a graph G has the *stretch factor* t if the distance in S between any two vertices is at most t times the distance between these vertices in G. A *tree t-spanner* of a graph G is a spanning tree with a stretch factor t. If we approximate the graph by a tree t-spanner, we can solve the problem on the tree and the solution interpret on the original graph. Unfortunately, not many graph families admit good tree spanners. This motivates the study of sparse spanners, i.e. spanners with a small amount of edges. There are many applications of spanners in various areas; especially, in distributed systems and communication networks. In [30], close relationships were established between the quality of spanners (in terms of stretch factor and the number of spanner edges), and the time and communication complexities of any synchronizer for the network based on this spanner. Another example is the usage of tree t-spanners in the analysis of arrow distributed queuing protocols [14,24]. Sparse spanners are very useful in message routing in communication networks; in order to maintain succinct routing tables, efficient routing schemes can use only the edges of a sparse spanner [31]. We refer to the survey paper of Peleg [27] for an overview on spanners.

In this work we study t-spanners of bounded treewidth (we postpone the definition of treewidth till the next section). Specifically,

PROBLEM: k-Treewidth t-spanner
INSTANCE: A connected graph G and integers k and t.
QUESTION: Is there a t-spanner S of G of treewidth at most k?

Many algorithmic problems are tractable on graphs of bounded treewidth, and a spanner of small treewidth can be used to obtain an approximate solution to a problem on G. Since every connected graph with n vertices and at most $n-1+m$ edges is of treewidth at most $m+1$, we can see this problem as a generalization of tree t-spanner and sparse t-spanner problems.

Related work. Substantial work has been done on the tree t-spanner problem, also known as the minimum stretch spanning tree problem. Cai and Corneil

[6] have shown that, for a given graph G, the problem to decide whether G has a tree t-spanner is NP-complete for any fixed $t \geq 4$ and is linear time solvable for $t = 1, 2$ (the status of the case $t = 3$ is open for general graphs). An $O(\log n)$-approximation algorithm for the minimum value of t for the tree t-spanner problem is due to Emek and Peleg [21]. See the survey of Peleg [27] on more details on this problem and its variants.

The tree t-spanner problem on planar graphs was studied intensively. Fekete and Kremer [22] proved that the tree t-spanner problem on planar graphs is NP-complete (when t is part of the input). They also show that it can be decided in polynomial time whether a given planar graph has a tree 3-spanner. They gave also a polynomial time algorithm for any fixed t that decides for planar graphs with bounded face length whether there is a tree t-spanner. For fixed $t \geq 4$, the complexity of the tree t-spanner problem on planar graphs was left as an open problem [22].

There are several works investigating the complexity of the problem on subclasses of planar graphs. Peleg and Tendler [29] showed that the problem can be solved in polynomial time on outerplanar graphs, and also in the special case of 1-face depth graphs in which no interior vertex has degree 2. Boksberger et al. [4] investigated the problem on grids and subgrids. They presented polynomial time algorithm on grids and $O(OPT^4)$-approximation for subgrids.

Sparse t-spanners were introduced in [28] and [30] and since that time studied extensively. We refer the reader to [18,19,20] for some inapproximability and approximability results for the sparse spanner problem on general graphs. On planar (unweighted) graphs, the problem of determining, for a given n-vertex graph G and integers m and t, if G has a t-spanner with at most $n+m-1$ edges is NP-complete for every fixed $t \geq 5$. (The case $2 \leq t \leq 4$ is open.) [5]. A PTAS for the minimum number of edges for a special case of 2-spanners of 4-connected planar triangulations was obtained in [17].

Recently, a lot of work has been done on parameterized algorithms on planar graphs and more general classes of graphs (we refer e.g. to book [15] for more information on parameterized complexity and algorithms). Alber et al. [1] initiated the study of subexponential parameterized algorithms for the dominating set problem and its different variations. Demaine et al. [12,13] gave a general framework called bidimensionality to design parameterized algorithms for many problems on planar graphs and showed how by making use of this framework to extend results from planar graphs to much more general graph classes including H-minor-free graphs. However, this framework cannot be used directly to solve the k-Treewidth t-spanner problem because the theory of Demaine et al. is strongly based on the assumption that the parameterized problem should be minor or edge contraction closed, which is not the case for spanners. In particular, it is easy to construct an example when by contracting of an edge in a graph G with a t-spanner of treewidth k, one can obtain a graph which does not have such a spanner.

Our results. In this paper we resolve the problem left open in [22] and extend the solution in several directions. Our general technique is combinatorial

in nature and is based on the following observation. Let $\mathcal{G}$ be a class of graphs such that for every fixed t and every $G \in \mathcal{G}$, the treewidth of every t-spanner of G is $\Omega(treewidth(G))$. Then as an almost direct corollary of Bodlaender's Algorithm and Courcelle's Theorem (see Section 5 for details), we have that the k-Treewidth t-spanner problem is fixed parameter tractable on $\mathcal{G}$. Our main combinatorial result is the proof that the class of apex-minor-free graphs, which contains planar and bounded genus graphs, is in $\mathcal{G}$.

After preliminary Section 2, we start (Section 3) by proving the combinatorial properties of t-spanners in planar graphs. Our main result here is the proof that every t-spanner of a planar graph of treewidth k has treewidth $\Omega(k/t)$. The proof idea is based on the Robertson et al. theorem [33] on planar graphs excluding a grid as a minor. A technical complication of a direct usage of this theorem is that non-existence of a k-treewidth t-spanner in a minor or a contraction of a graph G does not imply non-existence of a k-treewidth t-spanner in G. This is why we have to work here with walls and topological minors.

It is possible to extend the combinatorial result on t-spanners in planar graphs to apex-minor-free graphs (Section 4). This extension is quite technical and is based on a number of new insights on the structure of apex-minor-free graphs. The main tools here are the structural theorem of Robertson and Seymour characterizing graphs excluding a graph as a minor and the theorem of Demaine and Hajiaghayi on grid-minors in such graphs. We find the study of the k-treewidth t-spanner problem on apex-minor-free graphs worth of efforts because of the following reason. Apex-minor-free graphs form a natural barrier for extension of many parameter/treewidth combinatorial bounds which hold for planar graphs [11]. However, for almost every such a parameter, the class of apex-minor-free graphs is not an algorithmic obstacle, in a sense, that very often it is possible to construct parameterized algorithms for H-minor-free graphs, where H is not necessary an apex graph, see, e.g. [12]. Surprisingly, this is not the case for the t-spanner problem. We show that the result on tractability of the problem on the class of apex-minor-free graphs is tight and cannot be extended further: the problem becomes intractable on H-minor-free graphs, when H is not an apex graph. In particular, for every $t \geq 4$, the problem of finding a tree t-spanner is NP-complete even on *K_6-minor-free graphs*.

Due the space restrictions some proofs are omitted here but they are given in the technical report [16].

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with the vertex set V and edge set E. (We often will use notations $V(G) = V$ and $E(G) = E$.) The *distance* $\mathrm{dist}_G(u, v)$ between vertices u and v of a connected graph G is the length (the number of edges) of a shortest (u, v)-path in G.

Let t be a positive integer. A subgraph S of G, such that $V(S) = V(G)$, is called a *(multiplicative) t-spanner*, if $\mathrm{dist}_S(u, v) \leq t \cdot \mathrm{dist}_G(u, v)$ for every pair of

vertices u and v. The parameter t is called the *stretch factor* of S. It is easy to see that the t-spanners can equivalently be defined as follows.

Proposition 1. *Let G be a connected graph, and t be a positive integer. A spanning subgraph S of G is a t-spanner of G if and only if for every edge (x, y) of G, $\mathrm{dist}_S(x, y) \leq t$.*

Given an edge $e = (x, y)$ of a graph G, the graph G/e is obtained from G by contracting the edge e; that is, to get G/e we identify the vertices x and y and remove all loops and replace all multiple edges by simple edges. A graph H obtained by a sequence of edge-contractions is said to be a *contraction* of G. H is a *minor* of G if H is a subgraph of a contraction of G. We say that a graph G is *H-minor-free* when it does not contain H as a minor. We also say that a graph class $\mathcal{G}$ is *H-minor-free* (or, excludes H as a minor) when all its members are H-minor-free. For example, the class of planar graphs is a K_5-minor-free graph class. An *apex graph* is a graph obtained from a planar graph G by adding a vertex and making it adjacent to some vertices of G. A graph class $\mathcal{G}$ is *apex-minor-free* if $\mathcal{G}$ excludes a fixed apex graph H as a minor. If an edge of a graph G is replaced by the path between it's ends then it is said that this edge is *subdivided*. A graph H is a *topological minor* of a graph G, if G contains a subgraph which is isomorphic to a graph obtained from H by subdividing some of its edges.

The (r, s)-*grid* is the Cartesian product of two paths of lengths $r-1$ and $s-1$. The (r, s)-*wall* is a graph W_{rs} with the vertex set $\{(i, j)\colon 1 \leq i \leq r, 1 \leq j \leq s\}$ such that two vertices (i, j) and (i', j') are adjacent if and only if either $i = i'$ and $j' \in \{j-1, j+1\}$, or $j = j'$ and $i' = i + (-1)^{i+j}$.

Let W_{rs} be a wall. By P_i^h we denote the shortest path connecting vertices $(i, 1)$ and (i, s), and by P_j^v is denoted the shortest path connecting vertices $(1, j)$ and (r, j) with assumption that, for $j > 1$, P_j^v contains only vertices (x, y) with $x = j-1, j$. We call by the *southern part of* W_{rs} the path P_r^h, and by the *northern part of* W_{rs} the path P_1^h. Similar, the *eastern* and the *western* parts are the paths P_s^v and P_2^v, correspondingly.

If W is obtained by subdivision of edges of W_{rs}, with slightly abusing the notation, we also will be using these terms for the paths obtained by subdivisions from the corresponding paths of W_{rs}.

It is easy to check that if a graph G contains the (r, r)-grid as a minor, then it contains W_{rr} as a topological minor. Also if G contains W_{rr} as a topological minor, then it contains $(r, \lfloor r/2 \rfloor)$-grid as a minor.

A *tree decomposition* of a graph G is a pair (X, U) where U is a tree whose vertices we will call *nodes* and $X = (\{X_i \mid i \in V(U)\})$ is a collection of subsets of $V(G)$ such that i)$\bigcup_{i \in V(U)} X_i = V(G)$, ii)for each edge $(v, w) \in E(G)$, there is an $i \in V(U)$ such that $v, w \in X_i$, and iii)for each $v \in V(G)$ the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of U. The *width* of a tree decomposition $(\{X_i \mid i \in V(U)\}, U)$ equals $\max_{i \in V(U)} \{|X_i| - 1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. We use notation $\mathbf{tw}(G)$ to denote the treewidth of a graph G. A tree decomposition with U being a path,

is called a *path decomposition* and the pathwidth of G is the minimum width over all path decompositions of G.

We will need the following result which is due to Robertson, Seymour & Thomas [33].

Proposition 2 ([33]). *Every planar graph with no (r,r)-grid as a minor has treewidth $\leq 6r-5$.*

A *surface* Σ is a compact 2-manifold without boundary (we always consider connected surfaces). A *line* in Σ is a subset homeomorphic to $[0,1]$ and a (closed) *disc* $\Delta \subseteq \Sigma$ is a subset homeomorphic to $\{(x,y) : x^2+y^2 \leq 1\}$. An O-arc is a subset of Σ homeomorphic to a circle. Whenever we refer to a *Σ-embedded graph* G we consider a 2-cell embedding of G in Σ. To simplify notations, we do not distinguish between a vertex of G and the point of Σ used in the drawing to represent the vertex or between an edge and the line representing it. We also consider a graph G embedded in Σ as the union of the points corresponding to its vertices and edges. That way, a subgraph H of G can be seen as a graph H where $H \subseteq G$.

3 Planar Graphs

In this section we prove that for every fixed t, a planar graph of large treewidth cannot have a t-spanner of small treewidth.

Theorem 1. *Let G be a planar graph of treewidth k and let S be a t-spanner of G. Then the treewidth of S is $\Omega(k/t)$.*

Proof. We need the following technical claim (the proof can be seen in [16]). Let G be a planar graph embedded in the plane and containing the wall W_{rs} as a topological minor. Let W be a subgraph of G isomorphic to a subdivision of W_{rs}. Let Δ be the disc in the plane which is bordered by the union of the southern, western, northern and eastern parts of W (with exclusion of pendant vertices) and containing W.

Claim 1. *For every $t \leq \min\{s/4, r/2\} - 1$, every t-spanner S of G contains a path connecting the southern and the northern parts of W, and a path connecting the eastern and the western parts of W. Moreover, both these paths are in Δ.*

Proof. Let us prove the claim for the eastern and the western parts of W. Suppose that for some t-spanner S of G there is no path completely inside of Δ connecting the eastern and the western parts of W. Consider the path $P^h_{\lceil r/2 \rceil}$ in the wall. We find the first edge (x,y) in this path (starting from the western part) with the following property: there is a path in $S \cap \Delta$ connecting the eastern part of W and x but there are no such paths for y. Clearly, such an edge has to exist. Let P be a shortest path in S connecting x and y. By the choice of x and y, path P is not entirely in Δ. So it can be divided into three subpaths: the first path P_1 connects x with some vertex u on the border of Δ, the second part P_2

connects u with some vertex v, which also lies on the border of Δ, the third path P_3 connects v and y, and $P_1 \cup P_3 \subset \Delta$. Note that vertex u cannot belong to the eastern part, and vertex v cannot belong to the western part. The length of P is at least northern or the southern part, then $\mathrm{dist}_S(x,u) \geq r/2-1 \geq t$. If v is in the northern or the southern part. then $\mathrm{dist}_S(y,v) \geq r/2-1 \geq t$. If u is in the western part and v is in the eastern part, then $\mathrm{dist}_S(x,u)+\mathrm{dist}_S(y,v) \geq s/2-1 \geq t$. Hence, in all cases, the length of P is at least $t+1$, and S is not a t-spanner. The claim for the northern and southern parts is proved by similar arguments. We have only to consider path $P^v_{\lfloor s/2 \rfloor+1}$ instead of $P^h_{\lceil r/2 \rceil}$. Note also that here we need the requirement $t \leq s/4-1$. □

Set now $r = \lfloor \frac{k+4}{6} \rfloor$ and let S be a t-spanner of G. By Proposition 2, G has an (r,r)-grid as a minor. Thus G has an (r,r)-wall W_{rr} as a topological minor. Wall W_{rr} contains $\lfloor \frac{r}{4t+1} \rfloor$ disjoint $(4t+1,r)$-walls. Let W be a subgraph of G isomorphic to a subdivision of W_{rr}. By applying Claim 1 to each $(4t+1,r)$-wall, we have that there are $\lfloor \frac{r}{4t+1} \rfloor$ vertex disjoint paths in S connecting eastern and western parts of W. By similar arguments, S also contains $\lfloor \frac{r}{4t+1} \rfloor$ vertex disjoint paths connecting southern and northern parts of W. The union of these paths contains $(\lfloor \frac{r}{4t+1} \rfloor, \lfloor \frac{r}{4t+1} \rfloor)$-grid as a minor. So, S contains this grid as a minor, too, and the treewidth of S is at least $\lfloor \frac{r}{4t+1} \rfloor = \lfloor \frac{\lfloor (k+4)/6 \rfloor}{4t+1} \rfloor = \Omega(k/t)$. □

4 Apex-Minor-Free Graphs

In this section, we extended the results of Theorem 1 to graphs with bounded genus and to apex-minor-free graphs.

4.1 Bounded Genus

The Euler genus $\mathbf{eg}(\Sigma)$ of a nonorientable surface Σ is equal to the nonorientable genus $\tilde{g}(\Sigma)$ (or the crosscap number). The Euler genus $\mathbf{eg}(\Sigma)$ of an orientable surface Σ is $2g(\Sigma)$, where $g(\Sigma)$ is the orientable genus of Σ. The following extension of Proposition 2 on graphs of bounded genus is due to Demaine et al. [12].

Proposition 3 ([12]). *If G is a graph with treewidth more than $6r(\mathbf{eg}(G)+1)$ which is embeddable on a surface with Euler genus $\mathbf{eg}(G)$, then G has the (r,r)-grid as a minor.*

We also need a result roughly stating that if a graph G with a big wall as a topological minor is embedded on a surface Σ of small genus, then there is a disc in Σ containing a big part of the wall of G. This result is implicit in the work of Robertson and Seymour and there are simpler alternative proofs by Mohar and Thomassen [26,35]. We use the variant of this result from Geelen et al. [23]. Combining this result and Proposition 3, and using the same arguments as in the planar case, we have the following theorem, which proof can be seen in [16].

Theorem 2. *Let G be a graph of treewidth k and Euler genus g, and let S be a t-spanner of G. Then the treewidth of S is $\Omega(\frac{k}{t \cdot g^{3/2}})$.*

4.2 Excluding Apex as a Minor

This extension of Theorems 1 and 2 to apex-minor-free graphs is based on a structural theorem of Robertson and Seymour [32]. Before describing this theorem we need some definitions.

Definition 1 (Clique-Sums). *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two disjoint graphs, and $k \geq 0$ an integer. For $i = 1, 2$, let $W_i \subset V_i$, form a clique of size h and let G_i' be the graph obtained from G_i by removing a set of edges (possibly empty) from the clique $G_i[W_i]$. Let $F : W_1 \to W_2$ be a bijection between W_1 and W_2. We define the h-*clique-sum *of G_1 and G_2, denoted by $G_1 \oplus_{h,F} G_2$, or simply $G_1 \oplus G_2$ if there is no confusion, as the graph obtained by taking the union of G_1' and G_2' by identifying $w \in W_1$ with $F(w) \in W_2$, and by removing all the multiple edges. The image of the vertices of W_1 and W_2 in $G_1 \oplus G_2$ is called the* join *of the sum.*

Note that some edges of G_1 and G_2 are not edges of G, because it is possible that they were added by clique-sum operation. Such edges are called *virtual*.

We remark that $\oplus$ is not well defined; different choices of G_i' and the bijection F could give different clique-sums. A sequence of h-clique-sums, not necessarily unique, which result in a graph G, is called a *clique-sum decomposition* of G.

Definition 2 (**h-nearly embeddable graphs**). *Let Σ be a surface with boundary cycles $C_1, \ldots, C_h$, i.e. each cycle C_i is the border of a disc in Σ. A graph G is h-*nearly embeddable *in Σ, if G has a subset X of size at most h, called* apices, *such that there are (possibly empty) subgraphs $G_0, \ldots, G_h$ of $G \setminus X$ such that i)$G \setminus X = G_0 \cup \cdots \cup G_h$, ii)$G_0$ is embeddable in Σ, we fix an embedding of G_0, graphs $G_1, \ldots, G_h$ (called* vortices*) are pairwise disjoint, iii)for $1 \leq \cdots \leq h$, let $U_i := \{u_{i_1}, \ldots, u_{i_{m_i}}\} = V(G_0) \cap V(G_i)$, G_i has a path decomposition (B_{ij}), $1 \leq j \leq m_i$, of width at most h such that a)for $1 \leq i \leq h$ and for $1 \leq j \leq m_i$ we have $u_j \in B_{ij}$, b)for $1 \leq i \leq h$, we have $V(G_0) \cap C_i = \{u_{i_1}, \ldots, u_{i_{m_i}}\}$ and the points $u_{i_1}, \ldots, u_{i_{m_i}}$ appear on C_i in this order (either if we walk clockwise or anti-clockwise).*

The following proposition is known as the Excluded Minor Theorem [32] and is the cornerstone of Robertson and Seymour's Graph Minors theory.

Proposition 4 ([32]). *For every graph H there exists an integer h, depending only on the size of H, such that every graph excluding H as a minor can be obtained by h-clique-sums from graphs that can be h-nearly embedded in a surface Σ in which H cannot be embedded.*

We also need the following result of Demaine and Hajiaghayi [13].

Proposition 5 ([13]). *If G is an H-minor-free graph with treewidth more than k, then G has the $(\Omega(k), \Omega(k))$-grid as a minor (the hidden constants in the Ω notation depend only on the size of H).*

Theorem 3. *Let H be a fixed apex graph. For every t-spanner S of an H-minor-free graph G, the treewidth of S is $\Omega(\mathbf{tw}(G))$. (The hidden constants in the Ω notation depend only on the size of H and t).*

Proof. Due to space restrictions only the sketch of the proof is given here (see [16] for the complete proof). Let G be an H-minor-free graph of treewidth k. It is well known, that for any pair of graphs G_1, G_2, $\mathbf{tw}(G_1 \oplus G_2) \leq \max\{\mathbf{tw}(G_1), \mathbf{tw}(G_2)\}$. Thus, by decomposing G as a clique sum described in Proposition 4, we conclude that there is a summand G' in this clique sum such that a) G' is h-almost embeddable in a surface Σ of genus h; b) the treewidth of G' is at least k. The further proof is performed in two steps. First we prove that Σ contains a closed disc $\Delta' \subset \Sigma$ such that *i*) $G' \cap \Delta'$ contains an $(\Omega(k), \Omega(k))$-wall as a topological minor and *ii*) no vertex of $G' \cap \Delta'$ is adjacent to an apex vertex and to a vertex from a vortex. The proof is based on Proposition 5. In the second step, by extending the arguments used for planar graphs on the wall inside Δ', we prove that every t-spanner of G has a large grid as a minor, and thus has treewidth $\Omega(k)$. □

5 Algorithmic Consequences

This section discusses algorithmic consequences of the combinatorial results obtained above. The proof of the following generic algorithmic observation is a combination of known results.

Theorem 4. *Let $\mathcal{G}$ be a class of graphs such that, for every $G \in \mathcal{G}$ and every t-spanner S of G, the treewidth of S is at least $\mathbf{tw}(G) \cdot f_{\mathcal{G}}(t)$, where $f_{\mathcal{G}}$ is the function only of t. Then for every fixed k and t, the existence of a t-spanner of treewidth at most k in $G \in \mathcal{G}$ can be decided in linear time.*

Proof. Let $G \in \mathcal{G}$ be a graph on n vertices and m edges. For given integers k and t, we use Bodlaender's Algorithm [3] to decide in time $O(n+m)$ if $\mathbf{tw}(G) \leq k/f_{\mathcal{G}}(t)$ (the hidden constants in the big-O depend only on k and $f_{\mathcal{G}}(t)$). If Bodlaender's Algorithm reports that $\mathbf{tw}(G) > k/f_{\mathcal{G}}(t)$, then we conclude that G does not have a t-spanner of treewidth at most k. Otherwise (when $\mathbf{tw}(G) \leq k/f_{\mathcal{G}}(t)$), Bodlaender's Algorithm computes a tree decomposition of G of width at most $k/f_{\mathcal{G}}(t)$. Now we want to apply Courcelle's Theorem [8,9], namely that every problem expressible in monadic second order logic (MSOL) can be solved in linear time on graphs of constant treewidth. To apply Courcelle's Theorem (and to finish the proof of our Theorem), we have to show that, for every fixed positive integers k and t, the property that a graph S is a t-spanner of treewidth at most k is expressible in MSOL. It is known that the property that a subgraph S has the treewidth at most k is expressible in MSOL for every fixed k (see, for example, [10]). Since any path is a sequence of adjacent edges, we have that the condition "for every edge (x, y) of G, $\mathrm{dist}_S(x, y) \leq t$" can be written as an MSOL formula for every fixed t. By Proposition 1, this yields that "S is a t-spanner of treewidth at most k" is expressible in MSOL. □

Theorems 3 and 4 imply the following result, which is the main algorithmic result of this paper. (Let us note that for $k = 1$, Corollary 1 provides the answer to the question of Fekete and Kremer [22].)

Corollary 1. *Let H be a fixed apex graph. For every fixed k and t, the existence of a t-spanner of treewidth at most k in an H-minor-free graph G can be decided in linear time.*

It is easy to see that the treewidth of a connected n-vertex graph with $n+m-1$ edges is at most $m+1$. Since for a fixed m, the property of that a S is a spanning subgraph of G with $n+m-1$ edges is in MSOL, we have (as in the proof of Theorem 4) that the combination of Theorem 3 with Bodlaender's Algorithm and Courcelle's Theorem implies the following corollary

Corollary 2. *Let H be a fixed apex graph. For every fixed m and t, the existence of a t-spanner with at most $n-1+m$ edges in an n-vertex H-minor-free graph G can be decided in linear time.*

It is easy to show that it is not possible to extend Theorem 3 to the class of H-minor free graphs, where H is not necessary an apex graph. For $i \geq 1$, let H_i be a graph obtained by adding to the (i,i)-grid a vertex v and making it adjacent to all vertices of the grid. Each of the graphs H_i, $i \geq 1$, does not contain the complete graph on six vertices K_6 as a minor. The treewidth of H_i is i, but it has a 2-spanner of treewidth one, which is the star with center in v. Thus, Theorem 4 cannot be used on graphs excluding a non-apex graph as a minor. Similar "apex-minor-free barrier" for using combinatorial bounds for parameterized algorithms was observed for other problems (e.g., parameterized dominating set [11]). However, for many of those problems, there are parameterized algorithms for H-minor-free graphs, which are based on dynamic programming over clique-sums of apex-minor-free graphs by making use of Robertson-Seymour structural theorem (Proposition 4), see, e.g. [12]. So, for many parameterized problems, combinatorial "apex-minor-free barrier" can be overcame. Surprisingly, this is not the case for the t-spanner problem. In particular, the tree 4-spanner problem is NP-complete on apex graphs, and since each apex graph is K_6-minor-free, it is NP-complete, for example, for K_6-minor-free graphs.

Note also that for apex graphs the claim of Theorem 3 is not correct. For $i \geq 1$, let H_i be a graph obtained by adding to the (i,i)-grid a vertex v and making it adjacent to all vertices of the grid. The graphs H_i, $i \geq 1$, do not contain the complete graph on six vertices K_6 as a minor. The treewidth of H_i is i, but it has a 2-spanner of treewidth one, which is the star with center in v.

Theorem 5. *For every fixed $t \geq 4$, deciding if an apex graph G has a tree t-spanner is NP-complete.*

Proof. The proof of this result is based on a modification of the reduction of Cai and Corneil [6] adapted for our purposes, and is given in [16]. □

6 Conclusion

We have shown that for fixed k and t, one can decide in linear time if an apex-minor-free graph G has a t-spanner of treewidth at most k. The results we used in our proof, Bodlaender's Algorithm and Courcelle's Theorem, have huge hidden constants in the running time, and thus Corollary 1 is of theoretical interest mainly. Since for K_6-minor-free graphs and $t = 4$ the problem is NP complete, we doubt that it is possible to design fast practical algorithms solving t-spanner problem on apex-minor-free graphs. However, it is likely that on planar graphs and for small values of t, our ideas can be used to design practical algorithms. First of all, instead of using Bodlaender's algorithm, one can use Ratcatcher algorithm of Seymour-Thomas [34] to find exact branchwidth of a planar graph. The running time of the algorithm is cubic, but there is no hidden constants. The second bottleneck of our approach for practical applications is the usage of Courcelle's Theorem. Instead of that, for small values of t, it is more reasonable to construct dynamic programming algorithms that use the properties of planarity and of the problem.

References

1. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for dominating set and related problems on planar graphs. Algorithmica 33, 461–493 (2002)
2. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: FOCS 1996, pp. 184–193 (1996)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
4. Boksberger, P., Kuhn, F., Wattenhofer, R.: On the approximation of the minimum maximum stretch tree problem, Tech. Report 409, ETH Zürich, Switzerland (2003)
5. Brandes, U., Handke, D.: P-completeness results for minimum planar spanners. Discrete Mathematics & Theoretical Computer Science 3, 1–10 (1998)
6. Cai, L., Corneil, D.G.: Tree spanners. SIAM J. Discrete Math. 8, 359–387 (1995)
7. Charikar, M., Chekuri, C., Goel, A., Guha, S., Plotkin, S.: Approximating a Finite Metric by a Small Number of Tree Metrics. In: FOCS 1998, pp. 379–388 (1998)
8. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
9. Courcelle, B.: The monadic second-order logic of graphs III: Treewidth, forbidden minors and complexity issues. Informatique Théorique 26, 257–286 (1992)
10. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Handbook of graph grammars and computing by graph transformation, vol. 1, pp. 313–400. World Sci. Publ., River Edge (1997)
11. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Bidimensional parameters and local treewidth. SIAM J. Discrete Math. 18, 501–511 (2004)
12. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. Journal of the ACM 52, 866–893 (2005)
13. Demaine, E.D., Hajiaghayi, M.: Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In: SODA 2005, pp. 682–689 (2005)

14. Demmer, M.J., Herlihy, M.: The arrow distributed directory protocol. In: Kutten, S. (ed.) DISC 1998. LNCS, vol. 1499, pp. 119–133. Springer, Heidelberg (1998)
15. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer, New York (1999)
16. Dragan, F.F., Fomin, F.V., Golovach, P.A.: Spanners in sparse graphs, Tech. Rep. 366, Dept. of informatics, University of Bergen (2008), http://www.ii.uib.no/publikasjoner/texrap/pdf/2008-366.pdf
17. Duckworth, W., Wormald, N.C., Zito, M.: A PTAS for the sparsest 2-spanner of 4-connected planar triangulations. J. of Discrete Algorithms 1, 67–76 (2003)
18. Elkin, M., Peleg, D.: Strong Inapproximability of the Basic k-Spanner Problem. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 636–647. Springer, Heidelberg (2000)
19. Elkin, M., Peleg, D.: The Hardness of Approximating Spanner Problems. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 370–381. Springer, Heidelberg (2000)
20. Elkin, M., Peleg, D.: Approximating k-spanner problems for $k > 2$. Theoretical Computer Science 337, 249–277 (2005)
21. Emek, Y., Peleg, D.: Approximating minimum max-stretch spanning trees on unweighted graphs. In: SODA 2004, pp. 261–270 (2004)
22. Fekete, S.P., Kremer, J.: Tree spanners in planar graphs. Discrete Appl. Math. 108, 85–103 (2001)
23. Geelen, J.F., Richter, R.B., Salazar, G.: Embedding grids in surfaces. European J. Combin. 25, 785–792 (2004)
24. Herlihy, M., Tirthapura, S., Wattenhofer, R.: Competitive concurrent distributed queuing. In: PODC 2001, pp. 127–133 (2001)
25. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some its algorithmic applications. Combinatorica 15, 215–245 (1995)
26. Mohar, B.: Combinatorial local planarity and the width of graph embeddings. Canad. J. Math. 44, 1272–1288 (1992)
27. Peleg, D.: Low stretch spanning trees. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 68–80. Springer, Heidelberg (2002)
28. Peleg, D., Schäffer, A.A.: Graph Spanners. J. Graph Theory 13, 99–116 (1989)
29. Peleg, D., Tendler, D.: Low stretch spanning trees for planar graphs, Tech. Report MCS01-14, Weizmann Science Press of Israel, Israel (2001)
30. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. In: PODC 1987, pp. 77–85 (1987)
31. Peleg, D., Upfal, E.: A tradeoff between space and efficiency for routing tables. In: STOC 1988, pp. 43–52 (1988)
32. Robertson, N., Seymour, P.D.: Graph minors. XVI. Excluding a non-planar graph. J. Combin. Theory Ser. B 89, 43–76 (2003)
33. Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a planar graph. J. Comb. Theory, Ser. B 62, 323–348 (1994)
34. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. Combinatorica 14, 217–241 (1994)
35. Thomassen, C.: A simpler proof of the excluded minor theorem for higher surfaces. J. Combin. Theory Ser. B 70, 306–311 (1997)

Distance Oracles for Unweighted Graphs: Breaking the Quadratic Barrier with Constant Additive Error

Surender Baswana[1,⋆], Akshay Gaur[2], Sandeep Sen[2], and Jayant Upadhyay[2]

[1] Department of Comp. Sc. & Engg. Indian Institute of Technology Kanpur, Kanpur - 208016, India
sbaswana@iitk.ac.in

[2] Department of Comp. Sc. & Engg., Indian Institute of Technology Delhi, New Delhi-110016, India
{manu,ssen,jayant}@cse.iitd.ernet.in

Abstract. Thorup and Zwick, in the seminal paper [Journal of ACM, 52(1), 2005, pp 1-24], showed that a weighted undirected graph on n vertices can be preprocessed in subcubic time to design a data structure which occupies only subquadratic space, and yet, for any pair of vertices, can answer distance query approximately in constant time. The data structure is termed as approximate distance oracle. Subsequently, there has been improvement in their preprocessing time, and presently the best known algorithms [4,3] achieve expected $O(n^2)$ preprocessing time for these oracles. For a class of graphs, these algorithms indeed run in $\Theta(n^2)$ time. In this paper, we are able to break this quadratic barrier at the expense of introducing a (small) constant additive error for unweighted graphs. In achieving this goal, we have been able to preserve the optimal size-stretch trade offs of the oracles. One of our algorithms can be extended to weighted graphs, where the additive error becomes $2 \cdot w_{\max}(u, v)$ - here $w_{\max}(u, v)$ is the heaviest edge in the shortest path between vertices u, v.

1 Introduction

Let $G = (V, E)$ be a graph on $|V| = n$ vertices and $|E| = m$ edges, and $\delta(u, v)$ denote the distance between any pair of vertices $u, v \in V$ in graph G. The all-pairs shortest paths (APSP) problem requires preprocessing the given graph G so as to build a data structure using which we can retrieve distance or the shortest path between any pair of vertices efficiently. APSP is undoubtedly one of the most fundamental algorithmic graph problems of computer science. Despite being a classical problem with widespread applications, there exists a huge gap between the lower bound $\Omega(n^2)$ and the worst case upper bound $O(n^3/\log^2 n)$ (due to Chan [6]) of the time complexity of APSP problem. Furthermore, $\Theta(n^2)$

⋆ The work was supported by a fellowship from Research I Foundation, CSE, IIT Kanpur.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 609–621, 2008.

space requirement is a major bottleneck for graphs in many large scale applications. These two factors have motivated researchers to design efficient algorithms (or data structures) for reporting *approximate* distances. In the last fifteen years, many novel algorithms [1,8,7,2,11] have been designed which work for undirected graphs. However, among all these algorithms the *approximate distance oracles* designed by Thorup and Zwick [12] deserve special mention. They showed that any given weighted undirected graph on n vertices can be preprocessed in subcubic time for any integer $t \geq 3$ to build a data structure of sub-quadratic size which for any pair of vertices u, v reports t-approximate distance - at least $\delta(u,v)$ and at most $t\delta(u,v)$. There are two very impressive features of their data structure. First, the trade-off between *stretch* t and the size of data structure is essentially optimal assuming a 1963 girth lower bound conjecture of Erdős [9] and second, in spite of its sub-quadratic size their data structure can answer any distance query in *constant* time, hence the name "oracle". More precisely, Thorup and Zwick achieved the following result.

Theorem 1. *[12] For any integer $k \geq 1$, an undirected weighted graph on n vertices and m edges can be preprocessed in expected $O(kmn^{1/k})$ time to build a data structure of size $O(kn^{1+1/k})$ that can answer any $(2k-1)$-approximate distance query in $O(k)$ time.*

Having achieved optimal size-stretch trade offs, and essentially constant query time, it is only the preprocessing time of these oracles which may be improved. The preprocessing time has been improved to $O(\min(n^2, kmn^{1/k}))$ for unweighted graphs [4], and recently for weighted graphs as well [3]. Therefore, a natural question is whether it is possible to achieve $O(m+n^{2-\epsilon})$ - a subquadratic upper bound for constructing approximate distance oracles. Note that any approximate all pairs shortest path algorithm takes $\Omega(n^2)$ steps because of the output size. Therefore, a sub-quadratic time oracle construction provides a clear advantage over such algorithms when we are not interested in all the pair-wise distances. The main objective here is to achieve sub-quadratic preprocessing time for approximate distance oracles without violating the size-stretch trade off. It may be noted that the quadratic upper bound of the existing preprocessing algorithms [12,4] for these oracles is indeed tight - there exists a family of graphs on which these algorithms would execute in $\Theta(n^2)$ time.

In this paper, we design approximate distance oracles which, at the expense of constant additive error, are constructable in sub-quadratic time and preserve size stretch trade-off optimally. More precisely, we show the following. *For any $k > 1$, there is a data-structure which occupies $O(kn^{1+1/k})$ space such that for any pair of vertices $u, v \in V$, it takes $O(k)$ time to return $\hat{\delta}(u,v)$ satisfying*

$$\delta(u,v) \leq \hat{\delta}(u,v) \leq (2k-1)\delta(u,v) + c_k \quad \textit{where} \quad c_k = 2 \quad \textit{for } k \geq 3 \quad \textit{and} \quad c_2 = 8$$

As a natural extension of $(2k-1)$-approximate distance oracle of [12], we denote the above oracle by $(2k-1, c_k)$-approximate distance oracle, where the first term $(2k-1)$ is the stretch (multiplicative error) and c_k is the surplus (additive error). The expected preprocessing time for $(2k-1, c_k)$ oracle is $O(m+kn^{2-\alpha_k})$, where

Table 1. Comparing the new algorithms with the existing algorithms for approximate distance oracles

Stretch	Space	Preprocessing Time	Reference
$(2k-1,0)$	$O(kn^{1+1/k})$	$O(\min(n^2, kmn^{1/k}))$	[12,3,4]
$(3,8)$	$O(n^{3/2})$	$O(\min(m+n^{\frac{23}{12}}, m\sqrt{n}))$	**this paper**
$(2k-1,2)$, $k \geq 3$	$O(kn^{1+1/k})$	$O(\min(m+kn^{\frac{3}{2}+\frac{1}{2k}+\frac{1}{2k-2}}, kmn^{\frac{1}{k}}))$	**this paper**

α_k takes value in the interval $[\frac{1}{12}, \frac{1}{2})$ - takes value $\frac{1}{12}$ for $k=2$ and approaches $\frac{1}{2}$ steadily as k increases (see Table 1).

In short, the small additive error has allowed us to break the quadratic barrier of preprocessing of approximate distance oracles. It would be very important to explore the limits to which the preprocessing time can be further improved. The result of this paper can be viewed as the first significant step in this direction.

1.1 Overview of the New Algorithms

The observation which forms the basis of our algorithms is the simple fact that the $O(kmn^{1/k})$ time complexity of the algorithm of Thorup and Zwick [12] is already sub-quadratic provided the graph is sparse enough. In order to utilize this observation, we use the idea of partitioning the graph into sparse and dense subgraphs. Previously this idea was used by the algorithms which compute all-pairs approximate distance with purely additive error only [1,8]. Using a random sample $S \subseteq V$ of vertices, we define a sparse subgraph with $o(n^{2-1/k})$ edges, and execute Thorup and Zwick algorithm on this sub graph. This algorithm will execute in $o(n^2)$ time and will easily take care of the case when the shortest paths between a pair of vertices is fully preserved in the sparse graph. Novelty of our algorithms is to handle the other case. Our algorithms make use of a combination of old and new ideas which enables achieving sub-quadratic space without compromising the optimal size-stretch trade-off. In order to make these ideas work, our algorithms effectively use suitable emulators and spanners which are sufficiently sparse.

Definition 1. *An (α,β)-spanner of a graph $G=(V,E)$ is a subgraph (V,E'), $E' \subseteq E$ with the property that distance between any two vertices $u,v \in V$ in the spanner is at least $\delta(u,v)$ and at most $\alpha(\delta(u,v))+\beta$.*

Definition 2. *An (α,β)-emulator of a graph $G=(V,E)$ is a weighted graph (V,E^*) such that the distance $\delta^*(u,v)$ between any two vertices $u,v \in V$ in the emulator is at least $\delta(u,v)$ and at most $\alpha\delta(u,v)+\beta$.*

In the following section we describe the notations and lemmas which will be used throughout this paper. In section 3, we describe our $(3,8)$-approximate distance oracle. In section 4, we describe $(2k-1,2)$-approximate distance oracle for $k>2$.

2 Preliminaries

For a given graph $G = (V, E)$, and any subset $S \subseteq V$, we shall use the following notations :

- $\mathcal{N}(v)$: the set consisting of v and every neighbor of v in the graph G.
- $\mathcal{N}(S)$: $\cup_{v \in S} \mathcal{N}(v)$.
- $p_S(v)$: the vertex from set S which is nearest to v (break the tie arbitrarily in case there are multiple nearest vertices).
- $\delta(v, S)$: distance between v and $p_S(v)$.
- $E(v)$: the set of edges in G which are incident on v.
- $\mathcal{E}_S(v)$: the set $E(v)$ if v is not adjacent to any vertex of set S, and $\emptyset$ otherwise.
- $\mathcal{E}_S$: $\cup_{v \in V} \mathcal{E}_S(v)$.
- G_S : the subgraph $(V, \mathcal{E}_S)$.
- $\mathcal{O}_t^{\mathcal{G}}$: the t-approximate distance oracle of Thorup and Zwick [12] created on a subgraph $\mathcal{G}$ of G.

Our data structure will store information about $p_S(v)$ and $\delta(v, S)$ for each vertex in the given graph G. To compute this information, the graph G can be processed in just $O(m)$ time as follows : *insert a dummy vertex o in to the graph, connect it to all the vertices of set S, and perform a BFS traversal on the graph starting from o.* We shall use $\mathcal{T}_S$ to denote the set of edges of this BFS tree excluding the edges incident on the dummy vertex.

Lemma 1. *The edge set $\mathcal{T}_S$ preserves the shortest path between v and $p_S(v)$ for all $v \in V$. The size of $\mathcal{T}_S$ is $O(n)$.*

Now we redefine an important concept (due to Thorup and Zwick [12]) of *ball* around a vertex.

Definition 3. *[12] For a vertex $u \in V$ and a set $S \subseteq V$ in a graph $G = (V, E)$, we define $ball(u, V, S)$ as the sub graph induced by all those vertices $v \in V$ which satisfy $\delta(u, v) < \delta(u, S)$ (i.e., for u, it is v which is nearer than $p_S(u)$).*

We now state the following Lemma about the number of vertices and edges in $ball(u, V, S)$ when S is formed by random sampling.

Lemma 2. *[12,4] For a given graph $G = (V, E)$, let $S \subseteq V$ be a set formed by selecting each vertex from V independently with probability $q > 0$. Then the expected number of vertices and expected number of edges in $ball(u, V, S)$ are $O(1/q)$ and $O(1/q^2)$ respectively.*

We shall now state a few important Lemmas about the sparse subgraph $(V, \mathcal{E}_S)$.

Lemma 3. *If set $S \subseteq V$ is formed by selecting each vertex independently with probability $q > 0$, the expected size of the set $\mathcal{E}_S$ would be $O(n/q)$.*

Lemma 4. *If on the shortest path between any two vertices $u, v \in V$ in the graph $G = (V, E)$ there are no two consecutive vertices in set $\mathcal{N}(S)$, then the shortest path between u and v is preserved exactly in the subgraph $(V, \mathcal{E}_S)$.*

The above property of the edge set $\mathcal{E}_S$ will prove to be very useful in our construction. For the other case we observe the following.

Lemma 5. *If the shortest path between u and v in the graph G contains at least 2 consecutive vertices from $\mathcal{N}(S)$, then $\delta(u,S)+\delta(v,S) \leq \delta(u,v)+1$.*

Proof. Suppose on the shortest path between u and v, u' be the vertex from the set $\mathcal{N}(S)$ nearest to u, and v' be the vertex from the set $\mathcal{N}(S)$ nearest to v. Since $u' \in \mathcal{N}(S)$ either u' belongs to S or some neighbor of u' belongs to S. This implies that $\delta(u,u') \geq \delta(u,S)-1$. Similarly $\delta(v,v') \geq \delta(v,S)-1$. Also note that $\delta(u,u')+\delta(u',v')+\delta(v',v) = \delta(u,v)$. Furthermore, $\delta(u',v') \geq 1$ since there are at least two vertices from $\mathcal{N}(S)$ on the shortest path between u and v. Therefore, $\delta(u,u')+\delta(v',v)+1 \leq \delta(u,v)$. This along with the lower bounds on $\delta(u,u')$ and $\delta(v,v')$ derived above imply that $\delta(u,S)+\delta(v,S) \leq \delta(u,v)+1$.

For construction of our $(2k-1,2)$-oracle for $k>2$, we shall employ the following result on spanners.

Theorem 2. *[10] For a given unweighted graph $G=(V,E)$ and any integer $k>1$, there exists an $O(m)$ time algorithm for computing a $(2k-1)$-spanner of size $O(n^{1+1/k})$.*

3 A $(3,c)$-Approximate Distance Oracle in Expected $O(n^{2-\frac{1}{12}})$ Time

Let G be the given undirected unweighted graph. Let S be a set formed by selecting each vertex independently with probability $n^{-\frac{5}{12}}$. Our preprocessing algorithm for $(3,c)$-approximate distance oracle, where $c=8$, will employ the sparse subgraph $(V,\mathcal{E}_S)$ and an *emulator* of the given graph G. We shall need a $(3,2)$-emulator which also satisfies some additional properties which are very crucial (see Lemma 7). We describe the construction and properties of this emulator in the following subsection first.

3.1 The Emulator (V,E^*) : Its Construction and Properties

In the construction of the emulator, we shall employ the $(3,2)$-spanner designed by Baswana et al. [5].

Theorem 3. *[5] For a given graph $G=(V,E)$, let S' be a set formed by selecting each vertex from V independently with probability $p=n^{-\frac{1}{3}}$. It takes expected $O(m)$ time to construct a $(3,2)$-spanner of size $O(n^{4/3})$ that satisfies the following additional properties for each $u \in V$.*

1. *If $u \in V$ has no neighbor from set S' in G, then every edge incident onto u will be in the spanner.*
2. *If u has one or more neighbors from set S' in G, then for some unique neighbor among them, denoted by $c(u)$, the following assertions hold true.*

(a) the edge $(u, c(u))$ is present in the spanner also.
(b) for each edge $(u, v) \in E$ not present in the spanner, there is a path between $c(u)$ and $c(v)$ in the spanner with length at most 3.

Algorithm for building emulator (V, E^*)

1. Add the edges of $\mathcal{T}_S$ (see Lemma 1)to E^*.
2. Let S' be the set formed by selecting each vertex with probability $n^{-\frac{1}{3}}$, and let *span* be the $(3, 2)$-spanner of the graph as stated in Theorem 3. Add all the edges of *span* to E^*.
3. From each vertex $v \in V \backslash S$ perform BFS traversal up to level $\delta(v, S) - 1$ to compute distance to each $x \in S'$ which is present in $ball(v, V, S)$, and add an edge (v, x) of weight $\delta(v, x)$ to E^*.

Lemma 6. *The emulator (V, E^*) output by the above algorithm has size $O(n^{4/3})$ and can be constructed in expected $O(m + n^{11/6})$ time.*

Proof. The first two steps take expected $O(m)$ time [5]. The third step is performed by executing BFS traversal from each $v \in V \backslash S$ up to level $\delta(v, S) - 1$. This will involve traversing only the edges of the subgraph $ball(v, V, S)$. So it follows from Lemma 2 that the expected time spent per vertex v in step 3 is $O(n^{\frac{5}{6}})$. So the expected time spent for constructing E^* will be $O(n^{\frac{11}{6}} + m)$.

Let us analyze the number of edges of E^*. The first two steps will contribute $O(n^{\frac{4}{3}})$ edges to E^*. It follows from elementary probabilistic analysis that the expected number of vertices of set S' in $ball(v, V, S)$ will be $O(n^{\frac{1}{12}})$. So the expected number of weighted edges contributed to E^* by each vertex $v \in V \backslash S$ is $O(n^{\frac{1}{12}})$. Hence the expected number of edges in E^* will be $O(n^{\frac{13}{12}} + n^{\frac{4}{3}}) = O(n^{\frac{4}{3}})$. We repeat the above algorithm if $|E^*|$ exceeds twice its expected value; it follows from Markov inequality that the expected number of repetitions needed will be at most 2. So we can conclude that the above algorithm can be made to run in expected $O(m + n^{\frac{11}{6}})$ time to compute emulator (V, E^*) with $|E^*| = O(n^{\frac{4}{3}})$.

It follows from the second step in the above algorithm that (V, E^*) is a $(3, 2)$-emulator of the given graph. Furthermore, it *nearly* preserves exact distance from every vertex u to all the vertices with in $ball(u, V, S)$. The following Lemma formalizes this fact.

Lemma 7. *For each vertex u and any vertex $v \in ball(u, V, S)$, the distance $\delta^*(u, v)$ in the emulator (V, E^*) is at least $\delta(u, v)$ and at most $\delta(u, v) + 4$*

Proof. Consider the shortest path between u and v in the original graph. If $v \in S'$, it follows that $\delta^*(u, v) = \delta(u, v)$. Otherwise, let w be the vertex nearest from v(excluding v) on this path which belongs to $\mathcal{N}(S')$. If no such w exists,

then the entire path is preserved in the emulator as follows from Theorem 3(*1*). Otherwise it follows from Theorem 3(*2*) that there is a vertex $x \in S'$ such that the edge (x, w) is present in the $(3, 2)$-spanner (and hence in the emulator too). Note that x must belong to $ball(u, V, S)$ also, therefore, there is a weighted edge (u, x) in the emulator with weight $\delta(u, x)$. Now there are two cases.

1. $(w, v) \in E$: In this case, $\delta(u, x) \leq \delta(u, v)$. Now consider the sub-case when $v \in \mathcal{N}(S')$, it follows from Theorem 3(*2*) that for some vertex $c(v) \in S'$, the edge $(v, c(v))$ as well as a path between x and $c(v)$ of length at most 3 is present in the spanner, and hence in the emulator. This implies that $\delta^*(u, v) \leq \delta(u, x) + \delta^*(x, c(v)) + \delta(c(v), v) \leq \delta(u, v) + 4$.
   If $v \notin \mathcal{N}(S')$ then the edge (w, v) is preserved in emulator as follows from Theorem 3(*1*). Thus the distance $\delta^*(u, v) \leq \delta(u, x) + 2 \leq \delta(u, w) + 3 \leq \delta(u, v) + 2$.
2. $(w, v) \notin E$: In this case, $\delta(u, x) \leq \delta(u, w) + 1$. Furthermore, it follows from definition of w and Theorem 3(*1*) that the entire path between w and v is preserved in the spanner, and hence in the emulator. The weighted edge (u, x), the edge (x, w) and the $w \leftrightarrow v$ path constitute a path in the emulator of length at most $\delta(u, v) + 2$.

Now we shall describe the preprocessing algorithm for the $(3, c)$-approximate distance oracle.

3.2 Preprocessing and Query Algorithm for $(3, c)$-Oracle

The following is the preprocessing algorithm for $(3, c)$-oracle.

1. Choose a random sample $S \subseteq V$ by picking each vertex independently with probability $n^{-5/12}$.
2. Build a 3-approximate distance oracle $\mathcal{O}_3^{G_S}$ using Thorup and Zwick [12] for the subgraph $G_S = (V, \mathcal{E}_S)$.
3. For each $s \in S$, perform Dijkstra's single source shortest path (SSSP) algorithm on emulator (V, E^*). Let $Dist$ be the matrix which stores shortest distance between each pair of vertices from set S in the emulator.
4. Compute $p_S(u)$ and $\delta(u, S)$ for all $u \in V \backslash S$; Let $\mathcal{R}$ be an array such that $\mathcal{R}[u] = \delta(u, S)$.

The $(3, c)$-oracle will consist of $\mathcal{O}_3^{G_S}$, matrix $Dist[]$, and array $\mathcal{R}[]$. We now describe the query algorithm.

Query(u, v) {
  $d_1 \leftarrow \mathcal{O}_3^{G_S}(u, v)$.
  $d_2 \leftarrow \mathcal{R}[u] + \mathcal{R}[v] + Dist[p_S(u), p_S(v)]$.
return $\min(d_1, d_2)$ }

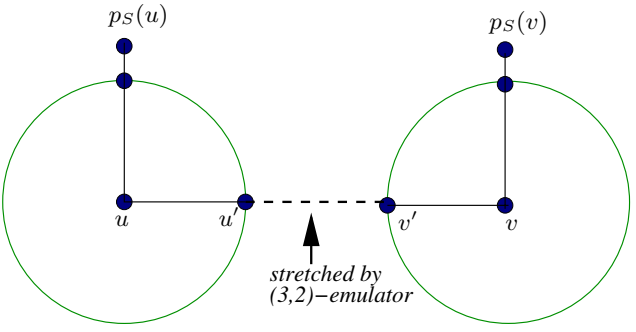


Fig. 1. Analyzing the $(3, c)$-approximate distance oracle when there are at least two consecutive vertices from $\mathcal{N}(S)$ on the shortest path between u and v

3.3 Analysis of the New Oracle

Let us first analyze the stretch of the approximate distance reported by **Query**(u, v) for any two vertices $u, v \in V$. If the shortest path between u and v in G doesn't contain two consecutive vertices in $\mathcal{N}(S)$, then from Lemma 4, it follows that this shortest path is captured in the sparse subgraph $G_S = (V, \mathcal{E}_S)$ as well. It follows from Theorem 1 that the data structure $\mathcal{O}_3^{G_S}$ will report a stretch-3 estimate of shortest distance, and we are done. Let us consider the case where the shortest path between u and v contains two or more consecutive vertices from $\mathcal{N}(S)$. It follows from Lemma 5 that $\mathcal{R}[u] + \mathcal{R}[v] \leq \delta(u, v) + 1$. In this case $ball(u, V, S)$ and $ball(v, V, S)$ will be non-overlapping (see Figure 1). We divide the shortest path between u and v into 3 separate sub-paths : the path $u \leftrightarrow u'$ lying inside $ball(u, V, S)$, the path $v' \leftrightarrow v$ lying inside $ball(v, V, S)$, and the path $u' \leftrightarrow v'$ not covered by either of the two balls. It follows from Lemma 7 that $\delta^*(u, u') \leq \delta(u, u') + 4 = (\delta(u, S) - 1) + 4 = \mathcal{R}[u] + 3$, and similarly $\delta^*(v', v) \leq \mathcal{R}[v] + 3$. Moreover, since the emulator has stretch $(3, 2)$, so

$$\begin{aligned}\delta^*(u', v') &\leq 3\delta(u', v') + 2 \leq 3(\delta(u, v) - \mathcal{R}[u] - \mathcal{R}[v] + 2) + 2 \\ &= 3\delta(u, v) - 3\mathcal{R}[u] - 3\mathcal{R}[v] + 8\end{aligned}$$

Combining the length of the three sub-paths in the emulator, it follows that that $\delta^*(u, v) \leq 3\delta(u, v) - 2\mathcal{R}[u] - 2\mathcal{R}[v] + 14$. Thus the distance reported by **Query**(u, v) is at most :

$$\mathcal{R}[u] + \mathcal{R}[v] + Dist[p_S(u), p_S(v)] \leq 2\mathcal{R}[u] + 2\mathcal{R}[v] + \delta^*(u, v) \leq 3\delta(u, v) + 14$$

Additive error can be reduced to 8 by a more careful analysis. However, for sake of simplicity, we have omitted the details in this version.

We now analyze the space and preprocessing time of this $(3, c)$-oracle.

Lemma 8. *The size of $(3, c)$-approximate distance oracle is $O(n^{3/2})$.*

Proof. It follows from Theorem 1 that the size of $\mathcal{O}_3^{G_S}$ is $O(n^{3/2})$. The size of S will be concentrated around its expected value of $O(n^{7/12})$ (this is because $|S|$ is sum of n independent Bernoulli random variables, and we can apply Chernoff's bound). Therefore, with high probability, the size of matrix $Dist$ is $|S| \times |S| = O(n^{14/12})$ which is $o(n^{3/2})$. For each $u \in V$ we also store vertex $p_S(u)$ and distance $\mathcal{R}[u]$; this will require additional $O(n)$ space for all the vertices. So the overall space required is concentrated around $O(n^{3/2})$ with high probability. We may rebuild the oracle again from scratch if the space exceeds its expected value by some large constant; using Markov inequality, it follows that the expected number of repetitions will be just a constant.

Lemma 9. *The expected time taken by preprocessing algorithm is* $O(n^{2-\alpha_2}+m)$ *where* $\alpha_2 = \frac{1}{12}$.

Proof. Steps 1 and 4 of the preprocessing algorithm take $O(m)$ time. In step 2, it follows from Theorem 1 that the $\mathcal{O}_3^{G_S}$ construction takes $O(|\mathcal{E}_S| \cdot n^{\frac{1}{2}})$ time, and we know from Lemma 3 that the expected value of $|\mathcal{E}_S|$ is $O(n^{1+\frac{5}{12}})$. Thus constructing $\mathcal{O}_3^{G_S}$ takes expected $O(n^{2-\frac{1}{12}})$ time. Emulator construction takes expected $O(m+n^{\frac{11}{6}})$ time which follows from Lemma 6. In Step 3 of the preprocessing algorithm, we execute Dijkstra's single source shortest path algorithm $\forall s \in S$ on E^* which will take expected $O(|S| \cdot (|E^*| + n \log n))$ time. Also note that $|E^*| = O(n^{\frac{4}{3}})$ (see Lemma 6). Thus the expected time required in step 3 of the preprocessing algorithm is $O(n^{\frac{7}{12}+\frac{4}{3}}) = O(n^{2-\frac{1}{12}})$. Thus the expected running time of the preprocessing algorithm is $O(n^{\frac{23}{12}})$.

We can thus conclude the following Theorem.

Theorem 4. *A given unweighted graphs on n vertices and m edges can be preprocessed in expected $O(m + n^{\frac{23}{12}})$ time to build a $(3, 8)$-approximate distance oracle of size $O(n^{\frac{3}{2}})$.*

4 A $(2k-1, 2)$ Approximate Distance Oracle in $o(n^2)$ Time

In this section we describe a $(2k-1, 2)$-approximate distance oracle, for any integer $k \geq 3$, which can be constructed in expected sub-quadratic time. The space occupied by the oracle is $O(kn^{1+1/k})$.

Let $S \subseteq V$ be formed by selecting each vertex independently with probability $n^{-\frac{1}{2}-\frac{1}{2k(k-1)}}$. Thus the expected size of S is $O(n^{\frac{1}{2}-\frac{1}{2k(k-1)}})$. Just like our $(3, c)$-approximate distance oracle, our preprocessing algorithm for $(2k-1, 2)$-approximate distance oracle will employ an emulator (V, E^*). However, this emulator will be a proper sub graph of the original graph G as can be observed from its construction described below.

Constructing emulator (V, E^*)

1. Add the edges of $\mathcal{T}_S$ (see Lemma 1)to E^*.
2. Compute a $(2k-3)$-spanner *span* of size $O(n^{1+\frac{1}{k-1}})$ for the given graph using $O(m)$ time algorithm of Theorem 2. Add all edges of *span* to E^*.

Lemma 10. *The subgraph (V, E^*) output by the above algorithm is an $(2k-3)$-emulator of size $O(n^{1+\frac{1}{k-1}})$. It is computed in $O(m)$ time. Furthermore, this emulator preserves distance between u and $p_S(u)$ for each $u \in V$.*

4.1 Preprocessing and Query Algorithm for $(2k-1, 2)$-Approximate Distance Oracle

We preprocess the graph to obtain $(2k-1, 2)$-oracle as follows.

1. Choose a random sample $S \subseteq V$ by picking each vertex independently with probability $n^{-\frac{1}{2}-\frac{1}{2k(k-1)}}$.
2. Construct a $(2k-1)$-approximate distance oracle of Thorup and Zwick [12] on $G_S = (V, \mathcal{E}_S)$. Let us denote this data structure by $\mathcal{O}_{2k-1}^{G_S}$.
3. For each $s \in S$, execute SSSP algorithm on emulator (V, E^*) to obtain matrix $Dist$, which stores distance $\delta^*(u, v)$ between u and v in the emulator for all $u, v \in S$.
4. Compute $p_S(u)$ for each $u \in V$, and construct an array $\mathcal{R}$ such that $\mathcal{R}[u] = \delta(u, S)$ for each $u \in V$.

The oracle will consist of $\mathcal{O}_{2k-1}^{G_S}$, matrix $Dist[]$, and array $\mathcal{R}[]$. We now describe the query algorithm.

Query(u, v) {
  $d_1 \leftarrow \mathcal{O}_{2k-1}^{G_S}(u, v)$.
  $d_2 \leftarrow \mathcal{R}[u] + \mathcal{R}[v] + Dist[p_S(u), p_S(v)]$.
return $\min(d_1, d_2)$ }

4.2 Analysis of Stretch, Space and Preprocessing Time of Oracle

Let us analyze the distance reported by **Query**(u, v) for any two vertices $u, v \in V$. We consider the two cases to prove the stretch bound as follows:

1. If the shortest path between u and v doesn't contain two or more consecutive vertices in $\mathcal{N}(S)$, then it follows from Lemma 4 that this shortest path is captured in the sparse subgraph $G_S = (V, \mathcal{E}_S)$ as well. By Theorem 1, the data structure $\mathcal{O}_{2k-1}^{G_S}$ will report a stretch-$(2k-1)$ estimate of the shortest distance between u and v.

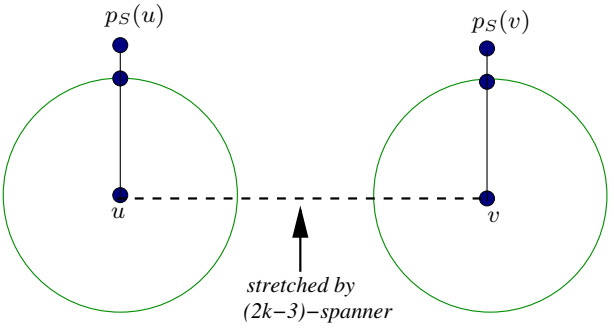


Fig. 2. Proving stretch bound in $(2k-1,2)$-approximate distance oracle

2. If the shortest path between u and v contains two or more consecutive vertices in $\mathcal{N}(S)$, then it follows from Lemma 5 that $\mathcal{R}(u)+\mathcal{R}(v) = \delta(u,S)+\delta(v,S) \leq \delta(u,v)+1$, where $\delta(u,v)$ is the length of the shortest path between u and v in G. In this case, it is easy to observe that $ball(u,V,S)$ and $ball(v,V,S)$ do not overlap (see Figure 2). It follows from Lemma 10 that $\delta^*(u,p_S(u)) = \delta(u,p_S(u))$, and similarly $\delta^*(v,p_S(v)) = \delta(v,p_S(v))$. Hence, $Dist(p_S(u),p_S(v)) \leq \delta(u,S)+\delta(v,S)+\delta^*(u,v)$. Now $\delta^*(u,v) \leq (2k-3)\delta(u,v)$ since (V,E^*) is a $(2k-3)$-emulator. Combining these inequalities and using Lemma 5, the approximate distance returned by **Query**(u,v) can be bounded by :

$$\begin{aligned}\mathcal{R}[u]+\mathcal{R}[v]+Dist[p_S(u),p_S(v)] &\leq 2\delta(u,S)+2\delta(v,S)+\delta^*(u,v)\\ &\leq (2k-1)\delta(u,v)+2\end{aligned}$$

Lemma 11. *The size of the* $(2k-1,2)$*-oracle constructed is* $O(kn^{1+1/k})$.

Proof. It follows from Theorem 1 that $\mathcal{O}^{G_S}_{2k-1}$ uses $O(kn^{1+\frac{1}{k}})$ space. *Dist* matrix will use $|S|^2 = O(n^{1-\frac{1}{k(k-1)}})$ space, and array $\mathcal{R}$ will use $O(n)$ space.

Lemma 12. *The expected time taken in computing a* $(2k-1,2)$*-approximate distance oracle is* $O(kn^{2-\alpha_k}+m)$ *where* $\alpha_k = (\frac{1}{2}-\frac{1}{2k}-\frac{1}{2k-2})$.

Proof. It follows from Theorem 1 that the construction of $\mathcal{O}^{G_S}_{2k-1}$ takes $O(k|\mathcal{E}_S|\cdot n^{\frac{1}{k}})$ time, and we know from Lemma 3 that the expected value of $|\mathcal{E}_S|$ is $O(n^{\frac{3}{2}+\frac{1}{2k(k-1)}})$. So the construction of $\mathcal{O}^{G_S}_{2k-1}$ will take expected $O(kn^{\frac{3}{2}+\frac{1}{2k}+\frac{1}{2k-2}}+m)$ time. In Step 3 of the preprocessing algorithm, the construction of emulator takes $O(m)$ time. We run Dijkstra's SSSP algorithm for each $s \in S$ on $(2k-3)$-emulator (V,E^*) of size $O(n^{1+\frac{1}{(k-1)}})$.[1] This will take

[1] Although this space exceeds $O(n^{1+1/k})$, it is only required during the construction of the oracle and does not contribute to the size of the oracle.

expected $O(|S| \cdot n^{1+\frac{1}{(k-1)}}) = O(n^{\frac{3}{2}+\frac{1}{2k}+\frac{1}{(2k-2)}})$ time. Step 4 will take $O(m)$ time. Thus the expected time complexity of computing $(2k-1,2)$-approximate distance oracle is $O(kn^{\frac{3}{2}+\frac{1}{2k}+\frac{1}{2k-2}}+m)$.

We can thus conclude the following Theorem.

Theorem 5. *For any integer $k > 2$, an unweighted undirected graphs on n vertices and m edges can be preprocessed in expected $O(kn^{\frac{3}{2}+\frac{1}{2k}+\frac{1}{2k-2}}+m)$ time to compute $(2k-1,2)$-approximate distance oracle of $O(kn^{1+\frac{1}{k}})$ size.*

5 Concluding Remarks

For the case of $k \geq 3$, we can obtain further improvements in time bounds by some modifications in the techniques we used. In particular for $k = 3$, we can obtain a $(5,4)$ oracle in $O(n^{11/6}+m)$ time and for $k \geq 4$ we can get a $(2k-1,2k-2)$ oracle in $O(kn^{2+\frac{1}{k}-\lfloor\frac{k}{2}\rfloor\cdot\frac{1}{k}}+m)$ running time, which is better for all even values of k. However, the additive stretch here is not a constant but depends on k.

A more careful analysis of the algorithm for $k \geq 3$ yields a $(2k-1, 2w_{\max}(u,v))$ oracle for weighted graphs where $w_{\max}(u,v)$ is the weight of the heaviest edge between u, v. In fact when edge weights are identical, we obtain the algorithm in the previous section as a special case. Details are straight forward and omitted from this version.

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, 1167–1181 (1999)
2. Baswana, S., Goyal, V., Sen, S.: All-pairs nearly 2-approximate shortest paths in $O(n^2 \operatorname{polylog} n)$ time. In: Proceedings of 22nd Annual Symposium on Theoretical Aspect of Computer Science. LNCS, vol. 3404, pp. 666–679. Springer, Heidelberg (2005)
3. Baswana, S., Kavitha, T.: Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In: Proceedings of the 47th IEEE Annual Symposium on Foundations of Computer Science (FOCS), pp. 591–602 (2006)
4. Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. ACM Transactions on Algorithms 2, 557–577 (2006)
5. Baswana, S., Telikepalli, K., Mehlhorn, K., Pettie, S.: New construction of (α, β)-spanners and purely additive spanners. In: Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 672–681 (2005)
6. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proceedings of 39th Annual ACM Symposium on Theory of Computing, pp. 590–598 (2007)
7. Cohen, E., Zwick, U.: All-pairs small stretch paths. Journal of Algorithms 38, 335–353 (2001)

8. Dor, D., Halperin, S., Zwick, U.: All pairs almost shortest paths. Siam Journal on Computing 29, 1740–1759 (2000)
9. Erdős, P.: Extremal problems in graph theory. In: Theory of Graphs and its Applications (Proc. Sympos. Smolenice,1963), pp. 29–36. House Czechoslovak Acad. Sci, Prague (1964)URL, 29
10. Halperin, S., Zwick, U.: Linear time deterministic algorithm for computing spanners for unweighted graphs (unpublished manuscript) (1996)
11. Roditty, L., Thorup, M., Zwick, U.: Deterministic construction 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)
12. Thorup, M., Zwick, U.: Approximate distance oracles. Journal of Association of Computing Machinery 52, 1–24 (2005)

All-Pairs Shortest Paths with a Sublinear Additive Error

Liam Roditty and Asaf Shapira

[1] Weizmann Institute
[2] Microsoft Research

Abstract. We show that for every $0 \leq p \leq 1$ there is an algorithm with running time of $O(n^{2.575-p/(7.4-2.3p)})$ that given a directed graph with small positive integer weights, estimates the length of the shortest path between every pair of vertices u, v in the graph to within an *additive* error $\delta^p(u, v)$, where $\delta(u, v)$ is the exact length of the shortest path between u and v. This algorithm runs faster than the fastest algorithm for computing exact shortest paths for any $0 < p \leq 1$.

Previously the only way to "bit" the running time of the exact shortest path algorithms was by applying an algorithm of Zwick [FOCS '98] that approximates the shortest path distances within a *multiplicative* error of $(1 + \epsilon)$. Our algorithm thus gives a smooth qualitative and quantitative transition between the fastest *exact* shortest paths algorithm, and the fastest *approximation* algorithm with a linear additive error. In fact, the main ingredient we need in order to obtain the above result, which is also interesting in its own right, is an algorithm for computing $(1 + \epsilon)$ multiplicative approximations for the shortest paths, whose running time is faster than the running time of Zwick's approximation algorithm when $\epsilon \ll 1$ and the graph has small integer weights.

1 Introduction

Computing all-pairs shortest paths (APSP) in graphs is without a doubt one of the most well-studied problems both in practical and theoretical computer-science (see [25] for a recent survey). For arbitrarily dense graphs with real weighted edges, the best algorithm is essentially the classical $O(n^3)$ time algorithm of Floyd-Warshall (see [7]). The first improvement over this algorithm was obtained by Fredman [12] who gave an $O(n^3/(\frac{\log n}{\log \log n})^{1/3})$ time algorithm. Several improvement then followed, culminating in a recent result of Chan [4] who gave an $O(n^3/\frac{\log^2 n}{\log \log n})$ algorithm. The question of whether the APSP problem can be solved in truly subcubic time, that is, in time $O(n^{3-c})$ for some $c > 0$, remains a major open problem.

Besides trying to obtain slight poly-logarithmic improvements over the naive $O(n^3)$ algorithm for general graphs, most of the research on the APSP problem focused on obtaining truely subcubic algorithms for graphs with *small integer edge weights*, where throughout the paper, when we say small integer edge weights we mean edge weights taken from the set $\{-M, \ldots, M\}$, with $M = n^{o(1)}$.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 622–633, 2008.

This problem turns out to be closely related to the problem of fast (that is, subcubic) matrix multiplication algorithms. The first to realize this connection were Alon, Galil and Margalit [3] who showed how to solve the APSP problem in directed graphs with small integer edge weights in time[1] $O(n^{\frac{\omega+3}{2}})$, where $\omega < 2.376$ is the exponent of the fastest algorithm for multiplying two matrices, due to Coppersmith and Winograd [6]. Zwick [24] obtained an improved algorithm that can solve the APSP problem in directed graphs with small integer weights in time $O(n^{2.575})$. Even in the case of unweighted directed graphs, the fastest APSP algorithm is Zwick's $O(n^{2.575})$ time algorithm. The only lower bound on the APSP problem in directed graphs is $\Omega(n^{\omega})$ that comes from the fact that APSP is at least as hard as boolean matrix multiplication. Therefore, even in unweighted directed graphs there is a gap between the $O(n^{2.575})$ upper bound and the $\Omega(n^{\omega})$ lower bound. In fact, if $\omega = 2$ then both the algorithms of [3] and [24] run in $O(n^{2.5})$, so if $\omega = 2$ then the last progress on the APSP problem in directed graph with small weights (or even unweighted) was [3] from 1991. We finally note that the only (general) case where the APSP problem can be solved in time $O(n^{\omega})$ is in the case of *undirected* graphs with small weights, see [15,16,21,20].

Our focus in this paper is on the APSP problem in directed graphs with small positive integer weights. As we have discussed in the previous paragraph, even this spacial case of the problem is not well understood. From the results above, we know that this problem has an upper bound of $O(n^{2.575})$ and a lower bound of $\Omega(n^{\omega})$. A natural question is therefore what can we do faster than we can exactly solve the APSP problem, that is, what can we do in time less than $O(n^{2.575})$? By faster we mean by a factor of n^c. The reason is (i) In most cases we don't even know the exact exponent, (ii) In most cases we disregard $n^{o(1)}$ factors as the fast matrix multiplication algorithms "hide" such factors anyway. To the best of our knowledge, the only result in this direction is an $O(n^{\omega}/\epsilon)$ algorithm of Zwick [24] that approximates the shortest path distances in a directed graph with positive edge weights[2] to within a *multiplicative* error $(1+\epsilon)$. That is, if $\delta(u,v)$ denotes the length of the shortest path connecting u and v, then Zwick's algorithm returns an estimate $\hat{\delta}(u,v)$ satisfying $\delta(u,v) \leq \hat{\delta}(u,v) \leq (1+\epsilon)\delta(u,v)$.

Our motivation for studying better approximations for the APSP problem in directed graphs stems from the fact that if one considers *undirected* graphs, then one can obtain much better approximations. Two notable example are an $O(n^{2.5})$ algorithm of Aingworth et. al. [2] that approximates the distances in *undirected unweighted* graphs to within an *additive* error 2, and an $O(n^2)$ time algorithm of Dor, Halperin and Zwick [9] that approximates the distances in *undirected unweighted* graphs to within an *additive* error $O(\log n)$. It is interesting to note that these two algorithms do *not* use fast matrix multiplication algorithms.

[1] Throughout the paper, with a slight abuse of notation, we use $O(n^r)$ to denote a running time of $O(n^{r+o(1)})$.

[2] We note that as opposed to the previous algorithms we have discussed, Zwick's approximation algorithm has a good *logarithmic* dependence on the size of the edge weights.

Our main result in this paper is that one can also obtain additive approximations in directed graphs. However, they are not as good as those that can be obtained in undirected graphs. We give an algorithm for computing additive approximations that are (arbitrarily) polynomially small in the actual distance between a pair of vertices. In fact our additive approximation gives a "smooth" transition between the fastest exact $O(n^{2.575})$ APSP algorithm and the $O(n^{\omega})$ $(1+\epsilon)$-multiplicative approximation algorithm. En-route we will also improve the running time of Zwick's $(1+\epsilon)$-multiplicative approximation algorithm whenever $\epsilon \ll 1$. We also show that our results are tight in the sense that the only way to obtain approximations, similar in quality to those achievable in undirected graphs, is to actually improve the running time of the fastest *exact* APSP algorithm.

As we have mentioned above, the study of the APSP problem in graphs with small weights is closely related to the problem of designing fast algorithms for matrix multiplication. Before turning to discuss our new results, which also apply fast matrix multiplication algorithms, let us briefly review the relevant results on fast matrix multiplication. Let $\omega(1,r,1)$ be the minimal exponent for which the product of an $n \times n^r$ matrix and an $n^r \times n$ matrix can be computed in $O(n^{\omega(1,r,1)})$ time[3]. The exponent $\omega = \omega(1,1,1)$ is usually called the exponent of fast matrix multiplication. Coppersmith and Winograd [6] proved that $\omega < 2.376$. Coppersmith [5] showed that $\omega(1,0.294,1) = 2$, that is, the product of an $n \times n^{0.294}$ matrix and an $n^{0.294} \times n$ matrix can be computed in $O(n^2)$. Let $\alpha = \sup\{0 \leq r \leq 1 : \omega(1,r,1) = 2\}$, so Coppersmith's [5] result mentioned above can be stated as $\alpha > 0.294$. Although one can trivially obtain bounds on $\omega(1,r,1)$ by breaking the two rectangular matrices into small square ones, Huang and Pan [18] obtained the following improved bound on $\omega(1,r,1)$.

Theorem 1 ([18]). *Let* $\omega = \omega(1,1,1) < 2.376$ *and* $\alpha = \sup\{0 \leq r \leq 1 : \omega(1,r,1) = 2\} > 0.294$. *Then*

$$\omega(1,r,1) = \begin{cases} 2 & 0 \leq r \leq \alpha \\ 2 + \frac{\omega-2}{1-\alpha}(r-\alpha) & \alpha \leq r \leq 1 \end{cases} \tag{1}$$

In particular, for any $0.294 \leq r \leq 1$ *we have* $\omega(1,r,1) \leq 1.843 + 0.532 \cdot r$.

2 The New Results

Let us introduce the main result of this paper. We show that it is possible to compute arbitrary polynomially small additive approximations for the APSP in directed graphs with small weights, and still bit the running time of the fastest exact algorithm that computes APSP.

Theorem 2 (Main Result). *For every* $0 \leq p \leq 1$ *there is a randomized* $O(n^{2.575-\frac{p}{7.4-2.3p}})$ *time algorithm that given a directed graph* $G = (V,E)$, *computes with high probability, for every pair* $u, v \in V$ *a value* $\hat{\delta}(u,v)$ *satisfying*

[3] More generally $\omega(r,s,t)$ denotes the exponent of the fastest algorithm for multiplying an $n^r \times n^s$ matrix with by an $n^s \times n^t$ matrix.

$$\delta(u,v) \leq \hat{\delta}(u,v) \leq \delta(u,v) + \delta^p(u,v) \ .$$

Observe that when $p = 1$ the running time of the above algorithm is $O(n^{2.376}) = O(n^{\omega})$. This corresponds to the case considered in [24] of computing estimates $\hat{\delta}(u,v)$ of the shortest paths satisfying $\delta(u,v) \leq \hat{\delta}(u,v) \leq \delta(u,v) + O(\delta(u,v))$. Also, when $p = 0$, that is when we want to compute exact shortest paths, the running time of the above algorithm is $O(n^{2.575})$, which is the running time of the exact APSP algorithm of [24]. In particular, we get that for any $0 < p \leq 1$ the running time of the algorithm of Theorem 2 is faster than $O(n^{2.575})$ by a polynomial factor. So for any $0 \leq p \leq 1$ Theorem 2 gives a "smooth" transition from the fastest exact APSP algorithm and the fastest approximation with a linear error for directed graphs with small integer weights. We stress again that in this paper we consider the APSP problem in directed graphs with small positive integer weights, while the exact $O(n^{2.575})$ APSP of [24] works also for small *negative* weights, and the approximation algorithm of [24] works also for *large* positive edge weights.

It is also interesting to consider the performance of our algorithm under the assumption that $\omega = 2$. As we briefly explain later, in that case the running time of the algorithm is $O(n^{2+\frac{1-p}{2-p}})$. Note that when $p \approx 1$ this running time becomes $O(n^2)$ and when $p \approx 0$ this running time becomes $O(n^{2.5})$. Recall again that assuming $\omega = 2$ the fastest APSP algorithm runs in $O(n^{2.5})$, so under the assumption that $\omega = 2$ the performance of the algorithm is also faster than that of the fastest exact APSP algorithm for any $0 < p \leq 1$.

We note that the only way one can use previous results in order to obtain approximations with the quality of those given in Theorem 2 is to use the $(1+\epsilon)$-multiplicative approximation algorithm of [24] with $1/n^{1-p}$. However, the running time of this algorithm is $O(n^{\omega+1-p})$ which is slower than the running time of the algorithm we obtain in Theorem 2 for any $0 \leq p < 1$. As we show in the next subsection, when $\epsilon = 1/n^t$ we can improve the running time of the algorithm of [24] but even this improvement does not directly imply the result in Theorem 2 and more ideas are needed.

2.1 An Improved Multiplicative Approximation Algorithm

The main ingredient that we need in order to obtain our main result stated in Theorem 2 is an algorithm for computing $(1+\epsilon)$ multiplicative approximations of the shortest paths in direct graphs with small integer weights. As we will see later, we will need to apply this algorithm when $\epsilon = 1/n^t$ for some $t > 0$, that is when $\epsilon \ll 1$. Zwick's algorithm [24] for computing such approximations runs in time $O(n^{\omega+t})$. The following theorem shows that for such values of ϵ one can obtain a faster algorithm.

Theorem 3. *For every $\epsilon = 1/n^t$ there is a randomized $O(n^{\omega(1,1-t,1)+t})$ time algorithm that given a directed graph $G = (V,E)$, computes with high probability, for every pair $u, v \in V$ a value $\hat{\delta}(u,v)$ satisfying*

$$\delta(u,v) \leq \hat{\delta}(u,v) \leq (1+\epsilon)\delta(u,v) \ .$$

Given the current bounds on $\omega(1,r,1)$ that were given in Theorem 1, we see that when $\epsilon = 1/n^t$ the running time of the algorithm of Theorem 3 is $O(n^{\omega+0.468\cdot t})$, which improves Zwick's algorithm [24] that runs in time $O(n^{\omega+t})$.

The algorithm given in Theorem 3 applies two of the main ideas from [24]. The first is the use of random sampling and rectangular matrix multiplication that was used in [24] in order to obtain an exact APSP algorithm. The second is the idea of *scaling* that was used in [24] in order to obtain an approximate APSP algorithm.

2.2 "Hardness" Results

A natural qualitative question that arises given Theorem 2 is whether one can design approximation algorithms for APSP whose running time is faster than that of the fastest $O(n^{2.575})$ APSP algorithm by some polynomial factor[4] and yet supply better than a polynomially small additive error. For example, given the results of [9,2] on additive approximations in undirected graphs that we have mentioned before, one can ask if for some $c > 0$, it is possible to design an $O(n^{2.575-c})$ time algorithm that will compute estimates $\hat{\delta}(u,v)$ satisfying $\delta(u,v) \leq \hat{\delta}(u,v) \leq \delta(u,v) + O(\log \delta(u,v))$. As the following proposition shows, even if we consider a relaxed version of this problem, where the error is relative to n rather than $\delta(u,v)$, such an algorithm would imply an improvement over the fastest APSP algorithm.

Proposition 1. *Suppose that for some $r \geq \omega$ and $\epsilon \leq \frac{1}{2}$ there is an $O(n^r)$ time algorithm*[5] *that given a directed graph $G = (V,E)$ computes for any pair $u, v \in V$ a value $\hat{\delta}(u,v)$ satisfying $\delta(u,v) \leq \hat{\delta}(u,v) \leq \delta(u,v) + n^\epsilon$. Then there is an $O(n^{r+2r\epsilon})$ time algorithm for the exact APSP problem. In particular, if $\epsilon = o(1)$ and $r = 2.575 - c$ for some positive $c > 0$, then there is also an $O(n^{2.575-c})$ time algorithm for the exact APSP problem.*

So the above proposition implies that if we are interested in an algorithm that runs faster than the best APSP algorithm by a polynomial factor, then unless we improve over the fastest APSP algorithms, all we can hope for is to obtain a polynomially small additive error.

The next proposition gives another such "hardness" result. One can ask if the $O(n^{\omega+0.468\cdot t})$ running time of the algorithm of Theorem 3 can be improved. For simplicity we consider algorithms with running time $O(n^{\omega+\beta t})$ for some $0 < \beta < 1$, and show the following:

Proposition 2. *Suppose that for some $\beta = 0.468 - c$ and for every $\epsilon = 1/n^t$ there is an $O(n^{\omega+\beta t})$ time algorithm that given a directed graph $G = (V,E)$*

[4] Remember that when we measure the running time in the context of fast matrix multiplication based algorithms, we disregard $n^{o(1)}$ factors so we are only interested in running time that is faster by a factor of n^c for some $c > 0$.

[5] We assume (for convenience) that $r \geq \omega$ because this problem is at least as hard as computing transitive closure of a graph, which is equivalent to boolean matrix multiplication.

computes for any pair $u, v \in V$ *a value* $\hat{\delta}(u,v)$ *satisfying* $\delta(u,v) \leq \hat{\delta}(u,v) \leq (1+\epsilon)\delta(u,v)$. *Then there is an* $O(n^{2.575-c/4})$ *time APSP algorithm.*

We note that although the above proposition assumes that the running time of an alleged improved algorithm is of type $O(n^{\omega+\beta t})$, the argument gives similar hardness result for (essentially) any type of running time.

2.3 Organization and Overview

The rest of the paper is organized as follows: in Section 3 we describe the algorithm whose running time was stated in Theorem 3. In Section 4 we apply Theorem 3 in order to obtain the main result of this paper stated in Theorem 2. In Section 4 we also prove Propositions 1 and 2. Section 5 contains some concluding remarks and open problems.

3 The Improved Multiplicative Approximation Algorithm

In this section we consider directed graphs with positive integer weights from the set $\{0,\ldots,M\}$, where $M = n^{o(1)}$. We start with some definitions and a short overview of the approximation algorithm of [24] for computing $(1+\epsilon)$ approximated distances in $O(n^{\omega}/\epsilon)$. We then proceed to present our improved $(1+\epsilon)$ approximation algorithm whose running time is polynomially faster when $\epsilon \ll 1$.

3.1 Computing Distance Products

The computation of shortest paths lengths can be reduced to computation of *min-plus* products:

Definition 1 (Min-plus products). *The min-plus product* $C = (c_{ij}) = A \star B$, *where* $A = (a_{ij})$ *is an* $\ell \times m$ *matrix and* $B = (b_{ij})$ *is an* $m \times n$ *matrix is defined as follows:* $c_{ij} = \min_{k=1}^{m}\{a_{ik} + b_{kj}\}$, *for* $1 \leq i \leq \ell$ *and* $1 \leq j \leq n$.

The min-plus product is known also as *distance product*. If D is a matrix that contains the weights of the edges of a given graph then D^n, the n^{th} power of D with respect to distance product, is the distance matrix of that graph.

The next Lemma was first stated by [3], following a related idea of [23].

Lemma 3. *Let* A *be an* $n \times n^r$ *matrix and let* B *be an* $n^r \times n$ *matrix, both with elements taken from* $\{0,\ldots,M\} \cup \{+\infty\}$. *Then, the distance product* $A \star B$ *can be computed in* $O(Mn^{\omega(1,r,1)})$ *time.*

Based on this Lemma it is possible to use fast matrix multiplication in order to compute distance product. The algorithm **dist-prod** is given in Figure 2 (**fast-prod** is the fast matrix multiplication algorithm). It receives two matrices A and B, where A is an $n \times m$ matrix, B is an $m \times n$ and $m = n^r$. The algorithm returns an $n \times n$ matrix C, where $C = A \star B$. The running time of the algorithm is $O(Mn^{\omega(1,r,1)})$.

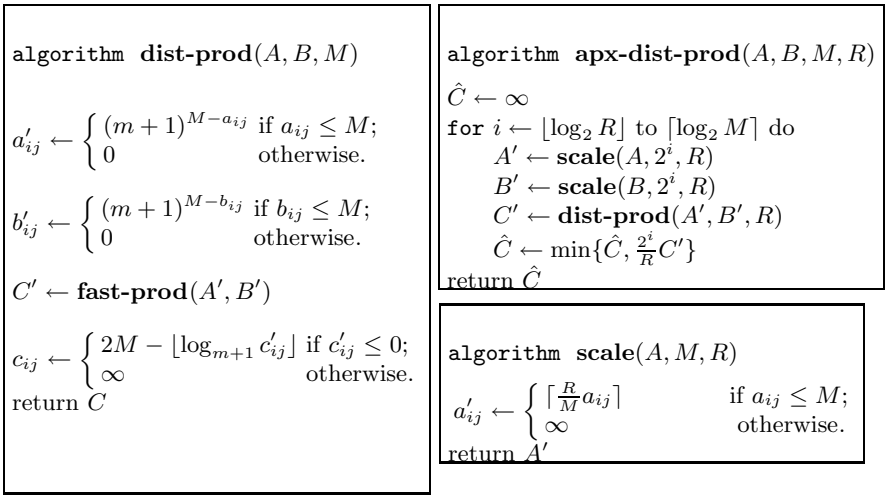


Fig. 1. The algorithm **apx-dist-prod** and its subroutines

3.2 The Approximate Distance Product Algorithm

In this section we present the $1+\epsilon$ approximation algorithm of Zwick from [24] whose running time is $O(n^\omega/\epsilon)$. The algorithm is based on a clever scaling technique.

The main ingredient in the algorithm of [24] is the algorithm **apx-dist-prod**. The algorithm is given in Figure 2. It receives two matrices A and B and computes an approximation of their distance product. The algorithm **apx-dist-prod** uses the algorithm **scale** to scale the elements of A and B each time with a different scale factor. The algorithm **scale** gets a matrix A whose elements are taken from $\{0, \ldots, M\}$ and returns a matrix A' whose elements are the elements of A scaled and rounded up to elements taken from $\{0, \ldots, R\}$.

Next, we restate a Lemma from [24] which shows that the matrix returned by **apx-dist-prod** is a good approximation of $A \star B$. Note that we adjust the Lemma to our future needs by stating it with respect to rectangular matrices and not just to square matrices as it is in [24]. The proof of this Lemma is included in the full version of this paper.

Lemma 4. *Let A be an $n \times n^r$ matrix let B be an $n^r \times n$ matrix, and suppose that elements of A and B that are larger than M are replaced with ∞. Set $C = A \star B$ and let M and R be powers of two. Let $\hat{C}$ be the matrix returned by* **apx-dist-prod**(A, B, M, R)*. Then, $c_{ij} \le \hat{c}_{ij} \le (1 + \frac{4}{R}) c_{ij}$. The running time of* **apx-dist-prod**(A, B, M, R) *is $O(\min\{R, M\} \cdot n^{\omega(1,r,1)} \cdot \log M)$.*

The algorithm of [24] that computes a $(1+\epsilon)$ approximation using the approximated computation of the distance product is given in the left side of Figure 2. Note that is uses **apx-dist-prod** when A, B are *always* square matrices. It sets R to be the first power of two that is greater or equal to $4\lceil \log_2 n \rceil / \ln(1+\epsilon)$. It

follows from Lemma 4 that after $\lceil \log_2 n \rceil$ iterations the stretch of the elements of F is at most:

$$\left(1+\frac{4}{R}\right)^{\lceil \log_2 n \rceil} \leq \left(1+\frac{\ln(1+\epsilon)}{\lceil \log_2 n \rceil}\right)^{\lceil \log_2 n \rceil} \leq 1+\epsilon\,. \tag{2}$$

The total running time of the algorithm is $O(n^\omega/\epsilon)$ so if $\epsilon = 1/n^t$ the running time is $O(n^{\omega+t})$. As it may be clear by now, the main idea of this algorithm is that once we are ready to settle for approximated distances then we can reduce the order of the numbers that are involved in the computation of the distance products. However, the dependency in ϵ is still large. Notice also that the matrices A' and B' which **apx-dist-prod** passes to **dist-prod** are squared matrices. Thus, the main ingredient that [24] uses in order to obtain the speedup in his exact algorithm is not used here and the distance product is done between two square matrices. This is exactly the ingredient that we use in order to reduce the dependency in ϵ. In particular, we use the bridging sets technique of [24] to obtain our improved algorithm. The improved algorithm is given in the right side of Figure 2.

The algorithm sets R to be the first power of two that is greater or equal to $4\lceil \log_{3/2} n \rceil / \ln(1+\epsilon)$. As opposed to the approximation algorithm of [24] (and similarly to the exact algorithm of [24]) our algorithm is composed of $\lceil \log_{3/2} n \rceil$ iterations instead of $\lceil \log_2 n \rceil$. In the ℓ^{th} iteration, a random subset B_ℓ of V of size $\min\{n, (9n\ln n)/(3/2)^\ell\}$ is chosen. Notice that $B_\ell = V$ in the first $\log_{3/2}(9\ln n) = O(\log\log n)$ iterations. However, from that point onwards, the size of B_ℓ shrinks at each iteration by a factor of 2/3.

In the ℓ^{th} iteration the algorithm computes the approximate distance product between $F[V,B]$ and $F[B,V]$, that is, the rectangular matrices $F[V,B]$ and $F[B,V]$ obtained by taking only the columns of F that corresponds to B and the rows of F that corresponds to B, respectively. As opposed to the exact algorithm of [24] (and similarly to the approximation algorithm of [24]) our algorithm computes the *approximation* of the distance product. Moreover, as the matrices that **apx-dist-prod** receives are rectangular then the distance product is done by exploiting this fact.

We claim that **apx-shortest-path**$_2$ computes, with high probability, $1+\epsilon$ approximations of the distances in the graph. This follows from the next Lemma, whose proof appears in the full version of this paper.

Lemma 5. *Let $i,j \in V$. If there is a shortest path from i to j in G that uses at most $(3/2)^\ell$ edges, then after the ℓ^{th} iteration, with high probability, $f_{ij} \leq (1+\frac{4}{R})^\ell \delta(i,j)$.*

3.3 Proof of Theorem 3

We start by observing that if $\epsilon = 1/n^t$ then the algorithm **apx-shortest-path**$_2$ uses R of order $O(\log n/\log(1+1/n^t)) = O(n^t)$ (we use the fact that $\log(1+\epsilon) \approx \epsilon$). Consider iteration ℓ of the algorithm and let r be such that $s = (3/2)^\ell = n^{1-r}$, and note that this means that $|B_\ell| = O(n^r)$. The running time of the ℓ^{th}

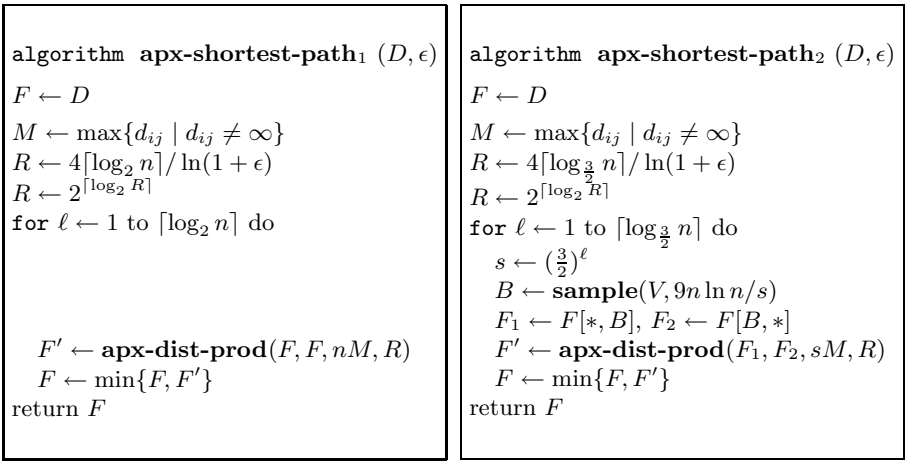


Fig. 2. Zwick's approximation algorithm [24] on the left, and the improved algorithm on the right

iteration of the algorithm is clearly dominated by the cost of **apx-dist-prod**. By Lemma 4, the cost of computing the distance product of A, whose size is $n \times n^r$, and B, whose size is $n^r \times n$, is $O(\min\{M, R\} \cdot n^{\omega(1,r,1)})$, where M is the the largest element that appears in A and B. When our improved algorithm **apx-shortest-path**$_2$ calls **apx-dist-prod** it passes two rectangular matrices A and B, whose sizes are $n \times n^r$ and $n^r \times n$. From the assumption that the graph has small integer weights, it follows that at this stage $sM = O(n^{1-r})$. Hence, the running time of **apx-dist-prod** is $O(\min\{n^{1-r}, n^t\} \cdot n^{\omega(1,r,1)})$. As $\omega(1,r,1) + t$ is increasing with r and $\omega(1,r,1) + 1 - r$ is decreasing with r, we infer that, ignoring poly-logarithmic factors, the worse running time of **dist-prod** is the iteration where $\omega(1,r,1)+t = \omega(1,r,1)+1-r$, that is, when $r = 1-t$. Hence, the running time of the worst iteration is $O(n^{\omega(1,1-t,1)+t})$. The total running time of **apx-shortest-path**$_2$ is also $O(n^{\omega(1,1-t,1)+t})$ as it is dominated by the calls to **dist-prod** and there are only $O(\log n)$ such calls. Combining Lemma 5 with the same argument that is used in (2), we get that after $\lceil \log_{3/2} n \rceil$ iterations the algorithm **apx-shortest-path**$_2$ computes a $(1+\epsilon)$ approximation of the distances in the graph.

4 Proof of Main Result

In this section we prove Theorems 2 as well as Propositions 1 and 2. For the proof of Theorem 2 we will need the following well known fact, whose proof is included in the full version of this paper.

Claim. For every $0 \leq t \leq 1$, there is a randomized $O(n^{3-t})$ time algorithm, that given a directed graph G with small positive integer weights, computes (with

high probability) the shortest path from u to v for every $u, v \in V$ that satisfy $\delta(u,v) \geq n^t$.

Proof of Theorem 2: Let $0.294 \leq \ell \leq 1$ be a value to be chosen later. The key observation is that in order to compute approximations of $\delta(u,v)$ to within an additive error $\delta^p(u,v)$ it is enough to compute the exact shortest paths of G of length at least n^ℓ (as in Claim 4) as well as compute a $(1+1/n^{(1-p)\ell})$-multiplicative approximation of the shortest paths of G (in the sense of Theorem 3), and then take, for each pair u, v, the best of the two values as the estimate of the shortest paths in G. Indeed, if $\delta(i,j) \geq n^\ell$ then we can get the *exact* value of $\delta(u,v)$ via Claim 4. If $\delta(i,j) \leq n^\ell$ then the $(1+1/n^{(1-p)\ell})$-multiplicative approximation of Theorem 3 returns a value $\delta(u,v) \leq \hat{\delta}(u,v) \leq \delta(u,v)+\delta^p(u,v)$.

Thus, it remains to choose the value of ℓ that will minimize the total running time of the two algorithms. The running time of the algorithm of Claim 4 is $n^{3-\ell}$ and the running time of the $(1+1/n^{(1-p)\ell})$-multiplicative approximation algorithm of Theorem 3 is $n^{\omega(1,1-(1-p)\ell,1)+(1-p)\ell}$. Note that the running time of the first algorithm decreases with ℓ and the running time of the second algorithm increases with ℓ, so we need to find the solution of the equation

$$\omega(1, 1-(1-p)\ell, 1) + (1-p)\ell = 3 - \ell \,. \tag{3}$$

By Theorem 1 we know that for any $0.294 \leq r \leq 1$ we have $\omega(1,r,1) = 1.843 + 0.532r$. So plugging this into the above equation we get:

$$1.843 + 0.532(1-(1-p)\ell) + (1-p)\ell = 3 - \ell \,,$$

and this is equivalent to $2.376 + 0.468(1-p)\ell = 3 - \ell$. The solution of this equation is $\ell = \frac{0.624}{1.468-0.468p}$. Note that indeed for any $0 \leq p \leq 1$ we have $0.294 \leq \ell \leq 1$. We get that the running time of the algorithm is

$$O(n^{3-\frac{0.624}{1.468-0.468p}}) = O(n^{2.575-\frac{p}{7.4-2.3p}}) \qquad \square$$

We have commented after the statement of Theorem 2 that under the assumption $\omega = 2$ the running time of the algorithm of Theorem 2 is $O(n^{2+\frac{1-p}{2-p}})$. To see this, note that if $\omega = 2$ then equation (3) becomes $2+(1-p)\ell = 3-\ell$, whose solution is $\ell = \frac{1}{2-p}$, giving that the running time is indeed $O(n^{2+\frac{1-p}{2-p}})$. We end this section with the proofs of Propositions 1 and 2.

Proof of Proposition 1: Given an n-vertex graph $G = (V,E)$ let us construct a graph $G' = (V', E')$ on $m = n^{1+2\epsilon}$ vertices, where G' is obtained from G as follows: we replace every vertex v with two vertices v_{in} and v_{out} that are connected by a path of length $n^{2\epsilon}-1$ on vertices $x_1^v, \ldots, x_{n^{2\epsilon}-2}^v$. All the above vertices are distinct, that is, the path connecting v_{in} and v_{out} and the one connecting u_{in} and u_{out} are disjoint. Hence, G' indeed contains $m = n^{1+2\epsilon}$ vertices. Finally for every edge (u,v) of G we have an edge connecting u_{out} to v_{in} in G'. It is not difficult to see that for any $u \neq v$ we have $\delta_{G'}(u_{in}, v_{in}) = n^{2\epsilon} \cdot \delta_G(u,v)$.

Our assumption is that we can compute for all pairs $i, j \in V'$ an estimation $\hat{\delta}(i,j)$ satisfying

$$\delta_{G'}(i,j) \leq \hat{\delta}_{G'}(i,j) \leq \delta_{G'}(i,j) + m^{\epsilon} \leq \delta_{G'}(i,j) + n^{2\epsilon} - 1 \tag{4}$$

in time $O(m^r) = O(n^{r+2r\epsilon})$, where in the rightmost inequality we have used the assumption that $\epsilon \leq \frac{1}{2}$. We claim that this means that within the same time we can compute all the values $\delta_G(u,v)$. Indeed, combining the fact that $\delta_{G'}(u_{in}, v_{in}) = n^{2\epsilon} \cdot \delta_G(u,v)$ with (4) we infer that

$$n^{2\epsilon} \cdot \delta_G(u,v) \leq \hat{\delta}_{G'}(u_{in}, v_{in}) \leq n^{2\epsilon} \cdot \delta_G(u,v) + n^{2\epsilon} - 1 \ .$$

Observe that this means that

$$\delta_G(u,v) = \left\lfloor \frac{\hat{\delta}_{G'}(u_{in}, v_{in})}{n^{2\epsilon}} \right\rfloor \ ,$$

so all the exact distances can be computed with an additional cost of $O(n^2)$ time. □

Proof of Proposition 2: Observe that if we compute estimates $\hat{\delta}(u,v)$ satisfying $\delta(u,v) \leq \hat{\delta}(u,v) \leq (1+\epsilon)\delta(u,v)$, where $\epsilon = 1/n^t$, then in particular we have computed all the shortest paths of length at most n^t. Remember also that by Claim 4 we can compute all the shortest paths of length at least n^t in time n^{3-t}. So if there is an $O(n^{\omega+(0.468-c)t})$ algorithm for computing the multiplicative estimates, then solving $3-t = \omega+(0.468-c)t$ we get that by choosing $t = \frac{3-\omega}{1.468-c}$ we obtain an algorithm for computing exact APSP in time $O(n^{3-\frac{0.624}{1.468-c}}) = O(n^{2.575-c/4})$. □

5 Concluding Remarks and Open Problems

A natural open problem is to improve the running time of the algorithm given in Theorem 1 for any $0 \leq p \leq 1$, that is, to obtain a faster transition between the exact and the multiplicative APSP algorithms. Our algorithm runs in time $O(n^{\omega+c})$ for any $p < 1$. Is there a $p < 1$ for which one can find additive approximation within an error $\delta^p(u,v)$ and still have running time $O(n^{\omega})$?

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM Journal on Computing 28, 1167–1181 (1999)
3. Alon, N., Galil, Z., Margalit, O.: On the exponent of the all pairs shortest path problem. Journal of Computer ans System Sciences 54, 255–262 (1997); Also, Proc. of FOCS 1991

4. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proc. of STOC 2007 (to appear, 2007)
5. Coppersmith, D.: Rectangular matrix multiplication revisited. Journal of Complexity 13, 42–49 (1997)
6. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symbol. Comput. 9, 251–280 (1990)
7. Cormen, T.H., Leisserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. McGraw-Hill, New York (2001)
8. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
9. Dor, D., Halperin, S., Zwick, U.: All pairs almost shortest paths. SIAM Journal on Computing 29, 1740–1759 (2000)
10. Czumaj, A., Kowaluk, M., Lingas, A.: Faster algorithms for finding lowest common ancestors in directed acyclic graphs (manuscript, 2006)
11. Fischer, M.J., Meyer, A.R.: Boolean matrix multiplication and transitive closure. In: Proc. of the 12^{th} Symposium on Switching and Automata Theory, East Lansing, Mich., pp. 129–131 (1971)
12. Fredman, M.L.: New bounds on the complexity of the shortest path problem. SIAM Journal on Computing 5, 49–60 (1976)
13. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34, 596–615 (1987)
14. Furman, M.E.: Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph. Dokl. Akad. Nauk SSSR 11(5), 1252 (1970)
15. Galil, Z., Margalit, O.: All pairs shortest distances for graphs with small integer length edges. Information and Computation 134, 103–139 (1997)
16. Galil, Z., Margalit, O.: All pairs shortest paths for graphs with small integer length edges. Journal of Computer and System Sciences 54, 243–254 (1997)
17. Gabow, H.N., Tarjan, R.E.: Algorithms for two bottleneck optimization problems. Journal of Algorithms 9, 411–417 (1988)
18. Huang, X., Pan, V.Y.: Fast rectangular matrix multiplications and applications. Journal of Complexity 14, 257–299 (1998)
19. Munro, I.: Efficient determination of the strongly connected components and the transitive closure of a graph. Univ. of Toronto, Toronto, Canada (1971) (unpublished manuscript)
20. Seidel, R.: On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. J. Comput. Syst. Sci. 51, 400–403 (1995)
21. Shoshan, A., Zwick, U.: All pairs shortest paths in undirected graphs with integer weights. In: Proc. of FOCS 1999, pp. 605–614 (1999)
22. Thorup, M., Zwick, U.: Approximate distance oracles. Journal of the ACM 52, 1–24 (2005)
23. Yuval, G.: An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. Information Processing Letters 4, 155–156 (1976)
24. Zwick, U.: All-pairs shortest paths using bridging sets and rectangular matrix multiplication. Journal of the ACM 49, 289–317 (2002); Also, Proc. of FOCS 1998
25. Zwick, U.: Exact and approximate distances in graphs - a survey. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 33–48. Springer, Heidelberg (2001)

Simpler Linear-Time Modular Decomposition Via Recursive Factorizing Permutations

Marc Tedder[1], Derek Corneil[1,⋆], Michel Habib[2], and Christophe Paul[3,⋆⋆]

[1] Department of Computer Science, University of Toronto
{mtedder,dgc}@cs.toronto.edu
[2] LIAFA and the University of Paris 7 - Denis Diderot
habib@liafa.jussieu.fr
[3] CNRS - LIRMM, Univ. Montpellier II France (part of this research was conducted while on sabbatical in the School of Computer Science at the University of McGill)
christophe.paul@lirmm.fr

Abstract. Modular decomposition is fundamental for many important problems in algorithmic graph theory including transitive orientation, the recognition of several classes of graphs, and certain combinatorial optimization problems. Accordingly, there has been a drive towards a practical, linear-time algorithm for the problem. This paper posits such an algorithm; we present a linear-time modular decomposition algorithm that proceeds in four straightforward steps. This is achieved by introducing the notion of factorizing permutations to an earlier recursive approach. The only data structure used is an ordered list of trees, and each of the four steps amounts to simple traversals of these trees. Previous algorithms were either exceedingly complicated or resorted to impractical data-structures.

1 Introduction

A natural operation to perform on a graph G is to take one of its vertices, say v, and replace it with another graph G', making v's neighbours universal to the vertices of G'. Modular decomposition is interested in the inverse operation: finding a set of vertices sharing the same neighbours outside the set – that is, finding a *module* – and contracting this module into a single vertex. A graph's modules form a partitive family [2], and as such, define a decomposition scheme for the graph with an associated decomposition tree composed of the graph's *strong modules* – those that don't overlap other modules. To compute this *modular decomposition tree* is to compute the *modular decomposition* (and vice versa), and with its succinct representation of a graph's structure, its computation is often a first-step in many algorithms. Indeed, since Gallai first noticed its importance to comparability graphs [12], modular decomposition has

⋆ Marc Tedder and Derek Corneil wish to thank NSERC for partially funding this research.

⋆⋆ Michel Habib and Christophe Paul were supported by the French ANR project ANR-O6-BLAN-0148-01, *Graph Decomposition and Algorithms* (GRAAL).

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 634–645, 2008.

been established as a fundamental tool in algorithmic graph theory. All efficient transitive orientation algorithms make essential use of modular decomposition (e.g., [17]). It is frequently employed in recognizing different families of graphs, including interval graphs [18], permutation graphs [23], and cographs [3]. Furthermore, restricted versions of many combinatorial optimization problems can be efficiently solved using modular decomposition (e.g., [8]). While the papers [18,19,20] provide older surveys of its numerous applications, new uses continue to be found, such as in the areas of graph drawing [22] and bioinformatics [11].

Not surprisingly, the problem of computing the modular decomposition has received considerable attention. Much like planarity testing and interval graph recognition, the importance of the problem has bent efforts toward a simple and efficient solution. The first polynomial-time algorithm appeared in the early 1970's and ran in time $O(n^4)$ [5]. Incremental improvements were made over the years – [14,21], for example – culminating in 1994 with the first linear-time algorithms, developed independently by McConnell and Spinrad [16], and Cournier and Habib [4]. These are unfortunately so complex as to be viewed primarily as theoretical contributions. Subsequent algorithms have not been much better.

The attempts made in [17] and [7] are illustrative. Both adopt an approach pioneered by Ehrenfeucht et. al. [9], later improved upon by Dahlhaus [6]. The idea is to pick an arbitrary vertex, say x, and recursively compute the modular decomposition tree for its neighbourhood, $N(x)$, and its non-neighbourhood, which we denote $\overline{N(x)}$. Any strong module not containing x must be a module of either $G[N(x)]$ or $G[\overline{N(x)}]$, and therefore can be extracted from their recursively computed modular decomposition trees. Once extracted, these can then be used to compute the strong modules containing x. The two types of modules are then assembled to form the tree. While conceptually simple, identifying the strong modules containing x and then constructing the tree has proven difficult to perform in linear-time. In [17] they settle for an $O(n + m \log n)$ implementation, while [7] must use conceptually difficult tricks, a careful charging argument, and the challenging Gabow-Tarjan [10] version of union-find.

Factorizing permutations were introduced by Capelle and Habib [1] as a means of avoiding the difficulty just described. A *factorizing permutation* is a permutation of a graph's vertices in which the strong modules appear consecutively. If such a permutation can be computed, then the algorithm of [1] can be used to derive the tree in linear-time. This indirect approach was used in [15] to get an $O(n + m \log n)$ algorithm, and while linear-time was claimed in [13], the paper contains an error which kills the algorithm's simplicity.

In this paper we introduce the notion of factorizing permutations to the recursive framework described above to produce a straightforward linear-time modular decomposition algorithm. The two approaches turn out to be complementary. From the recursively computed trees a factorizing permutation is easily constructed using a refinement technique generalizing traditional partition refinement from sets to trees. We then show how this factorizing permutation makes it easy to identify the strong modules containing x and assemble the tree. We manage to maintain the conceptual simplicity of both, facilitating the proof

of the algorithm's correctness and running time. Moreover, our algorithm avoids sophisticated data structures, using only an ordered list of trees.

1.1 Preliminaries

All graphs in this paper are simple and undirected. Connected components will simply be referred to as *components*, while the connected components of the complement will be referred to as *co-components*. We will talk often of an *ordered list of trees*, which will sometimes be referred to as an *ordered forest*. The leaves within this forest will always correspond to the vertices of the graph in question. When we refer to an ordering of these vertices it is with respect to one implicitly defined by the ordered forest; in particular, any pre-ordering of the leaves of each tree, processed from left to right. Thus, the leaves descendent from any one node will always appear consecutively. Note that sometimes a set of vertices will be referred to as a "tree". We do this to streamline the exposition; our intent will become clear.

A *module* is a set of vertices all of whom share the same neighbourhood outside the set. The modular decomposition tree will occasionally be referred to as the *MD tree*. The MD tree can be recursively defined as follows: the root of the tree corresponds to the entire graph; if the graph is disconnected, the root is called *parallel* and its children are the MD trees of its components; if the graph's complement is disconnected, the root is called *series* and its children are the MD trees of the co-components; in all other cases the root is called *prime*[1], and its children are the MD trees of the graph's maximal modules. Recall that the nodes in this tree are the graph's *strong modules*, which are those that don't *overlap* others.

2 Overview of the Algorithm

2.1 Recursion

The algorithm begins by selecting an arbitrary vertex, x, called the *pivot*, and placing its neighbourhood to its left and its non-neighbourhood to its right, giving us the ordered list of trees, $N(x), x, \overline{N(x)}$. Next, the modular decomposition tree for $G[N(x)]$ is recursively computed. As this occurs, with new pivots being selected, the neighbours of these pivots in $\overline{N(x)}$ are "pulled forward" so that afterwards we have the ordered list of trees, $T(N(x)), x, \overline{N_A(x)}, \overline{N_N(x)}$, where $T(N(x))$ is the modular decomposition tree for $G[N(x)]$, and $\overline{N_A(x)}$ is the subset of $\overline{N(x)}$ with at least one neighbour in $N(x)$, and $\overline{N_N(x)}$ is the subset of $\overline{N(x)}$ without neighbours in $N(x)$. The algorithm then recursively computes the modular decomposition tree for $\overline{N_A(x)}$, pulling its neighbours in $\overline{N_N(x)}$ forward in a similar fashion. And so on. Eventually we arrive at the following ordered list of trees:

[1] This definition of prime differs somewhat from that which normally appears in the literature.

$$\underbrace{T(N_0)}_{N(x)}, x, \underbrace{T(N_1), \ldots, T(N_k)}_{\overline{N(x)}}, \tag{1}$$

where the N_i's correspond to the distance layers in a breadth-first-search begun from x, and the $T(N_i)$'s are their modular decomposition trees. We will sometimes refer to the N_i's as *layers*.

The rest of this paper assumes that the graph is connected and thus each vertex in N_i has an edge to N_{i-1} (or x in the case of N_0). When the graph is disconnected, the layers up to N_{k-1} along with x form one of its connected components. In this case the algorithm builds the MD tree for this component as described below, then unifies the result with $T(N_k)$ under a common root labeled parallel. This adds a constant amount of work to each stage. Each stage is defined by a pivot, and vertices are only pivots once, so this work is consistent with linear-time.

2.2 Refinement

We wish to transform the above ordered list of trees into a factorizing permutation that will simplify the construction of the modular decomposition tree. We begin doing so by refining the trees using the edges active at this stage:

Definition 1. *An edge becomes* active *when one of its endpoints is pivot or if its endpoints reside in different layers.*

The refinement procedure, which generalizes traditional partition refinement from sets to trees, is detailed in section 3.1. In it we process each vertex in turn and use its incident active edges to refine the trees other than its own. What results is not a factorizing permutation but something very close.

To see why, first consider a strong module not containing x, say M. Notice that for some N_i, we have $M \subseteq N_i$, with M also a module in $G[N_i]$. A theorem of [19] says that either M is a strong module in $G[N_i]$, and thus an internal node in $T(N_i)$, or it corresponds to the union of siblings in $T(N_i)$. In the former case, refinement leaves the node corresponding to M unaffected. In the latter case, refinement groups the siblings under a new internal node inserted into $T(N_i)$ in their former location, in a sense "splitting" their former parent in two. Thus:

Lemma 1 (Proved in section 3.1). *The strong modules not containing x appear consecutively after refinement.*

We are not so fortunate for strong modules containing x, although refinement does get them close to appearing consecutively. As described above, refinement groups siblings under new internal nodes. When these new nodes are at depth-1, however, refinement deletes their parent, making that new node a root of its own tree in our ordered list, effectively splitting the siblings' old tree in two. The intuition here comes from the fact that the (co)-components of the layers correspond either to the nodes at depth-0 or depth-1 in the $T(N_i)$'s, combined with the special role played by these (co)-components:

Proposition 1. *If C is a co-component of $G[N_0]$ and M' is a strong module containing x, then either $C \subset M'$ or $C \cap M' = \emptyset$. Similarly for C a component of $G[N_i], i > 0$.*

During refinement the module will be contained within an interval of trees:

Lemma 2 (Proved in section 3.1). *Let $T_k, \ldots, T_1, x, T'_1, \ldots, T'_\ell$ be the ordered forest at some point during refinement, and let M' be a strong module containing x. Then there are trees T_i and T'_j (called the* bounding trees *for M') such that,*

(i) $M' \supset T_{i-1} \cup \cdots \cup T_1 \cup \{x\} \cup T'_1 \cup \cdots \cup T'_{j-1}$, *and*
(ii) $M' \subseteq T_i \cup \cdots \cup T_1 \cup \{x\} \cup T'_1 \cup \cdots \cup T'_j$.

The interval bounding the module becomes more and more precise as trees are split. The next stage in the algorithm makes the interval exact.

2.3 Promotion

When siblings are grouped under a new node during refinement it happens because a vertex in a different tree is adjacent to them but not their other siblings. The siblings' former parent cannot therefore correspond to a module; this is also true of all their ancestors. Refinement accounts for this by marking these nodes for deletion. We show in section 3.1 that when refinement has finished, the nodes without marked children will correspond to the strong modules not containing x. Promotion is the process of deleting all other nodes, – internal nodes are "promoted" upward as their ancestors are deleted – leaving only the strong modules not containing x.

The real benefit of promotion however is that it gives us the desired factorizing permutation. The strong modules not containing x end up consecutive as explained above. But now the strong modules containing x will also be consecutive: as nodes are deleted, the portion of the bounding trees that is in the module will be placed next to the other trees in the module (see lemma 2).

Lemma 3 (Proved in section 3.2). *The ordered forest that results from promotion provides a factorizing permutation.*

2.4 Assembly

In fact, promotion gives us much more than a factorizing permutation: we have an ordered list of trees whose nodes (excepting x) correspond to the strong modules not containing x; moreover, each of these strong modules is itself properly decomposed (their parts were originally in their respective $T(N_i)$'s, and neither refinement nor promotion changes this). What remains, then, is to identify the strong modules containing x, determine the trees in our list constituting them, then use this information to assemble the modular decomposition tree. This was the bottleneck encountered by the previous recursive algorithms. Our factorizing permutation makes it easy.

With a factorizing permutation we know the strong modules containing x are nested: $[\cdots[\cdots[\cdots x\cdots]\cdots]\cdots]$. Since our ordered forest consists of the strong modules not containing x, no tree in it overlaps these brackets. So to build the MD tree, it suffices to insert the brackets between the trees in our list: once this is done, a node is made for each pair of brackets and a "spine" for the MD tree is built; to this we merely affix the trees in our list according to the placement of the brackets. We show how to insert the brackets in section 3.3.

3 Details and Correctness

3.1 Refinement

The refinement process described in the overview is given by algorithm 1; note that it requires algorithm 2 which appears afterwards.

Algorithm 1. Refinement of the ordered list of trees in (1) by the active edges

```
foreach vertex v do
    Let α(v) be its incident active edges;
    Refine the list of trees using α(v) according to algorithm 2, such that:
    if v is to x's left or v is to x's right and refines a tree to x's left then
        refine using left splits according to algorithm 2, and when a node is
        marked, mark it with "left";
    else
        refine using right splits according to algorithm 2, and when a node is
        marked, mark it with "right";
    end
end
```

Below we sketch the proof of lemmas 1 and 2. For the former we actually prove something slightly stronger from which lemma 1 follows immediately:

Lemma 4. *The nodes in the ordered list of trees resulting from refinement that do not have marked children correspond exactly to the strong modules containing x.*

Proof. **[Sketch]** Let M be a strong module not containing x. As stated in the overview, M must be entirely contained in some N_i, and it must be a module of $G[N_i]$. A theorem of [19] guarantees that M is either a node in $T(N_i)$ or the union of children, say $c_1, \ldots, c_k$, of a series or parallel node in $T(N_i)$. Appealing to algorithm 1, we see that in the former case it remains a node throughout refinement and none of its children are ever marked, since each vertex outside $T(N_i)$ is either universal to, or isolated from, the node. Algorithm 1 also makes clear that in the latter case the children will remain siblings throughout refinement, and will not be marked at any time, since, again, each refining vertex is either universal to them or isolated from them. So for contradiction, assume

Algorithm 2. Refinement (using either "left" or "right" splits) of an ordered list of trees by the set X

Let $T_1, \ldots, T_k$ be the maximal subtrees in the forest whose leaves are all in X;
Let $P_1, \ldots, P_\ell$ be the set of parents of the roots of the T_i's;
foreach P_i **do**
  Let A be the set of P_i's children amongst the T_j's, and B its remaining children;
  Let T_a either be the single tree in A or the tree formed by unifying the trees in A under a common root, and define T_b symmetrically;
  When unifying under a common root, assign P_i's label to this root;
  if P_i *is a root* **then**
    Replace P_i in the forest with either T_a, T_b (for a left split) or T_b, T_a (for a right split)
  else
    Replace the children of P_i with T_a and T_b;
  end
  Mark the roots of T_a and T_b and all their ancestors;
  Mark the children of the prime nodes marked above;
end

that after refinement the c_i's have a sibling c different from them. Inspecting algorithm 1, we see that c must have been a sibling of the c_i's in $T(N_i)$, and that c and the c_i's must have the same set of neighbours outside N_i. Hence, $c \cup c_1$ is a module overlapping $c_1 \cup \cdots \cup c_k$, contradicting the latter being strong.

For the converse, consider a node N without any marked children, and suppose N was formed from the refinement of $T(N_i)$. Clearly, the vertices of N have the same neighbours outside $T(N_i)$. By algorithm 1, if N is prime, it existed in $T(N_i)$ and so has the same neighbours within $T(N_i)$. This is also true when N is not prime, since its children must have been children of the same non-prime node in $T(N_i)$. Hence, each node with unmarked children is a module. If the node existed in $T(N_i)$ then it is clearly strong. If it is new, a simple case analysis shows that no other module can overlap it, since two overlapping modules must be a module themselves.

Proof of lemma 2. **[Sketch]** Recall the statement of the lemma, and the bounding trees T_i and T'_j. We prove this by induction on the number of vertices refining. Prior to refinement we have the ordered list of trees $T(N_0), x, T(N_1), \ldots, T(N_k)$. It is easy to show that if $M' \cap N_i \neq \emptyset$ for some $i > 1$, then $M' = V$. Thus, the lemma holds prior to refinement since $T(N_0)$ and either $T(N_1)$ or $T(N_k)$ can be taken as the bounding trees. So suppose there are such bounding trees T_i and T'_j after some number of vertices have refined; now consider what happens after the next vertex refines. Clearly we need only focus on T_i and T'_j; we'll argue the case for T_i, with the case for T'_j being similar.

Now, if T_i is not split we are done, so assume T_i is split and replaced by the trees T_A, T_B in order. Let v be the vertex doing the refining and observe that v is universal to the leaves of T_A and not universal to the leaves of T_B; additionally,

we must have $v \in N_1$. If $v \in M'$ as well, then v is universal to the portion of T_i outside M' and hence we take T_A as the new left-bounding tree. If $v \notin M'$, then it is isolated from the portion of T_i in M', and so we take T_B as the new left-bounding tree in this case.

3.2 Promotion

The promotion process is given by algorithm 3. Below we sketch the proof of lemma 3. The key here is that refinement uses two types of marks, "left" and "right", with promotion handling each differently.

Algorithm 3. The promotion algorithm

while *there is a root r with a child c both marked by "left"* **do**
  Remove from r the subtree rooted at c and place it just before r;
end
while *there is a root r with a child c both marked by "right"* **do**
  Remove from r the subtree rooted at c and place it just after r;
end
Delete all marked roots in the forest with one child, replacing them with that child;
Delete all marked roots in the forest with no children;
Remove all marks;

Proof of lemma 3. **[Sketch]** By lemma 4 and inspection of algorithm 3, we see that the strong modules not containing x will appear consecutively after promotion.

Let M be a strong module containing x. Let T_i and T'_j be the bounding trees provided by lemma 2. It suffices to show that promotion deletes nodes in such a way as to place the portions of T_i and T'_j that are in M next to the other vertices in M. We'll focus on T'_j, with the case for T_i following similarly.

In the proof of lemma 2 we observed that if $M \cap N_i \neq \emptyset$ for some $i > 1$, then $M = V$. As such, we'll assume T'_j is composed of vertices in N_1. If T'_j only contains vertices in M, then clearly we are done since promotion does not rearrange trees in our ordered list. So assume T'_j contains some vertices in M and some outside M. By proposition 1, this means it contains vertices in at least two different components of $G[N_1]$, say C and C' with $C \subset M$ and $C' \cap M = \emptyset$. Now, C and C' were siblings at depth-1 in $T(N_1)$, and by assumption, some portion of each remains in the same tree after refinement. Appealing to algorithm 1, we see that this is only possible if all vertices in C and C' remain in the same tree after refinement; that is, C and C' must still be siblings after refinement, which means they remained siblings throughout refinement.

If both C and C' share the same neighbours outside N_1, then $C \cup C'$ is a module overlapping M, contradicting M being strong. It follows that at least one of C and C' is marked "left" or "right" (or both). We now consider the cases:

Case 1: Assume C' is marked by "left". This means a vertex in N_0 is adjacent to some but not all vertices in C'; let v be the first such vertex. Note that $v \notin M$ if it is adjacent to some of C'; thus, v is universal to C. But we remarked above that C and C' had the same parents throughout refinement; so at the time v refined, it would have split C away from C', contradicting their being siblings afterwards. This case is therefore impossible.

Case 2: Assume C' is marked by "right". Observe that no vertex in C can be adjacent to a vertex in N_i, $i > 1$, since such vertices are outside M and not adjacent to x. Thus C cannot be marked by "right". Thus, promotion places the vertices of C' to the right of those in C.

Case 3: Assume C' is not marked by a split. Then C must be marked by a split, as argued above, and as seen in case 2, it must be a left-split that marks it. Thus, promotion places the vertices of C to the left of those of C'.

In all cases, promotion puts C to the left of C'. Since C and C' were chosen arbitrarily, we can conclude that the vertices of M appear consecutively.

3.3 Assembly

Recall from the overview that to assemble the tree it suffices to insert the brackets delineating the strong modules containing x. We can simplify this by viewing our ordered list of trees as an ordered list of (co-)components. The (co-)components of the layers appear at depth-0 and depth-1 in the $T(N_i)$'s and thus appear consecutively prior to refinement. Examining algorithms 1, 2, and 3 we see that they will remain consecutive after promotion. Thus, our ordered list of trees can be seen as an ordered list of (co-)components: $C'_\kappa, \ldots, C'_1, x, C_1, \ldots, C_\lambda$, where the C'_i's correspond to the co-components of N_0 and the C_i's correspond to the components of the remaining layers. The process to insert the brackets is derived from the lemma below:

Lemma 5. *Let M be the smallest strong module containing x. Then M satisfies one of the following three conditions:*

(i) M is the maximally contiguous module containing x and no C'_i (in which case M is series);
(ii) M is the maximally contiguous module containing x and no C_i, and only C'_j's in N_1 with no edge to their right (in which case M is parallel);
(iii) M is the minimally contiguous module containing x and at least C_1 and C'_1 (in which case M is prime).

We can determine if a C_i has an edge to its right by checking each vertex's incident active edges. To help identify the modules required by the lemma we associate with each (co-)component a μ-value, defined as follows: for C'_i, let C_j be the component with smallest index such that C'_i is isolated from $C_\lambda, \ldots, C_j$, then if $j \neq 1$, $\mu(C'_i) = C_{j-1}$, otherwise $\mu(C'_i) = x$. The μ-values for the C_i's are defined symmetrically.

We apply the lemma by operating greedily. Let M be the smallest strong module containing x. We first check if M is a parallel module by comparing

$\mu(C_1)$ with x; if they are the same, we then check that C_1 has no edge to its right. If both tests succeed then M is a parallel module and we maximally add consecutive C_i's that satisfy both tests. If either test fails then a series module is attempted in a symmetric manner.

Failing both a series and parallel module, we know M is prime and must include C_1 and C_1'. In this case M is formed by iteratively applying the following rule: when a C_i is included in M, so too must be $C_1', \ldots, \mu(C_i)$, and symmetrically for a C_i' added to M. When no new (co-)components can be added we know we have found M. In all cases, once a module is found, brackets are inserted, the module contracted, and the process begins again with the just identified module in the role of x.

4 Running Time and Implementation

4.1 Recursion

In order to effect the partitioning required of the recursion, we need to traverse the pivot's adjacency list in its entirety. However, each vertex is a pivot exactly once during the algorithm, so this is consistent with linear-time.

We will need to isolate the incident active edges of each vertex so that refinement, promotion, and assembly can be performed efficiently; this can be done during the recursion. Initially we assume all vertices are marked as *unvisited* and that each has associated with it an empty list denoted by α (which will be used to store the incident active edges). As pivots are chosen during the recursion they are marked as *visited*. When a pivot's adjacency list is traversed, the pivot is appended to the α-list of all its visited neighbours. Thus, after recursion the α-lists of each vertex in N_i will correspond to their incident active edges to N_{i+1}. The rest of their active edges can then be added by traversing the α-list of each vertex, and appending vertices to the other α-lists in the obvious way. At the end of each stage the α-lists must be cleared to satisfy our induction hypothesis. We can thus assume that the active edges at each stage can be isolated at the cost of work proportional to their number. Notice that each edge is active precisely once during the algorithm, so this effort is consistent with linear-time overall.

4.2 Refinement

A simple recursive marking procedure finds the maximal subtrees required by algorithm 2. All nodes in our trees have at least two children, so the sizes of these subtrees are linear in the number of their leaves, which is equal to the number of incident active edges of the vertex refining. Notice that each vertex has at least one incident active edge. Thus, finding these trees is proportional to the number of active edges at each stage and so is consistent with linear-time.

The children of a prime node need only be marked once, and the ancestors of a node need only be marked twice (once each for "left" and "right"). The time for this marking is therefore proportional to the size of our ordered forest, which is linear in the number of its leaves, which is linear in the number of active

edges (since each leaf has at least one active edge), and hence consistent with linear-time overall.

4.3 Promotion

If we implement promotion in a depth-first manner, we see that it requires no more than a single traversal of our ordered forest, which as just observed, is consistent with linear-time.

4.4 Assembly

Identifying the (co-)components requires at most two traversals of the forest: one prior to refinement to mark them and one after promotion to retrieve them. Determining if a C_i' has an edge to its right needs only a traversal of each vertex's α-list. Computing the μ-values of the (co-)components can be accomplished by processing each vertex in order and traversing its α-list. All this work is therefore consistent with linear-time.

The placement of the brackets amounts to a single traversal of the list of (co-)components, each of which contains an active edge, and so is consistent with linear-time.

The final assembly of the tree can be done merely by traversing our ordered forest, and is therefore consistent with linear-time.

5 Conclusion

Given the fundamental relationship between modular decomposition and transitive orientation, the natural question to ask is whether the ideas here can be applied to the latter problem. In fact, the authors are confident they can. Modular decomposition for directed graphs should also be amenable to this approach. Code for the algorithm can be found at `www.cs.toronto.edu/~mtedder`; the webpage also provides a detailed example of the algorithm's execution.

References

1. Capelle, C., Habib, M., de Montgolfier, F.: Graph decompositions and factorizing permutations. Discrete Mathematics and Theoretical Computer Science 5, 55–70 (2002)
2. Chein, M., Habib, M., Maurer, M.C.: Partitive hypergraphs. Discrete Mathematics 37, 35–50 (1981)
3. Corneil, D.G., Perl, Y., Stewart, L.K.: A linear recognition algorithm for cographs. SIAM Journal of Computing 14, 926–934 (1985)
4. Cournier, A., Habib, M.: A new linear algorithm of modular decomposition. In: Tison, S. (ed.) CAAP 1994. LNCS, vol. 787, pp. 68–84. Springer, Heidelberg (1994)
5. Cowan, D.D., James, L.O., Stanton, R.G.: Graph decomposition for undirected graphs. In: 3rd S-E Conference on Combinatorics, Graph Theory and Computing, Utilitas Math., pp. 281–290 (1972)

6. Dahlhaus, E.: Efficient parallel algorithms for cographs and distance hereditary graphs. Discrete Applied Mathematics 57, 29–54 (1995)
7. Dahlhaus, E., Gustedt, J., McConnell, R.M.: Efficient and practical algorithm for sequential modular decomposition algorithm. Journal of Algorithms 41(2), 360–387 (2001)
8. de Figueiredo, C.M.H., Maffray, F.: Optimizing bull-free perfect graphs. SIAM J. Discret. Math. 18(2), 226–240 (2005)
9. Ehrenfeucht, A., Gabow, H.N., McConnell, R.M., Sullivan, S.L.: An $O(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of two-structures and modular decomposition of graphs. Journal of Algorithms 16, 283–294 (1994)
10. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. In: STOC 1983: Proceedings of the fifteenth annual ACM symposium on Theory of computing, pp. 246–251. ACM Press, New York (1983)
11. Gagneur, J., Krause, R., Bouwmeester, T., Casari, G.: Modular decomposition of protein-protein interaction networks. Genome Biology 5(8), R57 (2004)
12. Gallai, T.: Transitiv orientierbare graphen. Acta Math. Acad. Sci. Hungar. 18, 25–66 (1967)
13. Habib, M., de Montgolfier, F., Paul, C.: A simple linear-time modular decomposition algorithm for graphs, using order extension. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 187–198. Springer, Heidelberg (2004)
14. Habib, M., Maurer, M.C.: On the x-join decomposition of undirected graphs. Discrete Applied Mathematics 1, 201–207 (1979)
15. Habib, M., Paul, C., Viennot, L.: A synthesis on partition refinement: a useful routine for strings, graphs, boolean matrices and automata. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 25–38. Springer, Heidelberg (1998)
16. McConnell, R.M., Spinrad, J.: Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In: 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 536–545 (1994)
17. McConnell, R.M., Spinrad, J.: Ordered vertex partitioning. Discrete Mathematics and Theoretical Computer Science 4, 45–60 (2000)
18. Möhring, R.H.: Algorithmic aspects of comparability graphs and interval graphs. In: Rival, I. (ed.) Graphs and Orders, pp. 41–101. D. Reidel, Boston (1985)
19. Möhring, R.H.: Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions. Annals of Operations Research 4, 195–225 (1985)
20. Möhring, R.H., Radermacher, F.J.: Substitution decomposition for discrete structures and connections with cominatorial optimization. Annals of Discrete Mathematics 19, 257–356 (1984)
21. Muller, J.H., Spinrad, J.: Incremental modular decomposition. Journal of the ACM 36(1), 1–19 (1989)
22. Papadopoulos, C., Voglis, C.: Drawing graphs using modular decomposition. In: Healy, P., Nikolov, N.S. (eds.) Graph Drawing, Limerick, Ireland, September 12-14, 2005, pp. 343–354. Springer, Heidelberg (2006)
23. Pnueli, A., Even, S., Lempel, A.: Transitive orientation of graphs and identification of permutation graphs. Canad. J. Math. 23, 160–175 (1971)

The Complexity of the Counting Constraint Satisfaction Problem

Andrei A. Bulatov

School of Computing Science, Simon Fraser University, Burnaby, Canada
abulatov@cs.sfu.ca

Abstract. The Counting Constraint Satisfaction Problem ($\#\mathrm{CSP}(\mathcal{H})$) over a finite relational structure $\mathcal{H}$ can be expressed as follows: given a relational structure $\mathcal{G}$ over the same vocabulary, determine the number of homomorphisms from $\mathcal{G}$ to $\mathcal{H}$. In this paper we characterize relational structures $\mathcal{H}$ for which $\#\mathrm{CSP}(\mathcal{H})$ can be solved in polynomial time and prove that for all other structures the problem is #P-complete.

1 Introduction

In the Counting Constraint Satisfaction Problem, $\#\mathrm{CSP}(\mathcal{H})$, over a finite relational structures $\mathcal{H}$ the objective is, given a finite relational structure $\mathcal{G}$, to compute the number of homomorphisms from $\mathcal{G}$ to $\mathcal{H}$. Various particular cases of the #CSP arise and have been extensively studied in a wide range of areas from logic, graph theory, and artificial intelligence [11,20,31], to statistical physics [1]. However, in different areas this problem often appears in different equivalent forms: (1) the problem of finding the number of models of a conjunctive formula, (2) the problem of computing the size (number of tuples) of the evaluation $Q(D)$ of a conjunctive query (without projection) Q on a database D and also (3) the problem of counting the number of assignments to a set of variables subject to specified constraints.

Since the seminal papers [29,21], the complexity of the decision counterpart of the #CSP, the CSP, has been an object of intensive study. The ultimate goal of that research direction is to classify finite relational structures with respect to the complexity of the corresponding CSP. We shall refer to this research problem as the *classification problem*. A number of significant results have been obtained, see e.g. [29,21,2,3], but a full classification is far from being completed.

Although the classification problem for the general #CSP has been tackled for the first time very recently, a massive work has been done in the study of the complexity of various particular counting CSPs. These particular problems include classical combinatorial problems such as #CLIQUE, GRAPH RELIABILITY, ANTICHAIN, PERMANENT etc., see, e.g., [31], expressible in the form of the #CSP; the counting SATISFIABILITY and GENERALIZED SATISFIABILITY problems (in these problems the objective is to find the number of satisfying assignments to a propositional formula) [11], which correspond to $\#\mathrm{CSP}(\mathcal{H})$ for 2-element structures $\mathcal{H}$, and many others.

However, the real focus of research in this area has been the $\#H$-COLORING problem and its variants. In the $\#H$-COLORING problem the aim is to find the number of

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 646–661, 2008.

homomorphisms from a given graph G to a fixed graph H. Thus, it is equivalent to $\#\mathrm{CSP}(\mathcal{H})$ where $\mathcal{H}$ is a graph. Dyer and Greenhill [20] proved that, for every undirected graph H, its associated $\#H$-COLORING problem is either in FP or #P-complete and they also provided a complete characterization of the problems in FP. This result has been extended to the counting LIST $\#H$-COLORING problem [15], which allows additional restrictions on possible images of a node. Recently, Dyer, Goldberg, and Paterson [18] obtained a similar classification for directed acyclic graphs. Furthermore, some other variants of the $\#H$-COLORING problem for undirected graphs have been intensively studied during the last few years [13,14]. Another direction in this area is the study of problems with restricted input, that is subproblems of the $\#H$-COLORING problem in which the input graph G must be planar [30], a partial k-tree [16], sparse or of low degree [24,25], etc. Finally, we should mention the approach to counting problems using approximation and randomized algorithms, see e.g. [19,17].

Paper [8] started a systematic study of the classification problem for the general #CSP. The main approach chosen was the *algebraic approach* which has proved to be quite useful in the study of the decision CSP [27,2,3]. This approach uses invariance properties of predicates definable in relational structures. Invariance properties are usually expressed as *polymorphisms* of the predicates, that is (multi-ary) operations on the universe of the relational structure compatible with the predicates.

In [8], we proved that if $\#\mathrm{CSP}(\mathcal{H})$ is in FP, then $\mathcal{H}$ has a *Mal'tsev* polymorphism, that is a ternary operation $m(x, y, z)$ satisfying the identities $m(x, y, y) = m(y, y, x) = x$. Another observation was that the *congruences*, i.e. the definable equivalence relations, of $\mathcal{H}$ play a very important role. In [9], another necessary condition for $\#\mathrm{CSP}(\mathcal{H})$ to belong to FP has been identified. It imposes certain restrictions onto possible definable equivalence relations of $\mathcal{H}$ in terms of the sizes of their equivalence classes.

In this paper, we identify two more necessary conditions for the membership in FP, again expressed in terms of properties of definable equivalence relations; and then prove that, for every relational structure $\mathcal{H}$ satisfying all the conditions obtained, the problem $\#\mathrm{CSP}(\mathcal{H})$ can be solved in polynomial time. Thus, we completely solve the classification problem for the general counting CSP.

We intensively use methods and results from a number of areas of modern algebra: lattice theory, tame congruence theory, commutator theory and ring theory. To make the paper available for a wider audience we try to avoid the use of algebraic terminology as much as possible. However, some of the proofs are more demanding: they require from the reader some familiarity with basic algebraic objects and ideas. Most of them are omitted and can be found in the full version of the paper [6]. The keen reader is referred to textbooks [10,22,23,26]. The reader should be aware that to avoid yet another layer of objects we use algebraic terminology for relational structures, while in the algebraic literature the same concepts are used for "dual" objects, universal algebras.

2 Preliminaries

Logic. A *vocabulary* is a finite set of relational symbols $\sigma = \{R_1, \ldots, R_n\}$ each of which has a fixed arity. A *relational structure* over the vocabulary σ is a tuple $\mathcal{H} = (H; R_1^{\mathcal{H}}, \ldots, R_n^{\mathcal{H}})$ such that H is a non-empty set, called the *universe* of $\mathcal{H}$,

and each $R_i^{\mathcal{H}}$ is a relation on H having the same arity as the symbol R_i. We denote tuples of elements in boldface, e.g., $\mathbf{a}$, and their components by $\mathbf{a}[1], \mathbf{a}[2], \ldots$. Let $\mathcal{G}, \mathcal{H}$ be relational structures over the same vocabulary σ. A *homomorphism* from $\mathcal{G}$ to $\mathcal{H}$ is a mapping $\varphi\colon G \to H$ from the universe of $\mathcal{G}$ (the *instance*) to the universe H of $\mathcal{H}$ (the *template*) such that, for every relation $R^{\mathcal{G}}$ of $\mathcal{G}$ and every tuple $(\mathbf{a}[1], \ldots, \mathbf{a}[m]) \in R^{\mathcal{G}}$, we have $(\varphi(\mathbf{a}[1]), \ldots, \varphi(\mathbf{a}[m])) \in R^{\mathcal{H}}$.

A relation R is said to be *primitive positive* (*pp-*) *definable* in $\mathcal{H}$, if it can be expressed using the predicates $R_i^{\mathcal{H}}$ of $\mathcal{H}$ together with the binary equality predicate on H (denoted Δ_H), conjunction, and existential quantification. We use $\mathsf{def}(\mathcal{H})$ to denote the set of all pp-definable relations.

By $[n]$ we denote the set $\{1, \ldots, n\}$. If $\mathbf{a}$ is an n-ary tuple and $I = \{i_1, \ldots, i_k\} \subseteq [n]$ then $\mathrm{pr}_I \mathbf{a}$ denotes the tuple $(\mathbf{a}[i_1], \ldots, \mathbf{a}[i_k])$. For an n-ary relation R on a set H, $\mathrm{pr}_I R = \{\mathrm{pr}_I \mathbf{a} \mid \mathbf{a} \in R\}$. If $\mathrm{pr}_i R = H$ for all $i \in [n]$, the relation R is said to be a *subdirect power* of H. By $\langle \mathbf{a}, \mathbf{b} \rangle$ we denote a pair of tuples, and by $(\mathbf{a}, \mathbf{b})$ the 'concatenation' of $\mathbf{a}, \mathbf{b}$

#CSP. Let $\mathcal{H}$ be a relational structure. In the *counting constraint satisfaction problem associated with* $\mathcal{H}$ ($\#\mathrm{CSP}(\mathcal{H})$), the objective is, given a structure $\mathcal{G}$ over the same vocabulary, to compute the number of homomorphisms from $\mathcal{G}$ to $\mathcal{H}$.

Example 1. (1) $\#H$-COLORING ([20]). A graph $\mathcal{H}$ is a structure with a vocabulary consisting of one binary symbol R. Then $\#\mathrm{CSP}(\mathcal{H})$ is widely known as the $\#H$-COLORING problem, in which the objective is to compute the number of homomorphisms from a given graph into $\mathcal{H}$.

(2) #3-SAT ([11,12,31]). An instance of the #3-SAT problem is specified by giving a propositional logic formula in CNF, each clause of which contains 3 literals, and asking how many assignments satisfy it. Therefore, #3-SAT is equivalent to $\#\mathrm{CSP}(\mathcal{S}_3)$, where $\mathcal{S}_3$ is the 2-element relational structure with the universe $\{0, 1\}$ and the vocabulary $R_1, \ldots, R_8$, the predicates $R_1^{\mathcal{S}_3}, \ldots, R_8^{\mathcal{S}_3}$ are the 8 predicates expressible by 3-clauses.

(3) Let F be a finite field and let #LINEAR EQUATIONS be the problem of finding the number of solutions to a system of linear equations over F. As is easily seen, any such system can be transformed by adding new variables to a system of equations with at most 3 variables in such a way that the number of solutions remains unchanged. Therefore, #LINEAR EQUATIONS is equivalent to $\#\mathrm{CSP}(\mathcal{E}_3)$, where the universe of the structure $\mathcal{E}_3$ is F, and the relations are those ternary relations that can be represented as the solution space of a linear equation on F.

Every counting CSP belongs to the class #P. However, the exact complexity of $\#\mathrm{CSP}(\mathcal{H})$ strongly depends on the structure $\mathcal{H}$. We say that a relational structure $\mathcal{H}$ is *#-tractable* if $\#\mathrm{CSP}(\mathcal{H})$ is solvable in polynomial time; $\mathcal{H}$ is *#P-complete* if $\#\mathrm{CSP}(\mathcal{H})$ is #P-complete. Note that reductions used in the paper are always Turing reductions. The research problem we deal with is

Problem 1 (classification problem). Characterize #-tractable and #P-complete relational structures.

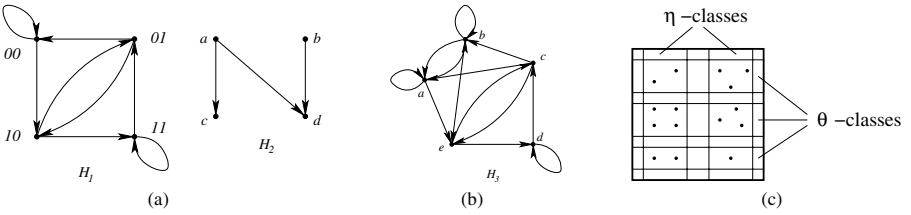


Fig. 1.

Example 2. (1) Dyer and Greenhill [20] proved that if $\mathcal{H}$ is an undirected graph then $\#\mathcal{H}$-COLORING can be solved in polynomial time if and only if every connected component of $\mathcal{H}$ is either a complete bipartite graph, or a complete graph with all loops present, or a single vertex. Otherwise the problem is #P-complete.
(2) A 2-element relational structure $\mathcal{H}$ is #-tractable if and only if every predicate of $\mathcal{H}$ can be represented by a system of linear equations over the 2-element field [11,12]. Otherwise, $\mathcal{H}$ is #P-complete.
(3) $\#\mathrm{CSP}(\mathcal{E}_3)$ is solvable in polynomial time.

Polymorphisms. An (m-ary) operation f *preserves* a relation R (or R is *invariant* under f, or f is a *polymorphism of* R) if for any $\mathbf{a}_1, \ldots, \mathbf{a}_m \in R$ the tuple $f(\mathbf{a}_1, \ldots, \mathbf{a}_m)$, where f acts component-wise, belongs to R. For a given set of operations, C, the set of all relations invariant under every operation from C is denoted by $\mathsf{Inv}(C)$. Conversely, for a relational structure $\mathcal{H}$ we use $\mathsf{Pol}(\mathcal{H})$ to denote the set of all operations preserving every relation of $\mathcal{H}$.

Example 3. Let R be the solution space of a system of linear equations over a field F. Then the operation $m(x, y, z) = x - y + z$ is a polymorphism of R. Indeed, let $A \cdot \mathbf{x} = \mathbf{b}$ be the system defining R, and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$. Then $A \cdot m(\mathbf{x}, \mathbf{y}, \mathbf{z}) = A \cdot (\mathbf{x} - \mathbf{y} + \mathbf{z}) = A \cdot \mathbf{x} - A \cdot \mathbf{y} + A \cdot \mathbf{z} = \mathbf{b}$.

The operation m in Example 3 is an example of a Mal'tsev operation, that is, a ternary operation satisfying the equalities $m(x, y, y) = m(y, y, x) = x$. Every ($n$-ary) relation R invariant under a Mal'tsev operation is *rectangular*: for any $I \subseteq [n]$, for any $\mathbf{a}, \mathbf{b} \in \mathrm{pr}_I R$ and $\mathbf{c}, \mathbf{d} \in \mathrm{pr}_{\underline{n-I}} R$ such that $(\mathbf{a}, \mathbf{c}), (\mathbf{a}, \mathbf{d}), (\mathbf{b}, \mathbf{c}) \in R$, we have $(\mathbf{b}, \mathbf{d}) \in R$.

Example 4. The Mal'tsev operation $m(x, y, z)$ is a polymorphism of the graph H_1 shown in Fig. 1(a), where m is defined as $m(i_1 j_1, i_2 j_2, i_3 j_3) = ij$, $i = i_1$ [$j = j_1$] unless $i_1 = i_2$ [$j_1 = j_2$], in this case $i = i_3$ [$j = j_3$]. The graph H_2 has no Mal'tsev polymorphisms. Indeed, if some $f(x, y, z)$ is a Mal'tsev operation, then $f\left(\begin{pmatrix} a \\ c \end{pmatrix}, \begin{pmatrix} a \\ d \end{pmatrix}, \begin{pmatrix} b \\ d \end{pmatrix}\right) = \begin{pmatrix} b \\ d \end{pmatrix} \notin E(H_2)$.

The following proposition links polymorphisms and pp-definability of relations.

Proposition 1 ([28]). *Let $\mathcal{H}$ be a finite structure, and let $R \subseteq H^r$ be a non-empty relation. Then R is preserved by all polymorphisms of $\mathcal{H}$ if and only if R is pp-definable in* $\mathbf{A}$.

A *universal algebra* is a pair $\mathbb{A} = (A; F)$ where A is a set called the *universe* of $\mathbb{A}$, and F is a set of operations on A called *basic operations*. Any structure $\mathcal{H}$ is related to an algebra $\mathsf{Alg}(\mathcal{H})$ with universe H and the set of basic operations $\mathsf{Pol}(\mathcal{H})$. Conversely, every algebra $\mathbb{A} = (A; F)$ corresponds to a class of structures $\mathsf{Str}(\mathbb{A})$ with universe A and relations from $\mathsf{Inv}(F)$.

Subalgebras and congruences. A *subalgebra* of a relational structure $\mathcal{H}$ is a unary pp-definable relation (that is a subset), and a *congruence* is a pp-definable equivalence relation. For an equivalence relation θ and element $a \in H$, the class of θ containing a is denoted by $a/_\theta$ and the set of all classes of θ by $H/_\theta$.

Example 5. Let $\mathcal{H}$ be an undirected graph. Then the set T of vertices belonging to a triangle is a subalgebra of $\mathcal{H}$, as the following pp-formula witnesses: $T(x) = \exists y, z\, (R(x, y) \wedge R(y, z) \wedge R(z, x))$ (R is the edge relation of the graph).

Example 6. Let $\mathcal{H}$ be a digraph without sources and sinks. Let binary relations θ, η on the vertex set H be defined as follows: $\langle a, b\rangle \in \theta$ if and only if a, b have a common out-neighbour, and $\langle a, b\rangle \in \eta$ if and only if a, b have a common in-neighbour. In general, θ, η are reflexive and symmetric relation. However, if $\mathcal{H}$ has a Mal'tsev polymorphism, they are also transitive. Thus, θ, η are congruences of $\mathcal{H}$. For the graph H_3 shown in Fig. 1(b), the θ-classes are $\{a, b, c\}, \{d, e\}$ and the η-classes are $\{a, b, e\}, \{c, d\}$.

Necessary conditions for #-tractability.

Proposition 2 ([8]). *If $\mathcal{H}$ is a relational structure which is invariant under no Mal'tsev operation then $\mathcal{H}$ is #P-complete.*

Notice that if $\mathcal{H}$ has a Mal'tsev polymorphism then the decision CSP corresponding to $\mathcal{H}$ can be solved in polynomial time [7], see also Section 4.

Let θ, η be congruences of a structure $\mathcal{H}$, where θ, η are incomparable. Let $A_1, \dots, A_k$ and $B_1, \dots, B_\ell$ be θ- and η-classes respectively (see Fig.1(c)). Then $M(\theta, \eta)$ denotes the $k \times \ell$-matrix (m_{ij}), where $m_{ij} = |A_i \cap B_j|$.

Proposition 3 ([9]). *Let $\mathcal{H}$ be a relational structure, and let θ, η be congruences of $\mathcal{H}$. If $\mathsf{rank}(M(\theta, \eta)) > k$, where k is the number of classes in the smallest congruence containing both θ and η, then $\#\mathrm{CSP}(\mathcal{H})$ is #P-complete.*

Example 7. Let $\mathcal{H}$ be the graph H_3 shown in Fig. 1(b), θ and η defined as in Example 6. We have $A_1 = \{a, b, c\}, A_2 = \{e, d\}$, $B_1 = \{a, b, e\}, B_2 = \{c, d\}$ and $M(\theta, \eta) = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$. By Proposition 3, the problem $\#\mathrm{CSP}(H_3)$ is #P-complete. Note also that H_3 has a Mal'tsev polymorphism.

An algebra is said to be *idempotent* if every one of its basic operation f satisfies the condition $f(x, \dots, x) = x$ for any x from the universe. The algebra $\mathsf{Alg}(\mathcal{H})$ is idempotent if and only if all the *constant* relations, $\{(a)\}$, are pp-definable in $\mathcal{H}$. Any algebra $\mathbb{A} = \mathsf{Alg}(\mathcal{H})$ can be converted into an idempotent one, $\mathsf{Id}(\mathbb{A}) = \mathsf{Alg}(\mathcal{H}_{\mathrm{id}})$, where $\mathcal{H}_{\mathrm{id}}$ is the expansion of $\mathcal{H}$ by the constant relations. A *variety* is a class of algebras which, if it contains algebras $\mathbb{A}$, $\mathbb{A}_i$, $i \in I$, for some set I, also contains every subalgebra of

$\mathbb{A}$, every homomorphic image of $\mathbb{A}$, and the direct product of $\mathbb{A}_i$, $i \in I$. The smallest variety containing an algebra $\mathbb{A}$ is called the *variety generated* by $\mathbb{A}$ and denoted by $\mathsf{var}(\mathbb{A})$. For further background on universal algebra, see [10].

In [8] we showed that if $\mathcal{H}_{\mathrm{id}}$ is #P-complete then $\mathcal{H}$ is #P-complete; and if some structure from $\mathsf{Str}(\mathbb{B})$ where $\mathbb{B} \in \mathsf{var}(\mathsf{Alg}(\mathcal{H}))$ is #P-complete then $\mathcal{H}$ is #P-complete. This allows us to strengthen the necessary conditions above.

An algebra $\mathbb{B}$ is said to be *congruence singular* if for every pair of congruences of any $\mathcal{H}' \in \mathsf{Str}(\mathbb{B})$ the condition of Proposition 3 is true. If $\mathcal{H}$ is such that for $\mathbb{A} = \mathsf{Alg}(\mathcal{H}_{\mathrm{id}})$ every $\mathbb{B} \in \mathsf{var}(\mathbb{A})$ is congruence singular then $\mathcal{H}$ is said to be congruence singular. It is not very hard to see that every congruence singular structure has a Mal'tsev polymorphism. The results mentioned above and Proposition 3 imply that if $\mathcal{H}$ is not congruence singular then it is #P-complete. The main result of this paper is that this necessary condition is also sufficient.

Theorem 1. *For a structure $\mathcal{H}$, the problem $\#\mathrm{CSP}(\mathcal{H})$ is solvable in polynomial time if and only if $\mathcal{H}$ is congruence singular.*

Although all our results are stated in terms of varieties, we do not need the full power of this construction. The members of $\mathsf{var}(\mathsf{Alg}(\mathcal{H}))$ we need can be represented as follows. Let $R \in \mathsf{def}(\mathcal{H})$ be an n-ary relation. It can be viewed as a subalgebra of nth direct power of $\mathcal{H}$ (or $\mathsf{Alg}(\mathcal{H})$). A *congruence on* R is a $2n$-ary relation $Q \in \mathsf{def}(\mathcal{H})$ such that $\mathrm{pr}_{\{1,\dots,n\}}Q = \mathrm{pr}_{\{n+1,\dots,2n\}}Q = R$, and, if Q is treated as a binary relation on R, it is an equivalence relation. In most cases relations and congruences appear as follows. Let $\mathcal{G}$ be a structure over the same vocabulary as $\mathcal{H}$, and let $\Phi(\mathcal{G},\mathcal{H})$ denote the set of homomorphisms from $\mathcal{G}$ to $\mathcal{H}$. This set is an $|\mathcal{G}|$-ary relation pp-definable in $\mathcal{H}$. Let also θ be a congruence of $\mathcal{H}$. Then $\Phi(\mathcal{G},\mathcal{H})/_{\theta^{|\mathcal{G}|}}$ denotes a relation on $H/_\theta$ defined as follows: $\Phi(\mathcal{G},\mathcal{H})/_{\theta^{|\mathcal{G}|}} = \{(\mathbf{a}[1]/_\theta, \dots, \mathbf{a}[|\mathcal{G}|]/_\theta) \mid \mathbf{a} \in \Phi(\mathcal{G},\mathcal{H})\}$.

3 Congruence Lattices and the Structure of Relations

Congruence lattices. In this section we look closer at the family of congruences of a relational structure $\mathcal{H}$. We shall assume that $\mathcal{H}$ has a Mal'tsev polymorphism $m(x,y,z)$. All definitions and results given here are well known in universal algebra [10]. We just reformulate them in terms of relational structures.

The set of all congruences of $\mathcal{H}$ is denoted by $\mathsf{Con}(\mathcal{H})$. Let $\theta, \eta \in \mathsf{Con}(\mathcal{H})$. The intersection of θ and η is again a congruence of $\mathcal{H}$ is denoted $\theta \wedge \eta$. The smallest equivalence relation containing both θ and η is the transitive closure of $\theta \cup \eta$, and again a congruence of $\mathcal{H}$, denoted $\theta \vee \eta$. The set $\mathsf{Con}(\mathcal{H})$ together with the operations $\wedge$ (*meet*) and $\vee$ (*join*) is called the *congruence lattice* of $\mathcal{H}$. The set $\mathsf{Con}(\mathcal{H})$ is naturally ordered with respect to inclusion. The least element of $\mathsf{Con}(\mathcal{H})$ is the equality relation, denoted by Δ, and the greatest element is the full relation, denoted by ∇.

We will deal with congruence lattices of several particular types. A congruence lattice $\mathsf{Con}(\mathcal{H})$ is said to be (a) *modular* if, for any $\theta, \eta, \gamma \in \mathsf{Con}(\mathcal{H})$ such that $\eta \leq \theta$, the equality $\theta \wedge (\eta \vee \gamma) = \eta \vee (\theta \wedge \gamma)$ holds; (b) *meet semi-distributive* if, for any $\theta, \eta, \gamma \in \mathsf{Con}(\mathcal{H})$ such that $\theta \wedge \eta = \theta \wedge \gamma$, the equality $\theta \wedge \eta = \theta \wedge (\eta \vee \gamma)$ holds; (c) *distributive* if for any $\theta, \eta, \gamma \in \mathsf{Con}(\mathcal{H})$, the equality $\theta \wedge (\eta \vee \gamma) = (\theta \wedge \eta) \vee (\theta \wedge \gamma)$

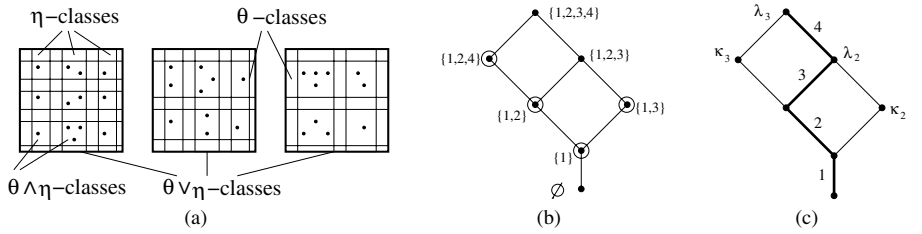


Fig. 2.

holds. Modular and distributive lattices are very well studied, see, e.g. [23, Ch. II, IV]. Some of their properties are implicitely used in the results below. We will also need the following observation that follows from [23, Theorem 2, Ch. II].

Observation 1. *Every modular semi-distributive lattice is distributive.*

Since $\mathcal{H}$ has a Mal'tsev polymorphism, every two members θ, η of $\mathsf{Con}(\mathcal{H})$ must be *permutable*, that is $\theta \circ \eta = \eta \circ \theta$ ($\circ$ denotes the product of binary relations). This means that, for any θ-class A and any η-class B belonging to the same $\theta \vee \eta$-class, $A \cap B$ is non-empty (see Fig.2(a)). If $\mathsf{Con}(\mathcal{H})$ satisfies this condition then it is modular. Thus, all congruence lattices that occur in the rest of the paper are modular.

If every polymorphism of a relational structure $\mathcal{H}$ is idempotent, then, for any congruence θ of $\mathcal{H}$, every θ-class is a subalgebra.

Types of prime quotients. We shall also use some notions and results of tame congruence theory [26]. A pair of congruences θ, η is said to be a *prime quotient* (denoted $\theta \prec \eta$) if $\theta \leq \eta$ and, for any γ such that $\theta \leq \gamma \leq \eta$, either $\gamma = \theta$ or $\gamma = \eta$. Tame congruence theory is a tool to study the local structure of universal algebras and relational structures through certain properties of prime quotients of the congruence lattice. In general, this theory identifies five possible types of such quotients. Fortunately, in our case of relational structures with a Mal'tsev polymorphism, only two of those types can occur, and the definition of these possible types can be significantly simplified.

A prime quotient $\theta \prec \eta$ is said to be of the *affine* type, if, for any η-class B, there is a module M_B with the base set $B/_\theta$ over a ring R_B such that for any $f(x_1, \dots, x_n, y_1, \dots, y_m) \in \mathsf{Pol}(\mathcal{H})$ and $a_1, \dots, a_m \in H$, if the operation $g(x_1, \dots, x_n) = f(x_1, \dots, x_n, a_1, \dots, a_m)$ preserves B, then it can be represented as an operation of the module M_B: $(g\big|_B(x_1, \dots, x_n))/_\alpha = c_1x_1 + \ldots + c_nx_n + a$. In all other cases, $\theta \prec \eta$ has the *Boolean* type. Prime quotients $\theta_1 \prec \eta_1$ and $\theta_2 \prec \eta_2$ are said to be *perspective* if $\eta_1 \vee \theta_2 = \eta_2$, $\eta_1 \wedge \theta_2 = \theta_1$ or $\theta_1 \vee \eta_2 = \eta_1$, $\theta_1 \wedge \eta_2 = \theta_1$. Thus perspectivity is a binary relation on the set of prime quotients of $\mathsf{Con}(\mathcal{H})$. Two quotients that belong to the transitive closure of this relation are said to be *projective* to each other. We shall use Lemma 6.2 from [26] that claims that If $\theta_1 \prec \eta_1$ and $\theta_2 \prec \eta_2$ are projective quotients in $\mathsf{Con}(\mathcal{H})$, then they have the same type. We will sometimes distinguish two cases: when the relational structure omits the affine type, and when the affine type occurs in the structure.

Congruence lattices of structures omitting affine type. If $\mathcal{H}$ omits the affine type then, by Theorem 9.15 of [26], $\mathsf{Con}(\mathcal{H})$ is meet semi-distributive, and by Observation 1 it is distributive. This implies that there is a finite set, K, and an injective mapping $\pi\colon \mathsf{Con}(\mathcal{H}) \to 2^K$ (the set of all subsets) such that $\pi(\theta \vee \eta) = \pi(\theta) \cup \pi(\eta)$ and $\pi(\theta \wedge \eta) = \pi(\theta) \cap \pi(\eta)$. We use the following representation of a set K. Take a maximal chain C in $\mathsf{Con}(\mathcal{H})$, that is, a chain of congruences $\Delta = \theta_0 \prec \theta_1 \prec \ldots \prec \theta_\ell = \nabla$. The set K is defined to be the set of the prime quotients of the chain. A congruence $\theta \in \mathsf{Con}(\mathcal{H})$ corresponds to the set of quotients from K that are projective to the quotients of the form $\gamma \prec \eta \leq \theta$. The bottom end of a prime quotient $\alpha \in \{1, \ldots, \ell\}$ will be denoted by α^-, and the top one by α^+.

Example 8. The lattice shown in Fig. 2(b,c) is distributive, a maximal chain in this lattice, and its representation as a lattice of subsets are also shown.

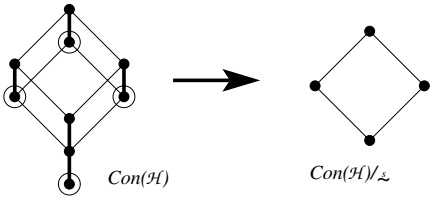
The following proposition comprises properties of $\mathsf{Con}(\mathcal{H})$ that follow easily from the representation of this lattice as a lattice of subsets.

Proposition 4. *(1) Every prime quotient in* $\mathsf{Con}(\mathcal{H})$ *is projective to one and only one of the quotients of* C.
(2) For any $\alpha \in K$, *there is the greatest prime quotient* $\kappa_\alpha \prec \lambda_\alpha$ *projective to* α; *that is, for any* $\eta \prec \theta$ *projective to* α *we have* $\eta \leq \kappa_\alpha$ *and* $\theta \leq \lambda_\alpha$.
(3) For any $\alpha \in K$, κ_α *is* meet-irreducible, *i.e., if* $\kappa_\alpha = \eta \wedge \theta$ *then* $\kappa_\alpha = \eta$ *or* $\kappa_\alpha = \theta$ *(see Fig.2(c)).*

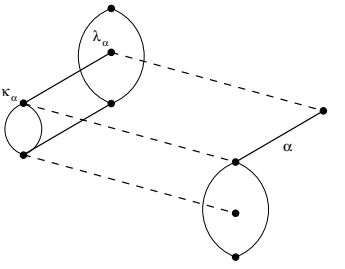
Congruence lattices of structures admitting the affine type. A congruence η is said to be *solvable* over θ if there are $\theta = \theta_1 \prec \ldots \prec \theta_k = \eta$ such that all the prime quotients $\theta_i \prec \theta_{i+1}$ have the affine type. Then $\overset{s}{\sim}$ denotes a binary relation on $\mathsf{Con}(\mathcal{H})$ defined as follows: $\theta \overset{s}{\sim} \eta$ if and only if $\theta \vee \eta$ is solvable over $\theta \wedge \eta$. If $\theta \leq \eta$ then the set of all γ such that $\theta \leq \gamma \leq \eta$ is said to be an *interval* in $\mathsf{Con}(\mathcal{H})$, denoted $[\theta, \eta]$. The next proposition lists some properties of $\overset{s}{\sim}$ that follows from basic facts about modular lattices, Mal'tsev operations, Observation 1, and Lemma 7.4, Theorem 7.7 from [26].

Proposition 5. *(1)* $\overset{s}{\sim}$ *is an equivalence relation and, moreover, a* congruence *of* $\mathsf{Con}(\mathcal{H})$; *that is, for any* $\theta_1, \theta_2, \eta_1, \eta_2 \in \mathsf{Con}(\mathcal{H})$ *such that* $\theta_1 \overset{s}{\sim} \theta_2, \eta_1 \overset{s}{\sim} \eta_2$, *we have* $(\theta_1 \vee \eta_1) \overset{s}{\sim} (\theta_2 \vee \eta_2), (\theta_1 \wedge \eta_1) \overset{s}{\sim} (\theta_2 \wedge \eta_2)$.
(2) Every class S *of* $\overset{s}{\sim}$ *has the greatest* η_S *and the least* θ_S *elements (with respect to* $\leq$*), and equals the interval* $[\theta_S, \eta_S]$. *Every prime quotient inside* S *has the affine type.*
(3) The quotient lattice $\mathcal{L} = \mathsf{Con}(\mathcal{H})/\overset{s}{\sim}$ *is distributive (see Fig.3(a)).*

Proposition 5(3) implies that $\mathcal{L}$ can be represented as a lattice of subsets of a finite set K. Similar to the previous section, K can be chosen to be the set of prime intervals of a maximal chain C in $\mathcal{L}$. Note that the endpoints of $\alpha \in K$ are sets S_1, S_2 of congruences from $\mathsf{Con}(\mathcal{H})$ (S_1 correspods to the bottom end of α). By α^- we denote the greatest element of S_1, and by α^+ the least element of S_2 such that $\alpha^- \leq \alpha^+$. Let $\eta \prec \theta$ be the greatest interval in $\mathcal{L}$ projective to α. Again, η and θ are sets T_1, T_2 of congruences from $\mathsf{Con}(\mathcal{H})$ (T_1 corresponds to β). By κ_α we denote the greatest element of T_1, and λ_α the least element in T_2 such that $\kappa_\alpha \leq \lambda_\alpha$ (see Fig.3(b)). The following result follows easily from basic properties of modular lattices and Lemma 6.2 of [26].



(a) Congruence lattice and its quotient lattice modulo $\overset{s}{\sim}$. Prime quotients of affine type are shown by thick lines; the least elements in the classes of $\overset{s}{\sim}$ are encircled



(b) Congruence lattice and congruences $\kappa_\alpha, \lambda_\alpha$. Solid lines represent prime intervals of Boolean type, ovals represent $\overset{s}{\sim}$-classes

Fig. 3.

Proposition 6. *(1) The interval $[\alpha^-, \alpha^+]$ is perspective to $[\kappa_\alpha, \lambda_\alpha]$*
(2) The intervals $[\alpha^-, \alpha^+]$ and $[\kappa_\alpha, \lambda_\alpha]$ are prime and have the Boolean type.
(3) The congruence κ_α is meet-irreducible.

More necessary conditions of #-tractability. The next two necessary conditions of #-tractability follow from Proposition 3.

Proposition 7. *If $\mathcal{H}$ is congruence singular, then for any congruences $\delta \le \theta \prec \eta \in \mathsf{Con}(\mathcal{H})$ such that $\theta \prec \eta$ has the affine type, any n-ary relation $R \in \mathsf{def}(\mathcal{H})$, and any sequences $A_1, \dots, A_n$ and $B_1, \dots, B_n$ of θ-classes such that A_i, B_i belong to the same η-class ($i \in [n]$), if $R_1 = R \cap (A_1 \times \dots \times A_n) \neq \varnothing$, $R_2 = R \cap (B_1 \times \dots \times B_n) \neq \varnothing$, then $|R_1/_{\delta^n}| = |R_2/_{\delta^n}|$.*

A congruence θ is said to be *uniform* over $\eta \le \theta$ if, for any class B of θ all the η-classes in B have the same size. By Proposition 7 if $\mathcal{H}$ is congruence singular and $\eta \le \theta$ are such that $\eta \overset{s}{\sim} \theta$ then θ is uniform over η.

Let T be a k-dimensional array, that is a collection of numbers $T[i_1, \dots, i_k]$ indexed by tuples $(i_1, \dots, i_k)$, where $1 \le i_k \le m_k$. The array T has rank 1, denoted $\mathsf{rank}(T) = 1$, if for each $\ell \in [k]$, and any $i_1, \dots, i_{\ell-1}, i_{\ell+1}, \dots, i_k, j_1, \dots, j_{\ell-1}, j_{\ell+1}, \dots, j_k$ with $i_u, j_u \in [m_u]$, we have

$$\frac{T[i_1, \dots, i_{\ell-1}, 1, i_{\ell+1}, \dots, i_k]}{T[j_1, \dots, j_{\ell-1}, 1, j_{\ell+1}, \dots, j_k]} = \dots = \frac{T[i_1, \dots, i_{\ell-1}, m_\ell, i_{\ell+1}, \dots, i_k]}{T[j_1, \dots, j_{\ell-1}, m_\ell, j_{\ell+1}, \dots, j_k]}.$$

It is not hard to see that if $k = 2$ then T is a matrix and this condition simply says that T has rank 1 in the usual sense. It also can equivalently be expressed as follows: for each $\ell \in [k]$ there are numbers $t^\ell_1, \dots, t^\ell_{m_\ell}$ such that $T[i_1, \dots, i_k] = t^1_{i_1} \cdot \dots \cdot t^\ell_{i_k}$.

Now let $\mathcal{H}$ be a structure with a Mal'tsev polymorphism and $\gamma_1, \dots, \gamma_k$ its congruences such that for each $i \in [k]$

$$\gamma_i \vee (\gamma_1 \wedge \dots \wedge \gamma_{i-1} \wedge \gamma_{i+1} \wedge \dots \wedge \gamma_k) = \gamma_1 \vee \dots \vee \gamma_k, \tag{1}$$

and let D be a class of $\gamma = \gamma_1 \vee \dots \vee \gamma_k$. Let also $A^i_1, \dots, A^i_{m_i}$ be the classes of γ_i from D. The condition (1) means that for any $i_1, \dots, i_k$ the set $A^1_{i_1} \cap \dots \cap A^k_{i_k}$ is a nonempty

class of $\beta = \gamma_1 \wedge \ldots \wedge \gamma_k$, and any two classes of this form are different. We consider a k-dimensional array $M(D; \gamma_1, \ldots, \gamma_k)$, where $M(D; \gamma_1, \ldots, \gamma_k)[i_1, \ldots, i_k] = |A^1_{i_1} \cap \ldots \cap A^k_{i_k}|$.

Proposition 8. *Let $\gamma_1, \ldots, \gamma_k$ be congruences of a structure $\mathcal{H}$ that has a Mal'tsev polymorphism, let them satisfy the condition (1), and let C be a class of $\gamma_1 \vee \ldots \vee \gamma_k$. Then, $\mathsf{rank}(M(C; \gamma_1, \ldots, \gamma_k)) = 1$ or $\#\mathrm{CSP}(\mathcal{H})$ is #P-complete.*

In the next section we show that a collection of congruences satisfying condition (1) naturally appears when solving counting CSPs.

Structure of relations invariant under a Maltsev operation. Let B_1, B_2 be subalgebras of $\mathcal{H}$ and let θ_1, θ_2 be equivalence relations on B_1, B_2, respectively, pp-definable in $\mathcal{H}$ (we call such relations congruences of subalgebras). Let also φ be a bijective mapping from $B_1/_{\theta_1}$ to $B_2/_{\theta_2}$. The *thick mapping* corresponding to φ is the binary relation $R = \{(a, b) \in B_1 \times B_2 \mid \varphi(a/_{\theta_1}) = b/_{\theta_2}\}$. Any congruence θ is the thick mapping corresponding to the identity mapping on $H/_\theta$.

Lemma 1. *Every binary relation with a Mal'tsev polymorphism is a thick mapping.*

Let $\mathcal{G}$ be a $\#\mathrm{CSP}(\mathcal{H})$ instance and $|\mathcal{G}| = m$. Clearly, the set $\Phi(\mathcal{G}, \mathcal{H})$ can be thought of as an m-ary relation definable in $\mathcal{H}$. We assume that $R = \Phi(\mathcal{G}, \mathcal{H})$ is a subdirect power of $\mathcal{H}$. (Later we show that this is a plausible assumption.) The *quotient structure* $\mathcal{H}/_\theta$ for $\theta \in \mathsf{Con}(\mathcal{H})$ is defined to be $\mathcal{H}/_\theta = (H/_\theta; R^{\mathcal{H}}_1/_\theta, \ldots, R^{\mathcal{H}}_k/_\theta)$ where $R_1, \ldots, R_k$ is the vocabulary of $\mathcal{H}$. Let $\pi \in R/_{\theta^m}$. By $\Phi(\mathcal{G}, \mathcal{H}, \pi)$ we denote the set of all homomorphisms $\varrho \in R$ such that $\varrho/_\theta = \pi$. Recall that for a congruence $\theta \in \mathsf{Con}(\mathcal{H})$ by θ^m we denote the congruence of R such that $\langle \varrho_1, \varrho_2 \rangle \in \theta^m$ if and only if $\langle \varrho_1(g), \varrho_2(g) \rangle \in \theta$ for all $g \in \mathcal{G}$.

Corollary 1. *Assuming that $\mathcal{H}$ is congruence singular, let $\eta \leq \theta \in \mathsf{Con}(\mathcal{H})$ be such that $\eta \overset{s}{\sim} \theta$. Let also $\pi \in \Phi(\mathcal{G}, \mathcal{H})/_{\theta^m}$ and $\varrho_1, \varrho_2 \in \Phi(\mathcal{G}, \mathcal{H})/_{\eta^m}$ be such that $\varrho_1/_\theta = \varrho_2/_\theta = \pi$. Then $|\Phi(\mathcal{G}, \mathcal{H}, \varrho_1)| = |\Phi(\mathcal{G}, \mathcal{H}, \varrho_2)|$.*

For $i, j \in [m]$, by $\psi_{i,j}$ we denote the thick mapping equal to $\mathrm{pr}_{i,j} R$. If it is a thick mapping corresponding to $\varphi\colon \mathcal{H}/_\theta \to \mathcal{H}/_\theta$ for some $\theta \in \mathsf{Con}(\mathcal{H})$, we say that $\psi_{i,j}$ is a *thick mapping of level* θ, and if $\theta = \theta_k$ (from the chain defining K) then $\psi_{i,j}$ is said to be a thick mapping of level k. Let $\eta \in \mathsf{Con}(\mathcal{H})$. By η^* we denote an equivalence relation on the set $[m]$ defined as follows: $\langle i, j \rangle \in \eta^*$ if and only if $\mathrm{pr}_{ij} R$ is a thick mapping of level θ for some $\theta \leq \eta$.

4 Algorithms

Decision CSPs over structures with a Mal'tsev polymorphism. If a relational structure $\mathcal{H}$ has a Mal'tsev polymorphism, then the decision CSP with the template $\mathcal{H}$ can be solved in polynomial time [7]. The algorithm presented in [7] builds a sort of a succinct (polynomial size) representation for the set of all solutions.

Let n be a positive integer, let H be a finite set, let $\mathbf{a}$, $\mathbf{b}$ be n-ary tuples and let (i,a,b) be any element in $[n] \times H^2$. We say that $(\mathbf{a},\mathbf{b})$ *witnesses* (i,a,b) if $\mathrm{pr}_{[i-1]}\mathbf{a} = \mathrm{pr}_{[i-1]}\mathbf{b}$, $\mathbf{a}[i] = a$, and $\mathbf{b}[i] = b$. Let R be any n-ary relation on H. The *signature* of R, $\mathsf{Sig}_R \subseteq [n] \times H^2$, is defined to be the set containing all those $(i,a,b) \in [n] \times H^2$ witnessed by tuples in R, that is $\mathsf{Sig}_R = \{(i,a,b) \in [n] \times H^2 \mid \exists \mathbf{a},\mathbf{b} \in R$ such that $(\mathbf{a},\mathbf{b})$ witnesses $(i,a,b)\}$. Note that if R has a Mal'tsev polymorphism, then the rectangularity of R implies that for any $(i,a,b) \in \mathsf{Sig}_R$ and any $\mathbf{a} \in \mathrm{pr}_{[i]}R$ with $\mathbf{a}[i] = a$ the tuple $\mathbf{b}$ such that $\mathrm{pr}_{[i-1]}\mathbf{b} = \mathrm{pr}_{[i-1]}\mathbf{a}$ and $\mathbf{b}[i] = b$ also belongs to $\mathrm{pr}_{[i]}R$. Another implication of rectangularity is that for any i the binary relation $\{(a,b) \mid (i,a,b) \in \mathsf{Sig}_R\}$ is a congruence on $\mathrm{pr}_i R$. A subset R' of R is called a *representation* of R if $\mathsf{Sig}_{R'} = \mathsf{Sig}_R$. If furthermore, $|R'| \leq 2|\mathsf{Sig}_R|$ then R is called a *compact* representation of R.

Let $\mathcal{H}$ be a relational structure and $R' \subseteq H^n$ for some n. By $\langle R' \rangle_{\mathcal{H}}$ we denote the relation *generated* by R', that is, the smallest relation R pp-definable in $\mathcal{H}$ and such that $R' \subseteq R$. Since $\mathcal{H}$ is usually clear from the context we shall omit this subscript. The key lemma proved in [7] states that if R is a relation pp-definable in a structure with a Mal'tsev polymorphism, and R' is a representation of R, then $\langle R' \rangle = R$. Given an instance $\mathcal{G}$ of the constraint satisfaction problem $\mathrm{CSP}(\mathcal{H})$, $m = |\mathcal{G}|$, the algorithm presented in [7] finds a compact representation of $\Phi(\mathcal{G},\mathcal{H})$ treated as an m-ary pp-definable relation in $\mathcal{H}$.

We will need to know unary and binary projections of the relation $\Phi(\mathcal{G},\mathcal{H})$, that is, sets of the form $\psi_g = \{\varphi(g) \mid \varphi \in \Phi(\mathcal{G},\mathcal{H})\}$ for $g \in \mathcal{G}$ and $\psi_{g,h} = \{(\varphi(g),\varphi(h)) \mid \varphi \in \Phi(\mathcal{G},\mathcal{H})\}$. It is not hard to see (see also [7]) that if R' is a compact representation of $\Phi(\mathcal{G},\mathcal{H})$, then $\psi_g, \psi_{g,h}$ are equal to $\langle \mathrm{pr}_g R' \rangle$ and $\langle \mathrm{pr}_{g,h} R' \rangle$. As $\mathcal{H}$ is fixed, we may assume that the relation generated by any set elements or pairs of elements is known.

Reduction to subdirect powers. In general, for an instance $\mathcal{G}$ of $\#\mathrm{CSP}(\mathcal{H})$, the sets ψ_g, $g \in \mathcal{G}$, are subalgebras of $\mathcal{H}$ that are not necessarily equal to $\mathcal{H}$. We show how to transform the problem so that ψ_g equals $\mathcal{H}$ for all $g \in \mathcal{G}$. To do this we borrow some methods from the multi-sorted CSP, see, e.g., [4].

Let $D_1,\dots,D_n$ be the subalgebras of $\mathcal{H}$. We shall assume that along with every (k-ary) relational symbol R and any $D_{i_1},\dots,D_{i_k}$ the vocabulary of $\mathcal{H}$ contains a symbol R' such that $R'^{\mathcal{H}} = R \cap (D_{i_1} \times \ldots \times D_{i_k})$. Then we define a relational structure $\chi(\mathcal{H})$ as follows. The universe of $\chi(\mathcal{H})$ is $D = D_1 \times \ldots \times D_n$; the ith component of an element $\overline{a} \in D$ is denoted by $\overline{a}[i]$. For any (n-ary) relation R definable in $\mathcal{H}$ we set $(\overline{a}_1,\dots,\overline{a}_n) \in \chi(R)$ if and only if $(\overline{a}_1[i_1],\dots,\overline{a}_n[i_n]) \in R$, where $D_{i_j} = \mathrm{pr}_j R$. In particular, each unary relation of $\chi(\mathcal{H})$ contains all elements of D and, therefore, can be thrown out. For any coordinate position i of any non-unary relation R, the set $\mathrm{pr}_i\chi(R)$ equals D. Finally, to define $\chi(\mathcal{H})$ formally for each relational symbol R we interpret it as $R^{\chi(\mathcal{H})} = \chi(R)$.

To transform an instance $\mathcal{G}$ of $\#\mathrm{CSP}(\mathcal{H})$ into an instance $\mathcal{G}'$ of $\#\mathrm{CSP}(\chi(\mathcal{H}))$ with the same universe, for each relational symbol R and each tuple $(g_1,\dots,g_n) \in R^{\mathcal{G}}$ we include $(g_1,\dots,g_n)$ into $R'^{\mathcal{G}'}$ where R' is such that $R'^{\mathcal{H}} = R^{\mathcal{H}} \cap (\psi_{g_1} \times \ldots \times \psi_{g_n})$. The next easy lemma completes the reduction.

Lemma 2. *Let $\mathcal{G}$ be an instance of $\#\mathrm{CSP}(\mathcal{H})$ and $\mathcal{G}'$ an instance of $\#\mathrm{CSP}(\chi(\mathcal{H}))$ consructed as above. Let also $\psi_g = \mathrm{pr}_g \Phi(\mathcal{G}, \mathcal{H})$ for $g \in \mathcal{G}$. Then $\Phi(\mathcal{G}', \chi(\mathcal{H}))$ is a subdirect power of $\chi(\mathcal{H})$ and $|\Phi(\mathcal{G}', \chi(\mathcal{H}))| = |\Phi(\mathcal{G}, \mathcal{H})| \cdot \prod_{g \in G} \frac{|D|}{|\psi_g|}$.*

The algorithm. Suppose that $\mathcal{H}$ is congruence singular. We recursively compute numbers of the form $|\Phi(\mathcal{G}, \mathcal{H}, \pi)|$ for an instance $\mathcal{G}$. We assume that the universe G of $\mathcal{G}$ is $[m]$. If π is a mapping of level ℓ then $|\Phi(\mathcal{G}, \mathcal{H}, \pi)| = |\Phi(\mathcal{G}, \mathcal{H})|$, and if π is a mapping of level 0 then $|\Phi(\mathcal{G}, \mathcal{H}, \pi)| = 1$. Let $\alpha \in K$ and let π be a mapping from $\mathcal{G}$ to $\mathcal{H}/_{\alpha^+}$. We show how to reduce computing the number $|\Phi(\mathcal{G}, \mathcal{H}, \pi)|$ to computing numbers $|\Phi(\mathcal{G}, \mathcal{H}, \varrho)|$ for certain ϱ, mappings from $\mathcal{G}$ to $\mathcal{H}/_{(\alpha - 1)^+}$.

First we construct a collection of congruences satisfying the condition (1). Let $\alpha \in K$, $A_1, \ldots, A_k$ be the κ^*_α-classes and $g_1, \ldots, g_k$ their representatives. Let also $\pi \in \Phi(\mathcal{G}, \mathcal{H})/_{(\alpha^+)^m}$ and $B^u_1, \ldots, B^u_{s_u}$ be the κ_α-classes from $\pi(g_u)/_{\lambda_\alpha}$, the λ_α-class containing elements from $\pi(g_u)$.

Lemma 3. *There is $J \subseteq [k]$ such that, for any $\pi \in \Phi(\mathcal{G}, \mathcal{H})/_{(\alpha^+)^m}$, there are i_u, $u \in [k] - J$, with $i_u \in [s_u]$, satisfying the following conditions. Every homomorphism $\varrho \in \Phi(\mathcal{G}, \mathcal{H})/_{(\alpha^-)^m}$ with $\varrho/_{(\alpha^+)^m} = \pi$ can be represented as follows: there are i_u for $u \in J$ with $i_u \in [s_u]$ such that $\varrho(g_u) \in B^u_{i_u}$ for $u \in [k]$ and, for any $g \in A_u$, $u \in [k]$, we have*

$$\varrho(g) = \pi(g) \cap \psi_{g_u, g}/_{\kappa^2_\alpha}(B^u_{i_u}).$$

Conversely, for any choice of $B^u_{i_u}$, $u \in J$, the mapping ϱ defined in this way is an element of $\Phi(\mathcal{G}, \mathcal{H})/_{(\alpha^-)^m}$, and $\varrho/_{\alpha^+} = \pi$.

The set J can be found by inspecting the unary projections of $\Phi(\mathcal{G}, \mathcal{H}, \pi)$, which in turn can be found by using a compact representation constructed by the algorithm from [7]. Without loss of generality we assume $J = [q]$. Let $\mathbf{s}$ be a q-tuple such that $\mathbf{s}[u] \in [s_u]$. The mapping defined as in Lemma 3 for the classes $B^1_{\mathbf{s}[1]}, \ldots, B^q_{\mathbf{s}[q]}$ will be denoted by $\varrho_{\mathbf{s}}$. Congruence γ^J_j is defined as follows: $\langle \varrho_1, \varrho_2 \rangle \in \gamma_j$ if and only if $\langle \varrho_1(i), \varrho_2(i) \rangle \in \alpha^-$ for $i \in A_j \cup \bigcup_{v \in [k] - J} A_v$ and $\langle \varrho_1(i), \varrho_2(i) \rangle \in \alpha^+$ otherwise. It is not hard to see that sets $\Phi(\mathcal{G}, \mathcal{H}, \varrho_{\mathbf{s}})$ are the classes of the congruence $(\alpha^-)^m$ on the relation $\Phi(\mathcal{G}, \mathcal{H}, \pi)$. Clearly, $(\alpha^-)^m = \gamma_1 \wedge \ldots \wedge \gamma_q$ and $(\alpha^+)^m = \gamma_1 \vee \ldots \vee \gamma_q$.

Lemma 4. *The congruences γ^J_j, $j \in J$, satisfy the condition (1).*

Let $T(\pi)$ denote the k-dimensional $m_1 \times \ldots \times m_k$ array such that its entry indexed by $\mathbf{s}$ is equal to $|\Phi(\mathcal{G}, \mathcal{H}, \varrho_{\mathbf{s}})|$. By Proposition 8, $T(\pi)$ has rank 1, that is, there are numbers $t^i_1, \ldots, t^i_{m_i}$ such that $|\Phi(\mathcal{G}, \mathcal{H}, \varrho_{\mathbf{s}})| = t^1_{\mathbf{s}[1]} \cdot \ldots \cdot t^k_{\mathbf{s}[k]}$. These numbers t^i_j can be found as follows. Fix a tuple $\mathbf{s}$. By $\mathbf{s}^i_v$ we denote the tuple, all entries of which are equal to the corresponding entries of $\mathbf{s}$, except for the ith entry that is equal to v. Then set

$$t^1_j = |\Phi(\mathcal{G}, \mathcal{H}, \varrho_{\mathbf{s}^1_j})| \quad \text{and} \quad t^i_j = \frac{|\Phi(\mathcal{G}, \mathcal{H}, \varrho_{\mathbf{s}^i_j})|}{|\Phi(\mathcal{G}, \mathcal{H}, \varrho_{\mathbf{s}^1_j})|} \qquad \text{for } i \in \{2, \ldots, k\}.$$

Algorithm `Uniform`
INPUT: an compact representation R'' of $\Phi(\mathcal{G},\mathcal{H},\pi)$, $\pi \in \Phi(\mathcal{G},\mathcal{H})/_{\beta^m}, \beta \overset{s}{\sim} \Delta$
OUTPUT: the cardinality of $\Phi(\mathcal{G},\mathcal{H},\pi)$
Step 1 **set** $N := 1$, $S := R''$, and $\overline{\mathcal{G}} := \mathcal{G}$
Step 2 **for** $g = m$ to 1 **do**
Step 2.1 let θ be a congruence of $\mathcal{H}$ such that $\langle a,b\rangle \in \theta$ if and only if $(g,a,b) \in \mathsf{Sig}_S$; since $\Delta \le \theta \le \beta$, θ is uniform over Δ; let w be the size of its classes
Step 2.2 **set** $N := N \cdot w$
Step 2.3 **set** $S := \mathrm{pr}_{[g-1]}S$ and $\overline{\mathcal{G}} := \overline{\mathcal{G}}_g$
endfor
Step 3 **output** N

Algorithm θ`-Signature`
INPUT: an instance $\mathcal{G}$ of $\#\mathrm{CSP}(\mathcal{H})$, and a congruence $\theta \in \mathsf{Con}(\mathcal{H})$
OUTPUT: the θ-signature of $\Phi(\mathcal{G},\mathcal{H},\pi)$
Step 1 **find** a compact representation of $\Phi(\mathcal{G},\mathcal{H},\pi)$
Step 2 **set** $S := \varnothing$ (the θ-signature of $\Phi(\mathcal{G},\mathcal{H},\pi)$)
Step 3 **for each** $(i,a,b) \in \{1,\dots,m\} \times H^2$ **do**
Step 3.1 **if** there is $\mathbf{a} \in R'$ such that $\mathbf{a}[i] = a$ **then do**
Step 3.1.1 **find** a compact representation R'' of $\Phi(\mathcal{G}',\mathcal{H},\pi)$ where $\mathcal{G}' = \mathcal{G} \cup \{\langle g_1, (\mathbf{a}[1])/_\theta\rangle, \dots, \langle g_{i-1}, (\mathbf{a}[i-1])/_\theta\rangle\}$)
Step 3.1.2 **if** $b \in \langle \mathrm{pr}_i R''\rangle$ **then** $S := S \cup \{(i,a,b)\}$
endif
endfor
Step 5 **return** S

Now, as the numbers of the form t_i^j are known, we have

$$\Phi(\mathcal{G},\mathcal{H},\pi) = \sum_{\mathbf{s}} \Phi(\mathcal{G},\mathcal{H},\varrho_{\mathbf{s}}) = \sum_{\mathbf{s}} t^1_{\mathbf{s}[1]} \cdot \ldots \cdot t^k_{\mathbf{s}[k]}$$
$$= t^1_1 \left(\sum_{\mathbf{s}[2],\dots,\mathbf{s}[k]} t^2_{\mathbf{s}[2]} \cdot \ldots \cdot t^k_{\mathbf{s}[k]} \right) + \ldots + t^1_{m_1} \left(\sum_{\mathbf{s}[2],\dots,\mathbf{s}[k]} t^2_{\mathbf{s}[2]} \cdot \ldots \cdot t^k_{\mathbf{s}[k]} \right)$$
$$= \ldots = \prod_{j=1}^{k} \sum_{i=1}^{m_j} t_i^j,$$

that can be easily computed.

Finally, by Corollary 1, for any mapping $\varrho \in \Phi(\mathcal{G},\mathcal{H})/_{(\alpha^-)^m}$ and any mapping $\varrho' \in \Phi(\mathcal{G},\mathcal{H})/_{((\alpha-1)^+)^m}$ with $\varrho'/_{\alpha^-} = \varrho$, we have $|\Phi(\mathcal{G},\mathcal{H},\varrho)| = |\Phi(\mathcal{G},\mathcal{H},\varrho')| \cdot |\Phi(\mathcal{G},\mathcal{H},\varrho)/_{((\alpha-1)^+)^m}|$. The number $|\Phi(\mathcal{G},\mathcal{H},\varrho)/_{(\alpha-1)^+}|$ can be found using algorithm `Uniform` from the next subsection.

Uniform counting CSPs. Let $\alpha \in K$, π be a mapping of level α^-, and ϱ be a mapping of level $(\alpha-1)^+$. We need a method to find the number $\frac{|\Phi(\mathcal{G},\mathcal{H},\pi)|}{|\Phi(\mathcal{G},\mathcal{H},\varrho)|}$.

We consider first the case when $(\alpha-1)^+$ is the equality relation. In this case the required number can be found by algorithm `Uniform` using a compact representation R'' of $\Phi(\mathcal{G},\mathcal{H},\pi)$. Note also that such a representation can be found by the algorithm from [7] applied to the instance $\mathcal{G}'$ with the same universe as $\mathcal{G}$ and additional unary constraints $\pi(g)$ imposed on each $g \in \mathcal{G}$. We shall assume that for each (n-ary) relational symbol R from the vocabulary of $\mathcal{H}$, and any set $\{i_1,\dots,i_k\} \in [n]$, the vocabulary of $\mathcal{H}$ also contains a k-ary relational symbol $\mathrm{pr}_{\{i_1,\dots,i_k\}}R$ interpreted as $\mathrm{pr}_{\{i_1,\dots,i_k\}}R^{\mathcal{H}}$. For an instance $\mathcal{G}$ of $\#\mathrm{CSP}(\mathcal{H})$ and $g \in \mathcal{G}$ we denote by $\mathcal{G}_g$ the relational structure with universe $G - \{g\}$ and such that, for any relational symbol R and any tuple $(g_1,\dots,g_n) \in R^{\mathcal{G}}$ with $g_{i_1} = \ldots = g_{i_\ell} = g$ and the rest of its entries are different from g, we exclude this tuple from $R^{\mathcal{G}_g}$, and include the tuple $\mathrm{pr}_{[n]-\{i_1,\dots,i_\ell\}}(g_1,\dots,g_n)$ into $\mathrm{pr}_{[n]-\{i_1,\dots,i_\ell\}}R^{\mathcal{G}_g}$. Recall that we assume $G = [m]$.

The correctness of algorithm `Uniform` follows from the rectangularity of the relation $\langle S\rangle$ constructed in the algorithm, and the observation that the congruence θ

constructed on Step 2.1 can be defined as follows: $\langle a, b\rangle \in \theta$ if and only if there is $\mathbf{a} \in \mathrm{pr}_{[g-1]}\langle S\rangle$ such that $(\mathbf{a}, a) \in \langle S\rangle$ and $(\mathbf{a}, b) \in \langle S\rangle$, that is w is the number of possible extensions of a tuple from $\mathrm{pr}_{[g-1]}\langle S\rangle$.

Observe that if we know the signature of the relation $\Phi(\mathcal{G}, \mathcal{H}, \pi)/_{\theta^m}$ we still can use algorithm `Uniform`, for we can consider $\Phi(\mathcal{G}, \mathcal{H}, \pi)/_{\theta^m}$ as a relation on $\mathcal{H}/_{\theta}$. Therefore the problem we are facing now is to find the signature of this relation. Unfortunately, it is not clear at all how to obtain this signature using the signature or a compact representation of $\Phi(\mathcal{G}, \mathcal{H}, \pi)$, nor we can use the algorithm from [7] to compute the signature of $\Phi(\mathcal{G}, \mathcal{H}/_{\theta}, \pi)$, since in general $\Phi(\mathcal{G}, \mathcal{H}/_{\theta}, \pi) \neq \Phi(\mathcal{G}, \mathcal{H}, \pi)/_{\theta^m}$. Instead, to compute each member of the required signature we find a compact representation of a certain modified problem using the algorithm from [7].

More specifically, we first find the *θ-signature* of the relation $\Phi(\mathcal{G}, \mathcal{H}, \pi)$. Let n be a positive integer, let H be a finite set, let θ be an equivalence relation on H, let $\mathbf{a}$, $\mathbf{b}$ be n-ary tuples and let (i, a, b) be any element in $[n] \times H^2$. We say that $(\mathbf{a}, \mathbf{b})$ *θ-witnesses* (i, a, b) if $\langle \mathbf{a}[j], \mathbf{b}[j]\rangle \in \theta$ for each $j < i$, $\mathbf{a}[i] = a$, and $\mathbf{a}'[i] = b$. Let R be any n-ary relation on H. The θ-signature of R, $\theta\mathsf{Sig}_R \subseteq [n] \times H^2$, is defined to be the set containing all those $(i, a, b) \in [n] \times H^2$ θ-witnessed by tuples in R, that is $\theta\mathsf{Sig}_R = \{(i, a, b) \in [n] \times H^2 : \exists \mathbf{a}, \mathbf{b} \in R$ such that $(\mathbf{a}, \mathbf{b})$ θ-witnesses $(i, a, b)\}$.

We shall assume that for each subalgebra B of $\mathcal{H}$ the vocabulary of $\mathcal{H}$ contains a unary relational symbol R_B such that $R_B^{\mathcal{H}} = B$. Let $\mathcal{G}$ be an instance of $\#\mathrm{CSP}(\mathcal{H})$, let $g_1, \ldots, g_k \in \mathcal{G}$, and let $B_1, \ldots, B_k$ be subalgebras of $\mathcal{H}$. By $\mathcal{G} \cup \{\langle g_1, B_1\rangle, \ldots, \langle g_k, B_k\rangle\}$ we denote the relational structure with the same universe as $\mathcal{G}$, and such that the interpretation of every relational symbol $R \notin \{R_{B_1}, \ldots, R_{B_k}\}$ equals to $R^{\mathcal{G}}$ while the interpretation of R_{B_j} equals $R_{B_j}^{\mathcal{G}} \cup \{g_j\}$. Thus, the elements $g_1, \ldots, g_k$ are forced to be mapped to $B_1, \ldots, B_k$ respectively. It is easy to see that the algorithm θ-`Signature` finds the θ-signature of $\Phi(\mathcal{G}, \mathcal{H}, \pi)$. The signature of $\Phi(\mathcal{G}, \mathcal{H}, \pi)/_{\theta^m}$ can then be found by replacing each $(i, a, b) \in S$ by $(i, a/_{\theta}, b/_{\theta})$.

Complexity. Observe that the problem of finding the number $|\Phi(\mathcal{G}, \mathcal{H}, \pi)|$ reduces to finding $m_1 + \ldots + m_k$ numbers of the form $|\Phi(\mathcal{G}, \mathcal{H}, \varrho)|$, where $\varrho\colon \mathcal{G} \to \mathcal{H}/_{(\alpha-1)^+}$, and solving the same number of uniform problems. Clearly, $k \leq |\mathcal{G}| = m$, $m_i \leq |\mathcal{H}| = a$, and $|K| \leq a^2$. If the uniform problem can be solved in time $p(m)$ then the overall time complexity of the algorithm is $(am \cdot p(m))^{a^2}$.

5 Concluding Remarks

The result obtained in this paper is rather general. It includes as particular case the results of [11,20,15,18]. Although we should note that the #P-completeness results from [20] are stronger than those which follow from our result: #P-complete $\#H$-COLORING problems remain #P-complete even when restricted to inputs of bounded degree. The Lovász-goodness condition, proved in [18] to be a criterion of polynomial time solvability of $\#H$-COLORING for directed acyclic graphs, follows from congruence singularity. However, we do not know how to prove that in the case of DAGs Lovász-goodness implies congruence singularity. Our algorithm works, of course, for #-tractable DAGs. Remarkably, the situation that a result on general CSPs cannot be readily translated into

the language of graph theory is not unique. For instance, finding a specialization of the dichotomy theorem for conservative (list) CSPs [3] for digraphs in graph theoretical terms has become a serious research direction.

A major question left unanswered is how to check if a given relational structure is congruence singular. This problem may turn out to be even undecidable.

References

1. Brightwell, G.R., Winkler, P.: Graph homomorphisms and phase transitions. J. of Comb. Th., Ser. B 77, 221–262 (1999)
2. Bulatov, A.A.: A dichotomy theorem for constraints on a three-element set. In: FOCS 2002, pp. 649–658 (2002)
3. Bulatov, A.A.: Tractable conservative constraint satisfaction problems. In: LICS 2003, Ottawa, pp. 321–330 (2003)
4. Bulatov, A.A., Jeavons, P.G.: An algebraic approach to multi-sorted constraits. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 197–202. Springer, Heidelberg (2003)
5. Bulatov, A.A.: Three-element Mal'tsev algebras. Acta Sci. Math. (Szeged) 72, 519–550 (2006)
6. Bulatov, A.A.: The complexity of the counting constraint satisfaction problem. ECCC TR07-093 (2007)
7. Bulatov, A.A., Dalmau, V.: A simple algorithm for Mal'tsev constraints. SIAM J. Comput. 36(1), 16–27 (2006)
8. Bulatov, A.A., Dalmau, V.: Towards a dichotomy theorem for the counting constraint satisfaction problem. Information and Computation 205(5), 651–678 (2007)
9. Bulatov, A.A., Grohe, M.: The complexity of partition functions. Theor. Comput. Sci. 348(2-3), 148–186 (2005)
10. Burris, S., Sankappanavar, H.P.: A course in universal algebra. Graduate Texts in Mathematics, vol. 78. Springer, New York (1981)
11. Creignou, N., Hermann, M.: Complexity of generalized satisfiability counting problems. Information and Computation 125(1), 1–12 (1996)
12. Creignou, N., Khanna, S., Sudan, M.: Complexity Classifications of Boolean Constraint Satisfaction Problems. SIAM Monographs on Discrete Mathematics and Applications, vol. 7. SIAM, Philadelphia (2001)
13. Diaz, J., Serna, M., Thilikos, D.M.: Recent results on parameterized H-coloring. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, DIMACS/DIMATIA Workshop on Graphs, Morphism and Statistical Physics. American Mathematical Society (to appear)
14. Diaz, J., Serna, M., Thilikos, D.M.: Complexity issues on bounded restrictive H-coloring. Discrete Mathematics 307(16), 2082–2093 (2007)
15. Diaz, J., Serna, M., Thilikos, D.M.: Counting list H-colorings and variants. Technical Report LSI-01-27-R, Departament LSI, Universitat Politècnica de Catalunya (2001)
16. Diaz, J., Serna, M., Thilikos, D.M.: Counting h-colorings of partial k-trees. Theor. Comput. Sci. 281, 291–309 (2002)
17. Dyer, M., Frieze, A., Jerrum, M.: On counting independent sets in sparse graphs. SIAM J. Comput. 31, 1527–1541 (2002)
18. Dyer, M., Goldberg, L., Paterson, M.: On counting homomorphisms to directed acyclic graphs. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 38–49. Springer, Heidelberg (2006)

19. Dyer, M., Goldberg, L.A., Greenhill, C., Jerrum, M.: On the relative complexity of approximate counting problems. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 108–119. Springer, Heidelberg (2000)
20. Dyer, M., Greenhill, C.: The complexity of counting graph homomorphisms. Random Structures and Algorithms 17, 260–289 (2000)
21. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. SIAM J. Comput. 28, 57–104 (1998)
22. Freese, R., McKenzie, R.: Commutator theory for congruence modular varieties. London Math. Soc. Lecture Notes, vol. 125. London (1987)
23. Gratzer, G.: General Lattice Theory. Birkhäuser Verlag, Basel (1998)
24. Greenhill, C.: The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. Computational Complexity 9, 52–73 (2000)
25. Hell, P., Nešetřil, J.: Counting list homomorphisms for graphs with bounded degrees. Discr. Math. (to appear)
26. Hobby, D., McKenzie, R.N.: The Structure of Finite Algebras. Contemporary Mathematics, vol. 76. American Mathematical Society, Providence (1988)
27. Jeavons, P.G.: On the algebraic structure of combinatorial problems. Theor. Comput. Sci. 200, 185–204 (1998)
28. Jeavons, P.G., Cohen, D.A., Gyssens, M.: How to determine the expressive power of constraints. Constraints 4, 113–131 (1999)
29. Schaefer, T.J.: The complexity of satisfiability problems. In: STOC 1978, pp. 216–226 (1978)
30. Vadhan, S.P.: The complexity of counting in sparse, regular and planar graphs. SIAM J. Comput. 31(2), 398–427 (2001)
31. Valiant, L.: The complexity of computing the permanent. Theor. Comput. Sci. 8, 189–201 (1979)

On the Hardness of Losing Weight*

Andrei Krokhin[1] and Dániel Marx[2]

[1] Department of Computer Science, Durham University, Durham, DH1 3LE, UK
andrei.krokhin@durham.ac.uk
[2] Department of Computer Science and Information Theory, Budapest University of Technology and Economics, H-1521 Budapest, Hungary
dmarx@cs.bme.hu

Abstract. We study the complexity of local search for the Boolean constraint satisfaction problem (CSP), in the following form: given a CSP instance, that is, a collection of constraints, and a solution to it, the question is whether there is a better (lighter, i.e., having strictly less Hamming weight) solution within a given distance from the initial solution. We classify the complexity, both classical and parameterized, of such problems by a Schaefer-style dichotomy result, that is, with a restricted set of allowed types of constraints. Our results show that there is a considerable amount of such problems that are NP-hard, but fixed-parameter tractable when parameterized by the distance.

1 Introduction

Local search is one of the most widely used approaches to solving hard optimization problems. The basic idea of local search is that one tries to iteratively improve a current solution by searching for better solutions in its (k-)neighborhood (i.e., within distance k from it). Any optimization algorithm can be followed by a local search phase, thus the problem of finding a better solution locally is of practical interest. As a brute force search of a k-neighborhood is not feasible for large k, thus it is natural to study the complexity of searching the k-neighborhood.

The constraint satisfaction problem (CSP) provides a framework in which it is possible to express, in a natural way, many combinatorial problems encountered in artificial intelligence and computer science. A CSP instance is represented by a set of variables, a domain of values for each variable, and a set of constraints on the variables. The basic aim is then to find an assignment of values to the variables that satisfies the constraints. Boolean CSP (when all variables have domain $\{0, 1\}$) is a natural generalization of Sat where constraints are given by arbitrary relations, not necessarily by clauses. Local search methods for Sat and CSP are very extensively studied (see, e.g., [5,9,10,11]).

Complexity classifications for various versions of (Boolean) CSP have recently attracted massive attention from researchers, and one of the most popular directions here is to characterise restrictions on the type of constraints that lead to

* The first author is supported by UK EPSRC grants EP/C543831/1 and EP/C54384X/1; the second author is supported by the Magyary Zoltán Felsőoktatási Közalapítvány and the Hungarian National Research Fund (OTKA grant 67651).

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 662–673, 2008.

problems with lower complexity in comparison with the general case (see [2,3]). Such classifications are sometimes called Schaefer-style because the first classification of this type was obtained by T.J. Schaefer in his seminal work [15]. A local-search related Schaefer-style classification for Boolean MAX CSP was obtained in [1], in the context of local search complexity classes such as PLS.

The hardness of searching the k-neighborhood (for any optimisation problem) can be studied very naturally in the framework of parameterized complexity [6,8], as suggested in [7]; such a study for the traveling salesman problem (TSP) was recently performed in [14]. Parameterized complexity studies hardness in finer detail than classical complexity. Consider, for example, two standard NP-complete problems MINIMUM VERTEX COVER and MAXIMUM CLIQUE. Both have the natural parameter k: the size of the required vertex cover/clique. Both problems can be solved in $n^{O(k)}$ time by complete enumeration. Notice that the degree of the polynomial grows with k, so the algorithm becomes useless for large graphs, even if k is as small as 10. However, MINIMUM VERTEX COVER can be solved in time $O(2^k \cdot n^2)$ [6,8]. In other words, for every fixed cover size there is a polynomial-time (in this case, quadratic) algorithm solving the problem where the degree of the polynomial is independent of the parameter. Problems with this property are called fixed-parameter tractable. The notion of W[1]-hardness in parameterized complexity is analogous to NP-completeness in classical complexity. Problems that are shown to be W[1]-hard, such as MAXIMUM CLIQUE [6,8], are very unlikely to be fixed-parameter tractable. A Schaefer-style classification of the basic Boolean CSP with respect to parameterized complexity (where the parameter is the required Hamming weight of the solution) was obtained in [13].

In this paper, we give a Schaefer-style complexity classification for the following problem: given a collection of Boolean constraints, and a solution to it, the question is whether there is a better (i.e., with smaller Hamming weight) solution within a given (Hamming) distance k from the initial solution. We obtain classification results both for classical (Theorem 9) and for parameterized complexity (Theorem 3). However, we would like to point out that it makes much more sense to study this problem in the parameterized setting. Intuitively, if we are able to decide in polynomial time whether there is a better solution within distance k, then this seems to be almost as powerful as finding the best solution (although there are technicalities such as whether there is a feasible solution at all). Our classification confirms this intuition: searching the k-neighborhood is polynomial-time solvable only in cases where finding the optimum is also polynomial-time solvable. On the other hand, there are cases (for example, 1-IN-3 SAT or affine constraints of fixed arity) where the problem of finding the optimum is NP-hard, but searching the k-neighborhood is fixed-parameter tractable. This suggests evidence that parameterized complexity is the right setting for studying local search.

The paper is organized as follows. Section 2 reviews basic notions of parameterized complexity and Boolean CSP. Section 3 presents the classificiation with respect to fixed-parameter tractability, while Section 4 deals with polynomial-time solvability. The proofs omitted from Section 4 will appear in the full version.

2 Preliminaries

Boolean CSP. A *formula* ϕ is a pair (V, C) consisting of a set V of *variables* and a set C of *constraints.* Each constraint $c_i \in C$ is a pair $\langle \overline{s}_i, R_i \rangle$, where $\overline{s}_i = (x_{i,1}, \ldots, x_{i,r_i})$ is an r_i-tuple of variables (the *constraint scope*) and $R_i \subseteq \{0,1\}^{r_i}$ is an r_i-ary Boolean relation (the *constraint relation*). A function $f : V \to \{0,1\}$ is a *satisfying assignment* of ϕ if $(f(x_{i,1}), \ldots, f(x_{i,r_i}))$ is in R_i for every $c_i \in C$. Let Γ be a set of Boolean relations. A formula is a *Γ-formula* if every constraint relation R_i is in Γ. In this paper, Γ is always a finite set. The *(Hamming) weight* $w(f)$ of assignment f is the number of variables x with $f(x) = 1$. The *distance* $\mathrm{dist}(f_1, f_2)$ of assignments f_1, f_2 is the number of variables x with $f_1(x) \neq f_2(x)$.

We recall various standard definitions concerning Boolean constraints (cf. [3]):

- R is *0-valid* if $(0, \ldots, 0) \in R$.
- R is *1-valid* if $(1, \ldots, 1) \in R$.
- R is *Horn* or *weakly negative* if it can be expressed as a conjunction of clauses such that each clause contains at most one positive literal. It is known that R is Horn if and only if it is *min-closed:* if $(a_1, \ldots, a_r) \in R$ and $(b_1, \ldots, b_r) \in R$, then $(\min(a_1, b_1), \ldots, \min(a_r, b_r)) \in R$.
- R is *affine* if it can be expressed as a conjunction of constraints of the form $x_1 + x_2 + \cdots + x_t = b$, where $b \in \{0,1\}$ and addition is modulo 2. The number of tuples in an affine relation is always an integer power of 2.
- R is *width-2 affine* if it can be expressed as a conjunction of constraints of the form $x = y$ and $x \neq y$.
- R is *IHS-B−* (or *implicative hitting set bounded*) if it can be represented by a conjunction of clauses of the form (x), $(x \to y)$ and $(\neg x_1 \vee \ldots \neg x_n)$, $n \geq 1$.
- The relation $R_{p\text{-IN-}q}$ (for $1 \leq p \leq q$) has arity q and $R_{p\text{-IN-}q}(x_1, \ldots, x_q)$ is true if and only if exactly p of the variables $x_1, \ldots, x_q$ have value 1.

The following definition is new in this paper. It plays a crucial role in characterizing the fixed-parameter tractable cases for local search.

Definition 1. *Let R be a Boolean relation and $(a_1, \ldots, a_r) \in R$. A set $S \subseteq \{1, \ldots, r\}$ is a* flip set *of $(a_1, \ldots, a_r)$ (with respect to R) if $(b_1, \ldots, b_r) \in R$ where $b_i = 1 - a_i$ for $i \in S$ and $b_i = a_i$ for $i \notin S$. We say that R is* flip separable *if whenever some $(a_1, \ldots, a_r) \in R$ has two flip sets S_1, S_2 with $S_1 \subset S_2$, then $S_2 \setminus S_1$ is also a flip set for $(a_1, \ldots, a_r)$.*

It is easy to see that $R_{1\text{-IN-}3}$ is flip separable: every flip set has size exactly 2, hence $S_1 \subset S_2$ is not possible. Moreover, $R_{p\text{-IN-}q}$ is also flip separable for every $p \leq q$. Affine constraints are also flip separable: to see this, it is sufficient to verify the definition only for the constraint $x_1 + \cdots + x_r = 0$.

The basic problem in CSP is to decide if a formula has a satisfying assignment:

CSP(Γ)
Input: A Γ-formula ϕ.
Question: Does ϕ have a satisfying assignment?

Schaefer completely characterized the complexity of CSP(Γ) for every finite set Γ of Boolean relations [15]. In particular, every such problem is either in PTIME or NP-complete, and there is a very clear description of the boundary between the two cases.

Optimization versions of Boolean CSP were investigated in [3,4]. A straightforward way to obtain an optimization problem is to relax the requirement that every constraint is satisfied, and ask for an assignment maximizing the number of satisfied constraints. Another possibility is to ask for a solution with minimum/maximum weight. In this paper, we investigate the problem of minimizing the weight. As we do not consider the approximability of the problem, we define here only the decision version:

MIN-ONES(Γ)

Input: A Γ-formula ϕ and an integer W.

Question: Does ϕ have a satisfying assignment f with $w(f) \leq W$?

The characterization of the approximability of finding a minimum weight satisfying assignment for a Γ-formula can be found in [3]. Here we state only the classification of polynomial-time solvable and NP-hard cases:

Theorem 2 ([3]). *Let Γ be a finite set of Boolean relations.* MIN-ONES(Γ) *is solvable in polynomial time if one the following holds, and* NP*-complete otherwise:*

- *Every $R \in \Gamma$ is 0-valid.*
- *Every $R \in \Gamma$ is Horn.*
- *Every $R \in \Gamma$ is width-2 affine.*

A Schaefer-style characterization of the approximability of finding two satisfying assignments to a formula with a largest distance between them was obtained in [4], motivated by the blocks world problem from KR, while a Schaefer-style classification of the problem of deciding whether a given satisfying assignment to a given CSP instance is component-wise minimal was presented in [12], motivated by the circumscription formalism from AI.

The main focus of the paper is the local search version of minimizing weight:

LS-CSP(Γ)

Input: A Γ-formula ϕ, a satisfying assignment f, and an integer k.

Question: Does ϕ have a satisfying assignment f' with $w(f') < w(f)$ and $\text{dist}(f, f') \leq k$?

LS in the above problem stands for both "local search" and "lighter solution."

Observe that the satisfying assignments of an $(x \vee y)$-formula correspond to the vertex covers of the graph where the variables are the vertices and the edges are the constraints. Thus LS-CSP($\{x \vee y\}$) is the problem of reducing the size of a (given) vertex cover by including and excluding a total of at most k vertices.

As we shall see (Lemma 7), this problem is W[1]-hard, even for bipartite graphs. Since the complement of an independent set is a vertex cover and vice versa, a similar W[1]-hardness result follows for increasing an independent set.

Parameterized complexity. In a *parmeterized problem,* each instance contains an integer k called the *parameter*. A parameterized problem is *fixed-parameter tractable (FPT)* if it can be solved by an algorithm with running time $f(k) \cdot n^c$, where n is the length of the input, f is an arbitrary (computable) function depending only on k, and c is a constant independent of k.

A large fraction of NP-complete problems is known to be FPT. On the other hand, analogously to NP-completeness in classical complexity, the theory of W[1]-hardness can be used to give strong evidence that certain problems are unlikely to be fixed-parameter tractable. We omit the somewhat technical definition of the complexity class W[1], see [6,8] for details. Here it will be sufficient to know that there are many problems, including MAXIMUM CLIQUE, that were proved to be W[1]-hard. To prove that a parameterized problem is W[1]-hard, we have to present a parameterized reduction from a known W[1]-hard problem. A *parameterized reduction* from problem L_1 to problem L_2 is a function that transforms a problem instance x of L_1 with parameter k into a problem instance x' of L_2 with parameter k' in such a way that

- x' is a yes-instance of L_2 if and only if x is a yes-instance of L_1,
- k' can be bounded by a function of k, and
- the transformation can be computed in time $f(k) \cdot |x|^c$ for some constant c and function $f(k)$.

It is easy to see that if there is a parameterized reduction from L_1 to L_2, and L_2 is FPT, then it follows that L_1 is FPT as well.

3 Characterizing Fixed-Parameter Tractability

In this section, we completely characterize those finite sets Γ of Boolean relations for which LS-CSP(Γ) is fixed-parameter tractable.

Theorem 3. *Let Γ be a finite set of Boolean relations. The problem* LS-CSP(Γ) *is in* FPT *if every relation in Γ is Horn or every relation in Γ is flip separable. In all other cases,* LS-CSP(Γ) *is* W[1]*-hard.*

First we handle the fixed-parameter tractable cases (Lemmas 4 and 6)

Lemma 4. *If Γ is finite and every $R \in \Gamma$ is Horn, then* LS-CSP(Γ) *is* FPT.

Proof. If there is a solution f' for the LS-CSP(Γ) instance (ϕ, f, k), then we can assume $f'(x) \leq f(x)$ for every variable x: by defining $f''(x) := \min\{f(x), f'(x)\}$, we get that f'' is also satisfying (as every $R \in \Gamma$ is min-closed) and $\mathrm{dist}(f'', f) \leq \mathrm{dist}(f', f)$. Thus we can restrict our search to solutions that can be obtained from f by changing some 1's to 0's, but every 0 remains unchanged.

Since $w(f') < w(f)$, there is a variable x with $f(x) = 1$ and $f'(x) = 0$. For every variable x with $f(x) = 1$, we try to find a solution f' with $f'(x) = 0$ using a simple bounded-height search tree algorithm. For a particular x, we proceed as follows. We start with initial assignment f. Change the value of x to 0. If there is a constraint $\langle (x_1, \ldots, x_r), R \rangle$ that is not satisfied by the new assignment, then we select one of the variables $x_1, \ldots, x_r$ that has value 1, and change it to 0. Thus at this point we branch into at most $r - 1$ directions. If the assignment is still not satisfying, the we branch again on the variables of some unsatisfied constraint. The branching factor of the resulting search tree is at most $r_{\max} - 1$, where $r_{\max}$ is the maximum arity of the relations in Γ. By the observation above, if there is a solution, then we find a solution on the first k levels of the search tree. Therefore, we can stop the search on the k-th level, implying that we visit at most $(r_{\max} - 1)^{k+1}$ nodes of the search tree. The work to be done at each node is polynomial in the size n of the input, hence the total running time is $(r_{\max} - 1)^{k+1} \cdot n^{O(1)}$. □

If every $R \in \Gamma$ is not only Horn, but IHS-B− (which is a subset of Horn), then the algorithm of Lemma 4 actually runs in polynomial time:

Corollary 5. *If every $R \in \Gamma$ is IHS-B−, then* LS-CSP(Γ) *is in* PTIME.

Proof. We can assume that every constraint is either (x), $(x \to y)$, or $(\bar{x}_1 \vee \cdots \vee \bar{x}_r)$. If a constraint $(\bar{x}_1 \vee \cdots \vee \bar{x}_r)$ is satisfied in the initial assignment f, then it remains satisfied after changing some 1's to 0. Observe that if a constraint (x) or $(x \to y)$ is not satisfied, then at most one variable has the value 1. Thus there is no branching involved in the algorithm of Lemma 4, making it a polynomial-time algorithm. □

For flip separable relations, we give a very similar branching algorithm. However, in this case the correctness of the algorithm requires a nontrivial argument.

Lemma 6. *If Γ is finite and every $R \in \Gamma$ is flip separable, then* LS-CSP(Γ) *is FPT.*

Proof. Let (ϕ, f, k) be an instance of LS-CSP(Γ). If $w(f') < w(f)$ for some assignment f', there is a variable x with $f(x) = 1$ and $f'(x) = 0$. For every variable x with $f(x) = 1$, we try to find a solution f' with $f'(x) = 0$ using a simple bounded-height search tree algorithm. For each such x, we proceed as follows. We start with the initial assignment f and set the value of x to 0. Iteratively do the following: (a) if there is a constraint in ϕ that is not satisfied by the current assignment and such that the value of some variable in it has not been flipped yet (on this branch), then we select one of such variables, and flip its value; (b) if there is no such constraint, but the current assignment is not satisfying then we move to the next branch; (c) if every constraint is satisfied, then either we found a required solution or else we move to the next branch. If a required solution is not found on the first k levels of the search tree then the algorithm reports that there is no required solution.

Assume that (ϕ, f, k) is a yes-instance. We claim that if f' is a required solution with minimal distance from f, then some branch of the algorithm finds it.

Let X be the set of variables on which f and f' differ, so $|X| \leq k$. We now show that on the first k levels of the search tree, the algorithm finds some satisfying assignment f_0 (possibly heavier than f) that differs from f only on a subset $X_0 \subseteq X$ of variables. To see this, assume that at some node of the search tree, the current assignment differs from the initial assignment only on a subset of X; we show that this remains true for at least one child of the node. If we branch on the variables $(x_1, \ldots, x_r)$ of an unsatisfied constraint, then at least one of its variables, say x_i, has a value different from f' (as f' is a satisfying assignment). It follows that $x_i \in X$: otherwise the current value of x_i is $f(x_i)$ (since so far we changed variables only in X) and $f(x_i) = f'(x_i)$ (by the definition of X), contradicting the fact that current value of x_i is different from $f(x_i)$. Thus if we change variable x_i, it remains true that only variables from X are changed. Since $|X| \leq k$, this branch of the algorithm has to find some satisfying assignment f_0.

If $w(f_0) < w(f)$, then, by the choice of f', we must have $f_0 = f'$. Otherwise, let $X_0 \subseteq X$ be the set of variables where f and f_0 differ and let f'' be the assignment that differs from f exactly on the variables $X \setminus X_0$. From the fact that every constraint is flip separable, it follows that f'' is a satisfying assignment. We claim that $w(f'') < w(f)$. Indeed, if changing the values of the variables in X decreases the weight and changing the values in X_0 does not decrease the weight, then the set $X \setminus X_0$ has to decrease the weight. This contradicts the assumption that f' is a solution whose distance from f is minimal: f'' is a solution with distance $|X \setminus X_0| < |X|$. Thus it is sufficient to investigate only the first k levels of the search tree. As in the proof of Lemma 4, the branching factor of the tree is at most $r_{\max} - 1$, and the algorithm runs in time $(r_{\max} - 1)^{k+1} \cdot n^{O(1)}$. □

All the hardness proofs in this section are based on the following lemma:

Lemma 7. LS-CSP($\{x \vee y\}$) *is* W[1]*-hard.*

Proof. The proof is by reduction from a variant of Maximum Clique: given a graph $G(V, E)$ with a distinguished vertex x and an integer t, we have to decide whether G has a clique of size t that contains x. It is easy to see that this problem is W[1]-hard. Furthermore, it can be assumed that t is odd. Let n be the number of vertices of G and let m be the number of edges. We construct a formula ϕ on $m + n(t-1)/2 - 1$ variables and a satisfying assignment f such that G has a clique of size t containing x if and only if ϕ has a satisfying assignment f' with $w(f') < w(f)$ and distance at most $k := t(t-1) - 1$ from f.

Let $d := (t-1)/2$ (note that t is odd). The formula ϕ has d variables v_1, $\ldots$, v_d for each vertex $v \neq x$ of G and a variable u_e for each edge e of G. The distinguished vertex x has only $d-1$ variables $x_1, \ldots, x_{d-1}$. If a vertex v is the endpoint of an edge e, then for every $1 \leq i \leq d$ (or $1 \leq i \leq d-1$, if $v = x$), we add the constraint $u_e \vee v_i$. Thus each variable u_e is in $2d-1$ or $2d$ constraints (depending on whether x is the endpoint of e or not). Set $f(u_e) = 1$ for every $e \in E$ and $f(v_i) = 0$ for every $v \in V$, $1 \leq i \leq d$. Clearly, f is a satisfying assignment.

Assume that G has a clique K of size t that includes x. Set $f'(v_i) = 1$ for every $v \in K$ $(1 \leq i \leq d)$ and set $f'(u_e) = 0$ for every edge e in K; let f' be the same

as f on every other variable. Observe that f' is also a satisfying assignment: if a variable u_e was changed to 0 and there is a constraint $u_e \vee v_i$, then $v \in K$ and hence $f'(v_i) = 1$. We have $w(f') < w(f)$: $dt - 1$ variables were changed to 1 (note that $x \in K$) and $t(t-1)/2 = dk$ variables were changed to 0. Moreover, the distance of f and f' is exactly $dt - 1 + t(t-1)/2 = t(t-1) - 1 = k$.

Assume now that f' satisfies the requirements. Let K be the set of those vertices v in G for which $f'(v_i) = 1$ for every i. We claim that K is a clique of size t in G. Observe that there are at least $d|K|-1$ variables v_i with $f'(v_i) > f(v_i)$ and $f'(u_e) < f(u_e)$ is possible only if both endpoints of e are in K, i.e., e is in the set $E(K)$ of edges in K. Thus $w(f') < w(f)$ implies $d|K| - 1 < |E(K)| \leq |K|(|K| - 1)/2$, which is only possible if $|K| \geq 2d + 1 = t$. If $|K| > t$, then $f'(v_i) > f(v_i)$ for at least $(t + 1)d - 1$ variables, hence there must be at least that many variables u_e with $f'(u_e) < f(u_e)$. Thus the distance of f and f' is at least $2(t+1)d-2 > t(t-1)-1$. Therefore, we can assume $|K| = t$. Now $dt-1 < |E(K)| \leq |K|(|K| - 1)/2 = t(t - 1)/2$ is only possible if $|E(K)| = t(t - 1)/2$ (i.e., K is a clique) and it follows that there are exactly $dt - 1$ variables v_i with $f'(v_i) > f(v_i)$ (i.e., $x \in K$). □

Now we are ready to present the main hardness proof of the section:

Lemma 8. *If Γ contains a relation R_1 that is not Horn and a relation R_2 that is not flip separable, then* LS-CSP(Γ) *is* W[1]*-hard.*

Proof. The proof is by reduction from LS-CSP($\{x \vee y\}$). Let (ϕ_1, f_1, k) be an instance of LS-CSP($\{x \vee y\}$), i.e., every constraint relation in formula $\phi_1 = (V, C)$ is $(x \vee y)$. Since R_1 is not min-closed, we can assume (by permuting the variables) that for some $r_1, r_2 \geq 1$, $r_3, r_4 \geq 0$, if we define

$$R'_1(x, y, w_0, w_1) = R_1(\overbrace{x, \ldots, x}^{r_1}, \overbrace{y, \ldots, y}^{r_2}, \overbrace{w_0, \ldots, w_0}^{r_3}, \overbrace{w_1, \ldots, w_1}^{r_4}),$$

then $(0, 1, 0, 1), (1, 0, 0, 1) \in R'_1$, but $(0, 0, 0, 1) \notin R'_1$. Since R'_1 is obtained from R_1 by identifying variables, we can use the relation R'_1 when specifying instances of LS-CSP(Γ). We consider two cases:

Case 1: $(1, 1, 0, 1) \in R'_1$. In this case $R'_1(x, y, 0, 1) = x \vee y$, hence it is easy to simulate LS-CSP($\{x \vee y\}$). The only difficulty is how to simulate the constants 0 and 1. We do this as follows. Let us construct a formula ϕ_2 that has every variable of V and new variables q_0^j, q_1^j for every $1 \leq j \leq k+1$ (these new variables will play the role of the constants). We define assignment f_2 of ϕ_2 by setting $f_2(x) = f_1(x)$ for $x \in V$ and $f_2(q_0^j) = 0$ and $f_2(q_1^j) = 1$ for $1 \leq j \leq k+1$. For $1 \leq a, b, c \leq k+1$, we add constraint $c^1_{a,b,c} = R'_1(q_1^a, q_0^b, q_0^b, q_1^c)$; it is clearly satisfied by assignment f_2. To simulate a constraint $x \vee y$, we add $c^2_{x,y,j} = R'_1(x, y, q_0^j, q_1^1)$ for every $1 \leq j \leq k + 1$.

It is easy to see that if there is a solution f'_1 for the original instance (ϕ_1, f_1, k), then by setting $f'_2(x) = f'_1(x)$ for every $x \in V$ and $f'_2(q_0^j) = 0$, $f'_2(q_1^j) = 1$ for every $1 \leq j \leq k+1$ gives a solution f'_2 for the constructed instance (ϕ_2, f_2, k). We claim the converse is also true: if f'_2 is a solution for the instance (ϕ_2, f_2, k), then

the restriction f_1' of f_2' to V gives a solution for (ϕ_1, f_1, k). Since the distance of f_2 and f_2' is at most k, there are $1 \leq b, c \leq k+1$ with $f_2'(q_0^b) = 0$ and $f_2'(q_1^c) = 1$. Because of the constraint $c_{a,b,c}^1$, we have that $f_2'(q_1^a) = 1$ for every $1 \leq a \leq k+1$. It follows that f_2' restricted to V is a satisfying assignment of ϕ_1: for every constraint $x \vee y \in C$, the constraint $c_{x,y,b}^2$ prevents the possibility $f_2'(x) = f_2'(y) = 0$. We have seen that $f_2'(q_0^j) \geq f_2(q_0^j)$ and $f_2'(q_1^j) \geq f_2(q_1^j)$ for every $1 \leq j \leq k+1$. Now $w(f_2') < w(f_2)$ implies that the weight of f_2' on V has to be less than the weight of f_2 on V. Thus $w(f_1') < w(f_1)$.

Case 2: $(1,1,0,1) \notin R_1'$, which means that $R_1'(x,y,0,1)$ is $x \neq y$. In this case we have to rely on the fact that R_2 is not flip separable to simulate the constraint $x \vee y$. We construct formula ϕ_2 and its satisfying assignment f_2 as follows. Each variable x is replaced by 3 variables x_1, x_2, x_3. We set $f_2(x_1) = f_2(x_2) = f_1(x)$ and $f_2(x_3) = 1 - f_1(x)$. Furthermore, for $1 \leq j \leq 3k+1$, we add the variables q_0^j and q_1^j and set $f_2(q_0^j) = 0$ and $f_2(q_1^j) = 1$.

For every $1 \leq a, b, c \leq 3k+1$, we add the constraint $c_{a,b,c}^1 = R_1'(q_1^a, q_0^b, q_0^b, q_1^c)$, as in the previous case. For every $x \in V$, $1 \leq j \leq 3k+1$, and $\ell = 1, 2$, we add $c_{x,\ell,j}^2 = R_1'(x_\ell, x_3, q_0^j, q_1^1)$, as we shall see, the role of these constraints is to ensure $f_2'(x_1) = f_2'(x_2) \neq f_2'(x_3)$.

Since R_2 is not flip separable, there is a tuple $(s_1, \ldots, s_r) \in R_2$ that has flip sets $S_1 \subset S_2$, but $S_2 \setminus S_1$ is not a flip set. For every constraint $x \vee y$ of ϕ_1, we add $3k+1$ constraints to ϕ_2 as follows. First, for $1 \leq i \leq r$ and $1 \leq j \leq 3k+1$, we define variable v_i^j as

$$v_i^j = \begin{cases} x_1 & \text{if } i \in S_1 \text{ and } s_i = 0, \\ x_3 & \text{if } i \in S_1 \text{ and } s_i = 1, \\ y_1 & \text{if } i \in S_2 \setminus S_1 \text{ and } s_i = 1, \\ y_3 & \text{if } i \in S_2 \setminus S_1 \text{ and } s_i = 0, \\ q_1^1 & \text{if } i \notin S_2 \text{ and } s_i = 1, \\ q_0^j & \text{if } i \notin S_2 \text{ and } s_i = 0. \end{cases}$$

For every $1 \leq j \leq 3k+1$, we add the constraint $c_{x,y,j}^3 = R_2(v_1^j, \ldots, v_r^j)$. For example, assume that $(0,1,0,1) \in R_2$ and this tuple has flip sets $S_1 = \{1,2\}$ and $S_2 = \{1,2,3,4\}$, but $S_2 \setminus S_1 = \{3,4\}$ is not a flip set. This means that $(0,1,0,1), (1,0,1,0), (1,0,0,1) \in R_2$ and $(0,1,1,0) \notin R_2$. In this case, constraint $c_{x,y,j}^3$ is $R_2(x_1, x_3, y_3, y_1)$. Assuming $f(x_1) \neq f(x_3)$ and $f(y_1) \neq f(y_3)$, any combination of values on x_1 and y_1 satisfies the constraint, except if $f(x_1) = f(y_1) = 0$. Thus the constraint effectively acts as a constraint $x_1 \vee y_1$.

Finally, we set the maximum allowed distance to $k' := 3k$. This completes the description of the constructed instance (ϕ_2, f_2, k').

Assume first that f_1' is a solution for the instance (ϕ_1, f_1, k). Define $f_2'(x_1) = f_2'(x_2) = f_1'(x)$ and $f_2'(x_3) = 1 - f_1'(x)$ for every $x \in V$, and define $f_2'(q_0^j) = 0$, $f_2'(q_1^j) = 1$ for every $1 \leq j \leq 3k+1$. The fact $w(f_1') < w(f_1)$ implies $w(f_2') < w(f_2)$. Furthermore, the distance of f_2 and f_2' is exactly three times the distance of f_1 and f_1', i.e., at most $3k$. We claim that f_2' satisfies the constraints of ϕ_2. This is easy to see for $c_{a,b,c}^1$ and $c_{x,\ell,j}^2$. For $c_{x,y,j}^3$, this can be seen as follows:

– If $f_2'(x) = 0$, $f_2'(y) = 1$, then this holds because $(s_1, \ldots, s_r) \in R_2$.
– If $f_2'(x) = 1$, $f_2'(y) = 0$, then this holds because S_2 is a flip set.
– If $f_2'(x) = 1$, $f_2'(y) = 1$, then this holds because S_1 is a flip set.

For the other direction, assume that f_2' is a solution for the instance (ϕ_2, f_2, k'). Define $f_1'(x) = f_2'(x_1)$ for every $x \in V$; we claim that f_1' is a solution for the instance (ϕ_1, f_1, k). Since the distance of f_2 and f_2' is at most $3k$, there are $1 \leq b, c \leq 3k+1$ with $f_2'(q_0^b) = 0$ and $f_2'(q_1^c) = 1$. Because of the constraint $c^1_{a,b,c}$, we have that $f_2'(q_1^a) = 1$ for every $1 \leq a \leq 3k+1$. The constraints $c^2_{x,1,b}$ and $c^2_{x,2,b}$ ensure that $f_2'(x_1) = f_2'(x_2) = 1 - f_2'(x_3)$ (since $(0,0,0,1) \notin R_1'$ and $(1,1,0,1) \notin R_1'$). It follows that the distance of f_1 and f_1' is at most k: $f_1(x) \neq f_1'(x)$ implies $f_2(x_\ell) \neq f_2'(x_\ell)$ for $\ell = 1, 2, 3$, hence this can hold for at most k different $x \in V$. Moreover, $w(f_1') < w(f_1)$: this follows from the facts $w(f_2') < w(f_2)$ and $f_2'(q_0^j) \geq f_2(q_0^k)$, $f_2'(q_1^j) \geq f_2(q_1^k)$ $(1 \leq j \leq 3k+1)$.

We claim that every constraint $x \vee y$ of ϕ_1 is satisfied. Assume that $f_1'(x) = f_1'(y) = f_2'(x_1) = f_2'(y_1) = 0$. Now $c^3_{x,y,b}$ is not satisfied: this follows from the fact that $S_2 \setminus S_1$ is not a flip set for $(s_1, \ldots, s_r)$ (with respect to R_2). □

4 Characterizing Polynomial-Time Solvability

In this section, we completely characterize those finite sets Γ of Boolean relations for which LS-CSP(Γ) is polynomial-time solvable.

Theorem 9. *Let Γ be a finite set of Boolean relations. The problem* LS-CSP(Γ) *is in* PTIME *if every relation in Γ is IHS-B− or every relation in Γ is width-2 affine. In all other cases,* LS-CSP(Γ) *is* NP*-hard.*

Proof. If every relation in Γ is IHS-B−, then Corollary 5 gives a polynomial-time algorithm. If every relation in Γ is width-2 affine then the following simple algorithm solves LS-CSP(Γ): for a given instance (ϕ, f, k), compute the graph whose vertices are the variables in ϕ and two vertices are connected if there is a constraint $=$ or $\neq$ in ϕ imposed on them. If there is a connected component of this graph which has at most k vertices and such that f assigns more 1's in this component than it does 0's, then flipping the values in this component gives a required lighter solution. If such a component does not exists, then there is no lighter solution within distance k from f.

By Lemma 8, if Γ contains a relation that is not Horn and a relation that is not flip separable then LS-CSP(Γ) is NP-hard. (Note that Lemma 8 gives a polynomial-time reduction from an NP-hard problem.) Thus we can assume that every relation in Γ is Horn or every relation in Γ is flip separable. We prove only the former case; the proof for the latter case will appear in the full version.

Assume now that Γ is Horn, and there is a relation $R \in \Gamma$ that is not IHS-B−. We prove that LS-CSP($\{R\}$) is NP-hard. It is shown in the proof of Lemma 5.27 of [3] that then R is at least ternary and one can permute the coordinates in R and then substitute 0 and 1 in R in such a way that the ternary relation $R'(x, y, z) = R(x, y, z, 0, \ldots, 0, 1, \ldots, 1)$ has the following properties:

1. R' contains tuples $(1,1,1),(0,1,0),(1,0,0),(0,0,0)$, and
2. R' does not contain the tuple $(1,1,0)$.

Note that if $(0,0,1) \in R'$ then $R'(x,x,y)$ is $x \to y$. If $(0,0,1) \notin R'$ then, since R (and hence R') is Horn (i.e., min-closed), at least one of of the tuples $(1,0,1)$ and $(0,1,1)$ is not in R'. Then it is easy to check that at least one of the relations $R'(x,y,x)$ and $R'(y,x,x)$ is $x \to y$. Hence, we can use constraints of the form $x \to y$ when specifying instances of LS-CSP$(\{R'\})$.

We reduce Minimum Dominating Set to LS-CSP$(\{R'\})$. Let $G(V,E)$ be a graph with n vertices and m edges where a dominating set of size at most t has to be found. Let $v_1, \ldots, v_n$ be the vertices of G. Let $S = 3m$. We construct a formula with $nS + 2m + 1$ variables as follows:

- There is a special variable x.
- For every $1 \leq i \leq n$, there are S variables $x_{i,1}, \ldots, x_{i,S}$. There is a constraint $x_{i,j} \to x_{i,j'}$ for every $1 \leq j, j' \leq n$.
- For every $1 \leq i \leq n$, if $v_{s_1}, \ldots, v_{s_d}$ are the neighbors of v_i, then there are d variables $y_{i,1}, \ldots, y_{i,d}$ and the following constraints: $x_{s_1,1} \to y_{i,1}$, $R'(x_{s_2,1}, y_{i,1}, y_{i,2}), R'(x_{s_3,1}, y_{i,2}, y_{i,3}), \ldots, R'(x_{s_d,1}, y_{i,d-1}, y_{i,d}), R'(x_{i,1}, y_{i,d}, x)$.
- For every variable z, there is a constraint $x \to z$.

Observe that the number of variables of type $y_{i,j}$ is exactly $2m$. Setting every variable to 1 is a satisfying assignment. Set $k := St + S - 1$.

Assume that there is a satisfying assignment where the number of 0's is at most k (but positive). Variable x has to be 0, otherwise every other variable is 1. If $x_{i,1}$ is 0, then $x_{i,j}$ is 0 for every $1 \leq j \leq S$. Thus $k < S(t+1)$ implies that there are at most t values of i such that $x_{i,1}$ is 0. Let D consist of all vertices v_i such that $x_{i,1}$ is 0. We claim that D is a dominating set. Suppose that some vertex v_i is not dominated. This means that if $v_{s_1}, \ldots, v_{s_d}$ are the neighbors of v_i, then the variables $x_{s_1,1}, \ldots, x_{s_d,1}, x_{i,1}$ all have the value 1. However, this means that these variables force variables $y_{i,1}, \ldots, y_{i,d}$ and variable x to value 1, a contradiction. Thus D is a dominating set of size at most t.

The reverse direction is also easy to see. Assume that G has a dominating set D of size at most t. For every $1 \leq i \leq n$ and $1 \leq j \leq S$, set variable $x_{i,j}$ to 1 if and only v_i is not contained in D. Set x to 0. It is easy to see that this assignment can be extended to the variables $y_{i,j}$ to obtain a satisfying assignment: indeed, if $v_{s_1}, \ldots, v_{s_d}$ are the neighbors of v_i and none of them is in D then $v_i \in D$, and we set $y_{i,1} = \ldots = y_{i,d} = 1$. Otherwise, if j is minimal such that $v_{s_j} \in D$, we set $y_{i,1} = \ldots = y_{i,j-1} = 1$ and $y_{i,q} = 0$ for $q \geq j$. This satisfying assignment contains at most $St + 2m + 1 \leq k$ variables with value 0, as required.

Finally, we reduce LS-CSP$(\{R'\})$ to LS-CSP$(\{R\})$ (and so to LS-CSP(Γ)). Take an instance (ϕ, f, k) of LS-CSP$(\{R'\})$, let V be the variables of ϕ and $c_1, \ldots, c_p$ the constraints of ϕ. We build an instance ϕ' of LS-CSP$(\{R\})$ as follows.

1. For each $1 \leq i \leq \max(p, k+1)$, introduce new variables x_0^i, x_1^i.
2. For each constraint $c_i = R'(x,y,z)$ in formula ϕ, replace it by the constraint $R(x,y,z,x_0^i,\ldots,x_0^i,x_1^i,\ldots,x_1^i)$.

3. For each ordered pair (i, j) where $1 \leq i, j \leq \max(p, k+1)$, add the constraints $R(x_0^i, x_0^i, x_0^j, x_0^j, \ldots, x_0^j, x_1^j, \ldots, x_1^j)$ and $R(x_1^j, x_1^j, x_1^i, x_0^j, \ldots, x_0^j, x_1^j, \ldots, x_1^j)$.

Finally, extend f so that, for all i, we have $x_0^i = 0$ and $x_1^i = 1$. It is clear that the obtained mapping f' is a solution to the new instance. Note that, by the choice of R', the tuple $(1, 1, 0, 0, \ldots, 0, 1, \ldots, 1)$ does not belong to R. Hence, the constraints added in step (3) above ensure that if a variable of the form x_0^i or x_1^i in f' is flipped then, in order to get a solution to ϕ' different from f', one must flip at least one of x_0^i and x_1^i for each $1 \leq i \leq \max(p, k+1)$. Consequently, all solutions to ϕ' that lie within distance k from f' must agree with f' on all such variables. In other words, searching for such a solution, it makes sense to flip only variables from V. Thus, clearly, the instances (ϕ, f, k) and (ϕ', f', k) are equivalent. □

References

1. Chapdelaine, P., Creignou, N.: The complexity of Boolean constraint satisfaction local search problems. Annals of Mathematics and Artificial Intelligence 43, 51–63 (2005)
2. Cohen, D., Jeavons, P.: The complexity of constraint languages. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, ch. 8. Elsevier, Amsterdam (2006)
3. Creignou, N., Khanna, S., Sudan, M.: Complexity Classifications of Boolean Constraint Satisfaction Problems. SIAM Monographs on Discrete Mathematics and Applications, vol. 7 (2001)
4. Crescenzi, P., Rossi, G.: On the Hamming distance of constraint satisfaction problems. Theoretical Computer Science 288(1), 85–100 (2002)
5. Dantsin, E., Goerdt, A., Hirsch, E., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., Schöning, U.: A deterministic $(2 - \frac{2}{k+1})^n$ algorithm for k-SAT based on local search. Theoretical Computer Science 289, 69–83 (2002)
6. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
7. Fellows, M.R.: Parameterized complexity: new developments and research frontiers. In: Aspects of Complexity (Kaikura, 2000). de Gruyter Series in Logic and Applications, vol. 4, pp. 51–72 (2001)
8. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
9. Gu, J., Purdom, P., Franko, J., Wah, B.W.: Algorithms for the Satisfiability Problem. Cambridge University Press, Cambridge (2000)
10. Hirsch, E.: SAT local search algorithms: worst-case study. Journal of Automated Reasoning 24, 127–143 (2000)
11. Hoos, H., Tsang, E.: Local search methods. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming. ch. 5. Elsevier, Amsterdam (2006)
12. Kirousis, L., Kolaitis, P.: The complexity of minimal satisfiability problems. Information and Computation 187, 20–39 (2003)
13. Marx, D.: Parameterized complexity of constraint satisfaction problems. Computational Complexity 14, 153–183 (2005)
14. Marx, D.: Searching the k-change neighborhood for TSP is W[1]-hard. Operations Research Letters 36, 31–36 (2008)
15. Schaefer, T.J.: The complexity of satisfiability problems. In: STOC 1978, pp. 216–226 (1978)

Product Theorems Via Semidefinite Programming

Troy Lee and Rajat Mittal

Department of Computer Science, Rutgers University

Abstract. The tendency of semidefinite programs to compose perfectly under product has been exploited many times in complexity theory: for example, by Lovász to determine the Shannon capacity of the pentagon; to show a direct sum theorem for non-deterministic communication complexity and direct product theorems for discrepancy; and in interactive proof systems to show parallel repetition theorems for restricted classes of games. Despite all these examples of product theorems—some going back nearly thirty years—it was only recently that Mittal and Szegedy began to develop a general theory to explain when and why semidefinite programs behave perfectly under product. This theory captured many examples in the literature, but there were also some notable exceptions which it could not explain—namely, an early parallel repetition result of Feige and Lovász, and a direct product theorem for the discrepancy method of communication complexity by Lee, Shraibman, and Špalek. We extend the theory of Mittal and Szegedy to explain these cases as well. Indeed, to the best of our knowledge, our theory captures all examples of semidefinite product theorems in the literature.

1 Introduction

A prevalent theme in complexity theory is what we might roughly call product theorems. These results look at how the resources to accomplish several independent tasks scale with the resources needed to accomplish the tasks individually. Let us look at a few examples of such questions:

Shannon Capacity. If a graph G has an independent set of size α, how large an independent set can the product graph $G \times G$ have? How does α compare with amortized independent set size $\lim_{k\to\infty} \alpha(G^k)^{1/k}$? This last quantity, known as the Shannon capacity, gives the effective alphabet size of a graph where vertices are labeled by letters and edges represent letters which can be confused if adjacent.

Hardness Amplification. Product theorems naturally arise in the context of hardness amplification. If it is hard to evaluate a function $f(x)$, then an obvious approach to create a harder function is to evaluate two independent copies $f'(x, y) = (f(x), f(y))$ of f. There are different ways that f' can be harder than f—a direct sum theorem aims to show that evaluation of f' requires twice as many resources as needed to evaluate f; direct product theorems aim to show that the error probability to compute f' is larger than that of f, given the same amount of resources.

Soundness Amplification. Very related to hardness amplification is what we might call soundness amplification. This arises in the context of interactive proofs where one wants to reduce the error probability of a protocol, by running several checks in parallel.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 674–685, 2008.

The celebrated parallel repetition theorem shows that the soundness of multiple prover interactive proof systems can be boosted in this manner [Raz98].

These examples illustrate that many important problems in complexity theory deal with product theorems. One successful approach to these types of questions has been through semidefinite programming. In this approach, if one wants to know how some quantity $\sigma(G)$ behaves under product, one first looks at a semidefinite approximation $\bar{\sigma}(G)$ of $\sigma(G)$. One then hopes to show that $\bar{\sigma}(G)$ provides a good approximation to $\sigma(G)$, and that $\bar{\sigma}(G \times G) = \bar{\sigma}(G)\bar{\sigma}(G)$. In this way one obtains that the original quantity must approximately product as well.

Let us see how this approach has been used on some of the above questions.

Shannon Capacity. Perhaps the first application of this technique was to the Shannon capacity of a graph. Lovász developed a semidefinite quantity, the Lovász theta function $\vartheta(G)$, showed that it was a bound on the independence number of a graph, and that $\vartheta(G \times G) = \vartheta(G)^2$. In this way he determined the Shannon capacity of the pentagon, resolving a long standing open problem [Lov79].

Hardness Amplification. Karchmer, Kushilevitz, and Nisan [KKN95] notice that another program introduced by Lovász [Lov75], the fractional cover number, can be used to characterize non-deterministic communication complexity, up to small factors. As this program also perfectly products, they obtain a direct sum theorem for non-deterministic communication complexity.

As another example, Linial and Shraibman [LS06] show that a semidefinite programming quantity γ_2^∞ characterizes the discrepancy method of communication complexity, up to constant factors. Lee, Shraibman and Špalek [LSŠ08] then use this result, together with the fact that γ_2^∞ perfectly products, to show a direct product theorem for discrepancy, resolving an open problem of Shaltiel [Sha03].

Soundness Amplification. Although the parallel repetition theorem was eventually proven by other means [Raz98, Hol07], one of the first positive results did use semidefinite programming. Feige and Lovász [FL92] show that the acceptance probability $\omega(G)$ of a two-prover interactive proof on input x can be represented as an integer program. They then study a semidefinite relaxation of this program, and use this to show that if $\omega(G) < 1$ then $\sup_{k\to\infty} \omega(G^k)^{1/k} < 1$, for a certain class of games G. More recently, Cleve et al. [CSUU07] look at two-prover games where the provers share entanglement, and show that the value of a special kind of such a game known as an XOR game can be exactly represented by a semidefinite program. As this program perfectly products, they obtain a perfect parallel repetition theorem for this game.

We hope this selection of examples shows the usefulness of the semidefinite programming approach to product theorems. Until recently, however, this approach remained an ad hoc collection of examples without a theory to explain when and why semidefinite programs perfectly product. Mittal and Szegedy [MS07] began to address this lacuna by giving a general sufficient condition for a semidefinite program to obey a product rule. This condition captures many examples in the literature, notably the Lovász theta function [Lov79], and the parallel repetition for XOR games with entangled provers [CSUU07].

Other examples cited above, however, do not fit into the Mittal and Szegedy framework: namely, the product theorem of Feige and Lovász [FL92] and that for

discrepancy [LSŠ08]. We extend the condition of Mittal and Szegedy to capture these cases as well. Indeed, in our (admittedly imperfect) search of the literature, we have not found a semidefinite product theorem which does not fit into our framework.

2 Preliminaries

We begin with some notational conventions and basic definitions which will be useful. In general, lower case letters like v will denote column vectors, and upper case letters like A will denote matrices. Vectors and matrices will be over the real numbers. The notation v^T or A^T will denote the transpose of a vector or matrix. We will say $A \succeq 0$ if A is positive semidefinite, i.e. if A is symmetric and $v^T A v \geq 0$ for all vectors v.

We will use several kinds of matrix products. We write AB for the normal matrix product. For two matrices A, B of the same dimensions, $A \circ B$ denotes the matrix formed by their entrywise product. That is, $(A \circ B)[x, y] = A[x, y]B[x, y]$. We will use $A \bullet B$ for the entrywise sum of $A \circ B$. Equivalently, $A \bullet B = \mathrm{Tr}(AB^T)$. We will use the notation $v \geq w$ to indicate that the vector v is entrywise greater than or equal to the vector w.

In applications we often face the situation where we would like to use the framework of semidefinite programming, which requires symmetric matrices, but the problem at hand is represented by matrices which are not symmetric, or possibly not even square. Fortunately, this can often be handled by a simple trick. This trick is so useful that we will give it its own notation. For an arbitrary real matrix A, we define

$$\widehat{A} = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$$

We will refer to this as the *bipartite version* of A, as such a matrix corresponds to the adjacency matrix of a (weighted) bipartite graph. In many respects $\widehat{A}$ behaves similarly to A, but has the advantages of being symmetric and square.

More generally, we will refer to a matrix M which can be written as

$$M = \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}$$

as *block anti-diagonal* and a matrix M which can be written

$$M = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$$

as *block diagonal*.

One subtlety that arises in working with the bipartite version $\widehat{A}$ instead of A itself is in defining the product of instances. Mathematically, it is most convenient to work with the normal tensor product

$$\widehat{A} \otimes \widehat{A} = \begin{bmatrix} 0 & 0 & 0 & A \otimes A \\ 0 & 0 & A \otimes A^T & 0 \\ 0 & A^T \otimes A & 0 & 0 \\ A^T \otimes A^T & 0 & 0 & 0 \end{bmatrix}$$

Whereas what naturally arises in the product of problems is instead the "bipartite tensor" product of A:

$$\widehat{A \otimes A} = \begin{bmatrix} 0 & A \otimes A \\ A^T \otimes A^T & 0 \end{bmatrix}$$

Kempe, Regev, and Toner [KRT07] observe, however, that a product theorem for the tensor product implies a product theorem for the bipartite tensor product. This essentially follows because $\widehat{A \otimes A}$ is a submatrix of $\widehat{A} \otimes \widehat{A}$, and so positive semidefiniteness of the latter implies positive semidefiniteness of the former. See [KRT07] for full details.

3 Product Rule with Non-negativity Constraints

In this section we prove our main theorem extending the product theorem of Mittal and Szegedy [MS07] to handle non-negativity constraints. As our work builds on the framework developed by Mittal and Szegedy, let us first explain their results.

Mittal and Szegedy consider a general affine semidefinite program $\pi = (J, \mathbf{A}, b)$. Here $\mathbf{A} = (A_1, \ldots, A_m)$ is a vector of matrices, and we extend the notation $\bullet$ such that $\mathbf{A} \bullet X = (A_1 \bullet X, A_2 \bullet X, \ldots, A_m \bullet X)$. The value of π is given as

$$\begin{aligned} \alpha(\pi) = \max_X\ & J \bullet X \text{ such that} \\ & \mathbf{A} \bullet X = b \\ & X \succeq 0. \end{aligned}$$

We take this as the primal formulation of π. Part of what makes semidefinite programming so useful for proving product theorems is that we can also consider the dual formulation of π. Dualizing in the straightforward way gives:

$$\begin{aligned} \alpha^*(\pi) = \min_y\ & y^T b \\ & y^T \mathbf{A} - J \succeq 0 \end{aligned}$$

A necessary pre-condition for the semidefinite programming approach to proving product theorems is that so-called strong duality holds. That is, that $\alpha(\pi) = \alpha^*(\pi)$, the optimal primal and dual values agree. We will assume this throughout our discussion. For more information about strong duality and sufficient conditions for it to hold, see [BV06].

We define the product of programs as follows: for $\pi_1 = (J_1, \mathbf{A}_1, b_1)$ and $\pi_2 = (J_2, \mathbf{A}_2, b_2)$ we define $\pi_1 \times \pi_2 = (J_1 \otimes J_2, \mathbf{A}_1 \otimes \mathbf{A}_2, b_1 \otimes b_2)$. If $\mathbf{A}_1$ is a tuple of m_1 matrices and $\mathbf{A}_2$ is a tuple of m_2 matrices, then the tensor product $\mathbf{A}_1 \otimes \mathbf{A}_2$ is a tuple of $m_1 m_2$ matrices consisting of all the tensor products $\mathbf{A}_1[i] \otimes \mathbf{A}_2[j]$.

It is straightforward to see that $\alpha(\pi_1 \times \pi_2) \geq \alpha(\pi_1)\alpha(\pi_2)$. Namely, if X_1 realizes $\alpha(\pi_1)$ and X_2 realizes $\alpha(\pi_2)$, then $X_1 \otimes X_2$ will be a feasible solution to $\pi_1 \times \pi_2$ with value $\alpha(\pi_1)\alpha(\pi_2)$. This is because $X_1 \otimes X_2$ is positive semidefinite, $(\mathbf{A}_1 \otimes \mathbf{A}_2) \bullet (X_1 \otimes X_2) = (\mathbf{A}_1 \bullet X_1) \otimes (\mathbf{A}_2 \bullet X_2) = b_1 \otimes b_2$, and $(J_1 \otimes J_2) \bullet (X_1 \otimes X_2) = (J_1 \bullet X_1) \otimes (J_2 \bullet X_2) = \alpha(\pi_1)\alpha(\pi_2)$.

Mittal and Szegedy show the following theorem giving sufficient conditions for the reverse inequality $\alpha(\pi_1 \times \pi_2) \leq \alpha(\pi_1)\alpha(\pi_2)$.

Theorem 1 (Mittal and Szegedy [MS07]). *Let $\pi_1 = (J_1, \mathbf{A}_1, b_1), \pi_2 = (J_2, \mathbf{A}_2, b_2)$ be two affine semidefinite programs for which strong duality holds. Then $\alpha(\pi_1 \times \pi_2) \leq \alpha(\pi_1)\alpha(\pi_2)$ if either of the following two conditions hold:*

1. *$J_1, J_2 \succeq 0$.*
2. *(Bipartiteness) There is a partition of rows and columns into two sets such that with respect to this partition, J_i is block anti-diagonal, and all matrices in $\mathbf{A}_i$ are block diagonal, for $i \in \{1, 2\}$.*

We extend item (2) of this theorem to also handle non-negativity constraints. This is a class of constraints which seems to arise often in practice, and allows us to capture cases in the literature that the original work of Mittal and Szegedy does not. More precisely, we consider programs of the following form:

$$\begin{aligned} \alpha(\pi) = \max_X\ & J \bullet X \text{ such that} \\ & \mathbf{A} \bullet X = b \\ & \mathbf{B} \bullet X \geq \mathbf{0} \\ & X \succeq 0 \end{aligned}$$

Here both $\mathbf{A}$ and $\mathbf{B}$ are vectors of matrices, and $\mathbf{0}$ denotes the all 0 vector.

We should point out a subtlety here. A program of this form can be equivalently written as an affine program by suitably extending X and modifying $\mathbf{A}$ accordingly to enforce the $\mathbf{B} \bullet X \geq \mathbf{0}$ constraints through the $X \succeq 0$ condition. The catch is that two equivalent programs do not necessarily lead to equivalent product instances. We explicitly separate out the non-negativity constraints here so that we can define the product as follows: for two programs, $\pi_1 = (J_1, \mathbf{A}_1, b_1, \mathbf{B}_1)$ and $\pi_2 = (J_2, \mathbf{A}_2, b_2, \mathbf{B}_2)$ we say

$$\pi_1 \times \pi_2 = (J_1 \otimes J_2, \mathbf{A}_1 \otimes \mathbf{A}_2, b_1 \otimes b_2, \mathbf{B}_1 \otimes \mathbf{B}_2).$$

Notice that the equality constraints and non-negativity constraints do not interact in the product, which is usually the intended meaning of the product of instances.

It is again straightforward to see that $\alpha(\pi_1 \times \pi_2) \geq \alpha(\pi_1)\alpha(\pi_2)$, thus we focus on the reverse inequality. We extend Condition (2) of Theorem 1 to the case of programs with non-negativity constraints. As we will see in Section 4, this theorem captures the product theorems of Feige-Lovász [FL92] and discrepancy [LSŠ08].

Theorem 2. *Let $\pi_1 = (J_1, \mathbf{A}_1, b_1, \mathbf{B}_1)$ and $\pi_2 = (J_2, \mathbf{A}_2, b_2, \mathbf{B}_2)$ be two semidefinite programs for which strong duality holds. Suppose the following two conditions hold:*

1. *(Bipartiteness) There is a partition of rows and columns into two sets such that, with respect to this partition, J_i and all the matrices of $\mathbf{B}_i$ are block anti-diagonal, and all the matrices of $\mathbf{A}_i$ are block diagonal, for $i \in \{1, 2\}$.*
2. *There are non-negative vectors u_1, u_2 such that $J_1 = u_1^T \mathbf{B}_1$ and $J_2 = u_2^T \mathbf{B}_2$.*

Then $\alpha(\pi_1 \times \pi_2) \leq \alpha(\pi_1)\alpha(\pi_2)$.

Proof. To prove the theorem it will be useful to consider the dual formulations of π_1 and π_2. Dualizing in the standard fashion, we find

$$\begin{aligned}\alpha(\pi_1) = \min_{y_1}\ & y_1^T b_1 \text{ such that} \\ & y_1^T \mathbf{A}_1 - (z_1^T \mathbf{B}_1 + J_1) \succeq 0 \\ & z_1 \geq 0\end{aligned}$$

and similarly for π_2. Fix y_1, z_1 to be vectors which realizes this optimum for π_1 and similarly y_2, z_2 for π_2. The key observation of the proof is that if we can also show that

$$y_1^T \mathbf{A}_1 + (z_1^T \mathbf{B}_1 + J_1) \succeq 0 \text{ and } y_2^T \mathbf{A}_2 + (z_2^T \mathbf{B}_2 + J_2) \succeq 0 \tag{1}$$

then we will be done. Let us for the moment assume (1) and see why this is the case.

If (1) holds, then we also have

$$\begin{aligned}\left(y_1^T \mathbf{A}_1 - (z_1^T \mathbf{B}_1 + J_1)\right) \otimes \left(y_2^T \mathbf{A}_2 + (z_2^T \mathbf{B}_2 + J_2)\right) \succeq 0 \\ \left(y_1^T \mathbf{A}_1 + (z_1^T \mathbf{B}_1 + J_1)\right) \otimes \left(y_2^T \mathbf{A}_2 - (z_2^T \mathbf{B}_2 + J_2)\right) \succeq 0\end{aligned}$$

Averaging these equations, we find

$$(y_1 \otimes y_2)^T (\mathbf{A}_1 \otimes \mathbf{A}_2) - \left((z_1^T \mathbf{B}_1 + J_1) \otimes (z_2^T \mathbf{B}_2 + J_2)\right) \succeq 0.$$

Let us work on the second term. We have

$$\begin{aligned}(z_1^T \mathbf{B}_1 + J_1) \otimes (z_2^T \mathbf{B}_2 + J_2) &= (z_1 \otimes z_2)^T (\mathbf{B}_1 \otimes \mathbf{B}_2) + z_1^T \mathbf{B}_1 \otimes J_2 + J_1 \otimes z_2^T \mathbf{B}_2 \\ &\quad + J_1 \otimes J_2 \\ &= (z_1 \otimes z_2)^T (\mathbf{B}_1 \otimes \mathbf{B}_2) + (z_1 \otimes u_2)^T \mathbf{B}_1 \otimes \mathbf{B}_2 \\ &\quad + (u_1 \otimes z_2)^T \mathbf{B}_1 \otimes \mathbf{B}_2 + J_1 \otimes J_2.\end{aligned}$$

Thus if we let $v = z_1 \otimes z_2 + z_1 \otimes u_2 + u_1 \otimes z_2$ we see that $v \geq 0$ as all of z_1, z_2, u_1, u_2 are, and also

$$(y_1 \otimes y_2)^T \otimes (\mathbf{A}_1 \otimes \mathbf{A}_2) - (v^T (\mathbf{B}_1 \otimes \mathbf{B}_2) + J_1 \otimes J_2) \succeq 0.$$

Hence $(y_1 \otimes y_2, v)$ form a feasible solution to the dual formulation of $\pi_1 \times \pi_2$ with value $(y_1 \otimes y_2)(b_1 \otimes b_2) = \alpha(\pi_1)\alpha(\pi_2)$.

It now remains to show that (1) follows from the condition of the theorem. Given $y\mathbf{A} - (z^T \mathbf{B} + J) \succeq 0$ and the bipartiteness condition of the theorem, we will show that $y\mathbf{A} + (z^T \mathbf{B} + J) \succeq 0$. This argument can then be applied to both π_1 and π_2.

We have that $y^T \mathbf{A}$ is block diagonal and $z^T \mathbf{B} + J$ is block anti-diagonal with respect to the same partition. Hence for any vector $x^T = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$, we have

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \left(y^T \mathbf{A} - (z^T \mathbf{B} + J)\right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & -x_2 \end{bmatrix} \left(y^T \mathbf{A} + (z^T \mathbf{B} + J)\right) \begin{bmatrix} x_1 \\ -x_2 \end{bmatrix}$$

Thus the positive semidefiniteness of $y\mathbf{A} + (z^T \mathbf{B} + J)$ follows from that of $y\mathbf{A} - (z^T \mathbf{B} + J)$.

One may find the condition that J lies in the positive span of $\mathbf{B}$ in the statement of Theorem 2 somewhat unnatural. If we remove this condition, however, a simple counterexample shows that the theorem no longer holds. Consider the program

$$\alpha(\pi) = \max_X \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \bullet X$$
$$\text{such that } I \bullet X = 1, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \bullet X \geq 0, \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \bullet X \geq 0, X \succeq 0.$$

Here I stands for the 2-by-2 identity matrix. This program satisfies the bipartiteness condition of Theorem 2, but J does not lie in the positive span of the matrices of $\mathbf{B}$. It is easy to see that the value of this program is zero. The program $\pi \times \pi$, however, has positive value as $J \otimes J$ does not have any negative entries but is the matrix with ones on the main anti-diagonal.

4 Applications

Two notable examples of semidefinite programming based product theorems in the literature are not captured by Theorem 1. Namely, a recent direct product theorem for the discrepancy method of communication complexity, and an early semidefinite programming based parallel repetition result of Feige and Lovász. As we now describe in detail, these product theorems can be explained by Theorem 2.

4.1 Discrepancy

Communication complexity is an ideal model to study direct sum and direct product theorems as it is simple enough that one can often hope to attain tight results, yet powerful enough that such theorems are non-trivial and have applications to reasonably powerful models of computation. See [KN97] for more details on communication complexity and its applications.

Shaltiel [Sha03] began a systematic study of when we can expect direct product theorems to hold, and in particular looked at this question in the model of communication complexity for exactly these reasons. He showed a general counterexample where a direct product theorem does not hold, yet also proved a direct product for communication complexity lower bounds shown by a particular method—the discrepancy method under the uniform distribution. Shaltiel does not explicitly use semidefinite programming techniques, but proceeds by relating discrepancy under the uniform distribution to the spectral norm, which can be cast as a semidefinite program.

This result was recently generalized and strengthened by Lee, Shraibman, and Špalek [LSŠ08] who show an essentially optimal direct product theorem for discrepancy under arbitrary distributions. This result follows the general plan for showing product theorems via semidefinite programming: they use a result of Linial and Shraibman [LS06] that a semidefinite programming quantity $\gamma_2^\infty(M)$ characterizes the discrepancy of the communication matrix M up to a constant factor, and then show that $\gamma_2^\infty(M)$ perfectly products. The semidefinite programming formulation of $\gamma_2^\infty(M)$ is not affine but involves non-negativity constraints, and so does not fall into the original framework of Mittal and Szegedy.

Let us now look at the semidefinite program describing γ_2^∞:

$$\begin{aligned}\gamma_2^\infty(M) = \max_X\ & \widehat{M} \bullet X \text{ such that} \\ & X \bullet I = 1 \\ & X \bullet E_{ij} = 0 \text{ for all } i \neq j \leq m, i \neq j \geq m \\ & X \bullet (\widehat{M} \circ E_{ij}) \geq 0 \text{ for all } i \leq m, j \geq m, \text{ and } i \geq m, j \leq m \\ & X \succeq 0.\end{aligned}$$

Here $E_{i,j}$ is the 0/1 matrix with exactly one entry equal to 1 in coordinate (i, j). In this case, $\mathbf{A}$ is formed from the matrices I and E_{ij} for $i \neq j \leq m$ and $i \neq j \geq m$. These matrices are all block diagonal with respect to the natural partition of $\widehat{M}$. Further, the objective matrix $\widehat{M}$ and matrices of $\mathbf{B}$ are all block anti-diagonal with respect to this partition. Finally, we can express $\widehat{M} = u^T\mathbf{B}$ by simply taking u to be the all 1 vector.

4.2 Feige-Lovász

In a seminal paper, Babai, Fortnow, and Lund [BFL91] show that all of non-deterministic exponential time can be captured by interactive proof systems with two-provers and polynomially many rounds. The attempt to characterize the power of two-prover systems with just one round sparked interest in a parallel repetition theorem—the question of whether the soundness of a two-prover system can be amplified by running several checks in parallel. Feige and Lovász [FL92] ended up showing that two-prover one-round systems capture NEXP by other means, and a proof of a parallel repetition theorem turned out to be the more difficult question [Raz98]. In the same paper, however, Feige and Lovász also take up the study of parallel repetition theorems and show an early positive result in this direction.

In a two-prover one-round game, the Verifier is trying to check if some input x is in the language L. The Verifier chooses questions $s \in S, t \in T$ with some probability $P(s, t)$ and then sends question s to prover Alice, and question t to prover Bob. Alice sends back an answer $u \in U$ and Bob replies $w \in W$, and then the Verifier answers according to some Boolean predicate $V(s, t, u, w)$. We call this a game $G(V, P)$, and write the acceptance probability of the Verifier as $\omega(G)$. In much the same spirit as the result of Lovász on the Shannon capacity of a graph, Feige and Lovász show that if the value of a game $\omega(G) < 1$ then also $\sup_k \omega(G^k)^{1/k} < 1$, for a certain class of games known as unique games.

The proof of this result proceeds in the usual way: Feige and Lovász first show that $\omega(G)$ can be represented as a quadratic program. They then relax this quadratic program in the natural way to obtain a semidefinite program with value $\sigma(G) \geq \omega(G)$. Here the proof faces an extra complication as $\sigma(G)$ does not perfectly product either. Thus another round of relaxation is done, throwing out some constraints to obtain a program with value $\bar{\sigma}(G) \geq \sigma(G)$ which does perfectly product. Part of our motivation for proving Theorem 2 was to uncover the "magic" of this second round of relaxation, and explain why Feige and Lovász remove the constraints they do in order to obtain something which perfectly products.

Although the parallel repetition theorem was eventually proven by different means [Raz98, Hol07], the semidefinite programming approach has recently seen renewed interest for showing tighter parallel repetition theorems for restricted classes of games and where the provers share entanglement [CSUU07, KRT07].

The Relaxed Program. As mentioned above, Feige and Lovász first write $\omega(G)$ as an integer program, and then relax this to a semidefinite program with value $\sigma(G) \geq \omega(G)$. We now describe this program. The objective matrix C is a $|S| \times |U|$-by-$|T| \times |W|$ matrix where the rows are labeled by pairs (s,u) of possible question and answer pairs with Alice and similarly the columns are labeled by (t,w) possible dialogue with Bob. The objective matrix for a game $G = (V,P)$ is given by $C[(s,u),(t,w)] = P(s,t)V(s,t,u,w)$. We also define an auxiliary matrices B_{st} of dimensions the same as $\widehat{C}$, where $B_{st}[(s',u),(t',w)] = 1$ if $s = s'$ and $t = t'$ and is zero otherwise.

With these notations in place, we can define the program:

$$\sigma(G) = \max_X \frac{1}{2}\widehat{C} \bullet X \text{ such that} \tag{2}$$
$$X \bullet B_{st} = 1 \text{ for all } s,t \in S \cup T \tag{3}$$
$$X \geq 0 \tag{4}$$
$$X \succeq 0 \tag{5}$$

We see that we cannot apply Theorem 2 here as we have global non-negativity constraints (not confined to the off-diagonal blocks) and global equality constraints (not confined to the diagonal blocks). Indeed, Feige and Lovász remark that this program does not perfectly product.

Feige and Lovász then consider a further relaxation with value $\bar{\sigma}(G)$ whose program does fit into our framework. They throw out all the constraints of Equation (3) which are off-diagonal, and remove the non-negativity constraints for the on-diagonal blocks of X. More precisely, they consider the following program:

$$\bar{\sigma}(G) = \max_X \frac{1}{2}\widehat{C} \bullet X \text{ such that} \tag{6}$$
$$\sum_{u,w \in U} |X[(s,u),(s',w)]| \leq 1 \text{ for all } s,s' \in S \tag{7}$$
$$\sum_{u,w \in W} |X[(t,u),(t',w)]| \leq 1 \text{ for all } t,t' \in T \tag{8}$$
$$X \bullet E_{(s,u),(t,w)} \geq 0 \text{ for all } s \in S, t \in T, u \in U, w \in W \tag{9}$$
$$X \succeq 0 \tag{10}$$

Let us see that this program fits into the framework of Theorem 2. The vector of matrices $\mathbf{B}$ is composed of the matrices $E_{(s,u),(t,w)}$ for $s \in S, u \in U$ and $t \in T, w \in W$. Each of these matrices is block diagonal with respect to the natural partition of $\widehat{C}$. Moreover, as $\widehat{C}$ is non-negative and bipartite, we can write $\widehat{C} = u^T\mathbf{B}$ for a non-negative u, namely where u is given by concatenation of the entries of C and C^T written as a long vector.

The on-diagonal constraints given by Equations (7), (8) are not immediately seen to be of the form needed for Theorem 2 for two reasons: first, they are inequalities rather than equalities, and second, they have of absolute value signs. Fortunately, both of these problems can be easily dealt with.

It is not hard to check that Theorem 2 also works for inequality constraints $\mathbf{A} \bullet X \leq b$. The only change needed is that in the dual formulation we have the additional constraint $y \geq \mathbf{0}$. This condition is preserved in the product solution constructed in the proof of Theorem 2 as $y \otimes y \geq 0$.

The difficulty in allowing constraints of the form $\mathbf{A} \bullet X \leq b$ is in fact that the opposite direction $\alpha(\pi_1 \times \pi_2) \geq \alpha(\pi_1)\alpha(\pi_2)$ does not hold in general. Essentially, what can go wrong here is that $a_1, a_2 \leq b$ does not imply $a_1 a_2 \leq b^2$. In our case, however, this does not occur as all the terms involved are positive and so one can show $\bar{\sigma}(G_1 \times G_2) \geq \bar{\sigma}(G_1)\bar{\sigma}(G_2)$.

To handle the absolute value signs we consider an equivalent formulation of $\bar{\sigma}(G)$. We replace the condition that the sum of absolute values is at most one by constraints saying that the sum of every possible $\pm$ combination of values is at most one:

$$\begin{aligned}
\bar{\sigma}'(G) = \max_X \frac{1}{2}\widehat{C} \bullet X \text{ such that} \\
\sum_{u,w \in U} (-1)^{x_{uw}} X[(s,u),(s',w)] \leq 1 \text{ for all } s, s' \in S \text{ and } x \in \{0,1\}^{|U|^2} \\
\sum_{u,w \in W} (-1)^{x_{uw}} X[(t,u),(t',w)] \leq 1 \text{ for all } t, t' \in T \text{ and } x \in \{0,1\}^{|W|^2} \\
X \bullet E_{(s,u),(t,w)} \geq 0 \text{ for all } s \in S, t \in T, u \in U, w \in W \\
X \succeq 0
\end{aligned}$$

This program now satisfies the conditions of Theorem 2. It is clear that $\bar{\sigma}(G) = \bar{\sigma}'(G)$, and also that this equivalence is preserved under product. Thus the product theorem for $\bar{\sigma}(G)$ follows from Theorem 2 as well.

5 Conclusion

We have now developed a theory which covers all examples of semidefinite programming product theorems we are aware of in the literature. Having such a theory which can be applied in black-box fashion should simplify the pursuit of product theorems via semidefinite programming methods, and we hope will find future applications. That being said, we still think there is more work to be done to arrive at a complete understanding of semidefinite product theorems. In particular, we do not know the extension of item (1) of Theorem 1 to the case of non-negative constraints, and it would nice to understand to what extent item (2) of Theorem 2 can be relaxed.

So far we have only considered tensor products of programs. One could also try for more general *composition* theorems: in this setting, if one has a lower bound on the complexity of $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^k \to \{0,1\}$, one would like to obtain a lower bound on $(f \circ g)(\boldsymbol{x}) = f(g(x_1), \ldots, g(x_n))$. What we have studied so far in looking at tensor products corresponds to the special cases where f is the PARITY or

AND function, depending on if the objective matrix is a sign matrix or a $0/1$ valued matrix. One example of such a general composition theorem is known for the adversary method, a semidefinite programming quantity which lower bounds quantum query complexity. There it holds that $\mathrm{ADV}(f \circ g) \geq \mathrm{ADV}(f)\mathrm{ADV}(g)$ [Amb03, HLŠ07]. It would be interesting to develop a theory to capture these cases as well.

Acknowledgements

We would like to thank Mario Szegedy for many insightful conversations. We would also like to thank the anonymous referees for their helpful comments. TL is supported by a NSF Mathematical Sciences Postdoctoral Fellowship, and RM is supported by NSF Grant 0523866.

References

[Amb03] Ambainis, A.: Polynomial degree vs. quantum query complexity. In: Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, pp. 230–239. IEEE, Los Alamitos (2003)

[BFL91] Babai, L., Fortnow, L., Lund, C.: Non-deterministic exponential time has two-prover interactive protocols. Computational Complexity 1, 3–40 (1991)

[BV06] Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge University Press, Cambridge (2006)

[CSUU07] Cleve, R., Slofstra, W., Unger, F., Upadhyay, S.: Perfect parallel repetition theorem for quantum XOR proof systems. In: Proceedings of the 22nd IEEE Conference on Computational Complexity. IEEE, Los Alamitos (2007)

[FL92] Feige, U., Lovász, L.: Two-prover one-round proof systems: their power and their problems. In: Proceedings of the 24th ACM Symposium on the Theory of Computing, pp. 733–744. ACM, New York (1992)

[HLŠ07] Høyer, P., Lee, T., Špalek, R.: Negative weights make adversaries stronger. In: Proceedings of the 39th ACM Symposium on the Theory of Computing. ACM, New York (2007)

[Hol07] Holenstein, T.: Parallel repetition theorem: simplifications and the no-signaling case. In: Proceedings of the 39th ACM Symposium on the Theory of Computing, pp. 411–419 (2007)

[KKN95] Karchmer, M., Kushilevitz, E., Nisan, N.: Fractional covers and communication complexity. SIAM Journal on Discrete Mathematics 8(1), 76–92 (1995)

[KN97] Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)

[KRT07] Kempe, J., Regev, O., Toner, B.: The unique game conjecture with entangled provers is false. Technical Report 0712.4279, arXiv (2007)

[Lov75] Lovász, L.: On the ratio of optimal integral and fractional covers. Discrete Mathematics 13, 383–390 (1975)

[Lov79] Lovász, L.: On the Shannon capacity of a graph. IEEE Transactions on Information Theory IT-25, 1–7 (1979)

[LS06] Linial, N., Shraibman, A.: Learning complexity versus communication complexity. In: Proceedings of the 23rd IEEE Conference on Computational Complexity. IEEE, Los Alamitos (2008)

[LSŠ08] Lee, T., Shraibman, A., Špalek, R.: A direct product theorem for discrepancy. In: Proceedings of the 23rd IEEE Conference on Computational Complexity. IEEE, Los Alamitos (2008)

[MS07] Mittal, R., Szegedy, M.: Product rules in semidefinite programming. In: 16th International Symposium on Fundamentals of Computation Theory (2007)

[Raz98] Raz, R.: A parallel repetition theorem. SIAM Journal on Computing 27(3), 763–803 (1998)

[Sha03] Shaltiel, R.: Towards proving strong direct product theorems. Computational Complexity 12(1–2), 1–22 (2003)

Sound 3-Query PCPPs Are Long*

Eli Ben-Sasson[1,**], Prahladh Harsha[2,***], Oded Lachish[3], and Arie Matsliah[1]

[1] Computer Science Department, Technion, Israel Institute of Technology, Haifa, Israel
{eli,ariem}@cs.technion.ac.il
[2] Toyota Technological Institute, Chicago, USA
prahladh@tti-c.org
[3] Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, Coventry, United Kingdom
oded@dcs.warwick.ac.uk

Abstract. We initiate the study of the tradeoff between the *length* of a probabilistically checkable proof of proximity (PCPP) and the maximal *soundness* that can be guaranteed by a 3-query verifier with oracle access to the proof. Our main observation is that a verifier limited to querying a short proof cannot obtain the same soundness as that obtained by a verifier querying a long proof. Moreover, we quantify the *soundness deficiency* as a function of the proof-length and show that any verifier obtaining "best possible" soundness must query an exponentially long proof.

1 Introduction

In this extended abstract (see [BHLM07] for a full version) we discuss the relationship between two basic parameters of *probabilistically checkable proofs of proximity* (PCPPs) — their *proof length* and *soundness*. PCPPs were simultaneously introduced in [BGH+06] and (under the name *assignment testers*) in [DR06] and a similar notion also appeared earlier in [Sze99] and [EKR04]. The interest in PCPPs stems first and foremost from the role they play within the proof of the celebrated PCP Theorem of [AS98, ALM+98]. All recent constructions of PCPs, starting with [BGH+06, DR06], use PCPPs to simplify the proof of the PCP theorem and improve certain aspects of it, most notably, to decrease the length of proofs as in [BGH+06, BS05, Din07]. All previous proofs of the PCP theorem implicitly use PCPPs and can be augmented to yield them. (See, e.g., [BGH+06, Theorem 3.2] for a conversion of the original PCP system of [AS98, ALM+98] into a PCPP). But PCPPs are also interesting beyond the

* Work of first three authors supported in part by a European Community International Reintegration Grant, an Alon Fellowship and a grant from the Israeli Science Foundation.

** Landau Fellow – supported by the Taub and Shalom Foundations.

*** Work done while the author was visiting the Technion, Israel Institute of Technology.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 686–697, 2008.

scope of the PCP Theorem. They can be used to transform any error correcting code into a locally testable one and to construct "relaxed" locally decodable codes [BGH+06]. Additionally, as shown in [FF05, GR05], they have applications to questions in the theory of "tolerant" property testing that was introduced in [PRR06].

A *PCPP verifier*, (or, simply, verifier) for a property $P \subset \{0,1\}^n$ is a randomized, sublinear-time algorithm that distinguishes with high probability between inputs that belong to P and inputs that are far in relative Hamming distance from all members of P. In this respect a verifier is similar to a *property-tester* as defined in [GGR98]. However, in contrast to a tester, the verifier may query an auxiliary proof, called a *proof of proximity*. A PCPP system has four basic parameters of interest, described next — *length, query complexity, completeness* and a *soundness function*. The *proof length* is the length of the auxiliary proof that is queried by the verifier[1]. The *query complexity* is the maximal number of bits that can be read from *both* the input and the proof. The *completeness* parameter is the minimal probability with which inputs that belong to P are accepted when they are presented along with a "good" proof of proximity. Finally, the *soundness function* $s(\delta)$ is the minimal rejection probability of inputs that are δ-far (in relative Hamming distance) from (all members of) P, where the minimum is taken over all such δ-far inputs and all possible proofs that may accompany them.[2] (See Section 2 for a formal definition of PCPPs and further discussion of their parameters).

1.1 Informal Description of Main Results

To describe our results, let us discuss the range of parameters we can expect from a verifier for a *linear* property over the binary alphabet, i.e., a property that is closed under addition modulo 2. (This amounts to saying P is a linear subspace of $\mathbb{F}_2^n$ where $\mathbb{F}_2$ denotes the two-element field.) We look at nonadaptive 3-query verifiers with perfect completeness, thereby fixing two of the four basic parameters, and look at the tradeoff between proof length and soundness. We point out that all known constructions of PCPPs naturally yield nonadaptive 3-query verifiers with perfect completeness , so the results described next apply to all of them.

Suppose we are interested in minimizing proof length. The results of [Din07, BS05] give constructions with proofs of length at most $m \cdot \text{polylog } n$ where m is the minimal size of circuit deciding P. (Notice the linearity of P implies $m = O(n^2)$.) Regarding the soundness function, consider a random word that can be shown to have, with high probability, distance $\delta \approx \frac{1}{2}$ from P. The "short PCPP"

[1] In PCP literature one often encounters *randomness complexity* as a means for bounding proof-length. The two parameters are closely related, i.e., proof-length $\approx 2^{\text{randomness}}$ and we stick to the former parameter.

[2] Often, in literature on PCPs, the term "soundness" refers to "soundness-error" which is defined to be the *maximal* acceptance probability of a "bad" input. The connection between soundness (used here) and soundness-error, denoted s_{error}, is given by $s = 1 - s_{error}$.

construction mentioned above gives $s(\delta) > \varepsilon$ for some small and unspecified constant $\varepsilon > 0$ that depends only on δ and neither on P, nor on n.

Next, let us try to increase the soundness. We show in Theorem 2.1 that soundness can be boosted to $s(\delta) \geq \delta$ and this soundness is obtained by a *linear* verifier. A verifier is called linear if the set of answer-bits that cause it to accept forms a linear space. (For $\mathbb{F}_2$ this amounts to saying the verifier accepts iff the sum (mod 2) of the queried bits is 0.) For such verifiers, it can be shown that $s(\delta)$ is at most $\frac{1}{2}$ and thus the soundness of our construction is optimal. On the down side, the length of the proof used by this verifier is exponential in n. (We note in passing that this soundness-optimal construction can be carried out over any finite field of prime size. See Theorem 2.1 for details.)

To sum up the situation so far, we have constructions that are nearly optimal in length, but are deficient in soundness and we have constructions that are optimal in soundness but deficient in length. One could have conjectured (as we did before embarking on this research project) that a "super-PCPP" with short proofs *and* optimal soundness exists. Our first main result, stated in Theorem 2.2 and Corollary 2.1, rules this out. We show a tradeoff between proof length and soundness that essentially matches our soundness-optimal construction. In plain words, for some properties (discussed below) any PCPP verifier that queries a short proof of length ℓ must incur a *soundness deficiency*, and this deficiency increases as ℓ decreases (see Definition 2.5 for a formal definition of deficiency).

Our next main result, stated in Theorem 2.3 and Corollary 2.2, proves a tighter tradeoff similar to the one mentioned above for the case of $\mathbb{F}_p$-linear verifiers for $\mathbb{F}_p$-linear properties over a finite field of size p. Our results in this case are stronger even though the query complexity, when measured in bits, is greater than 3 (however, the bits are read from three "blocks", where each block encodes a field element).

1.2 Proof Techniques

In terms of techniques, we focus on the special class of *inspective* verifiers that read at most 2 proof-bits per invocation. For such verifiers we prove *exponential* length-soundness tradeoffs that are later on used to imply our main results for the case of general (i.e., not necessarily inspective) verifiers. To prove the exponential tradeoff for inspective verifiers we show a connection between PCPP proof length and property-testing query complexity, that may be of independent interest. The connection is that any linear property that can be verified with proofs of length ℓ by linear inspective verifiers must be testable with query complexity $\approx \log \ell$.

To show this connection between PCPP proof length and property-testing query complexity, we construct a natural constraint graph that represents the verifier, and then we apply a the *Decomposition Lemma* due to [LR99] to break this graph into components with small radius[3], while removing only a small fraction of its edges. Our analysis is completed by showing that inspective PCPPs

[3] The radius of a connected graph is the minimum maximal distance between any vertex and any other vertex (i.e, $rad(G) = \min_v \max_u d(u, v)$, where $d(u, v)$ denotes the distance between the vertices u and v.

whose induced graph has radius R can be converted with no loss in soundness into (proofless) testers with query complexity $O(R)$.

The decomposition lemma mentioned above was previously used in a closely related context in [Tre05] to provide algorithms for approximating unique games. We use it for similar purposes, namely, for analyzing constraint graphs, but our setting differs from that of [Tre05] in several important aspects (see [BHLM07] for further details).

In the next Section we give formal definitions and statements of our main results. Due to lack of space, all proofs are omitted from this extended abstract. Instead, in Section 3 we prove a weaker version of one of our main theorems. All omitted proofs, additional results and further detailed discussion can be found in [BHLM07].

2 Definitions and Main Results

2.1 Probabilistically Checkable Proofs of Proximity (PCPPs)

Recall the basic task of *property testing*. Let Σ be a finite alphabet. A set $P \subseteq \Sigma^n$ is called a *property* of length n over Σ. We are interested in deciding the promise problem whose set of YES instances is P and whose set of NO instances is $\mathrm{NO}_{\delta_0} = \{w \in \Sigma^n \mid \delta(w, P) > \delta_0\}$, where $\delta(\cdot)$ denotes fractional Hamming distance and δ_0 is called the *proximity parameter*. The decision should be made after making a small number of queries into the input *word* $w \in \Sigma^n$ and the decision should be correct with high probability. (More information on property testing can be found in [GGR98] and in the survey [Fis01].)

In the context of *proximity testing* we try to decide the very same promise problem but the difference is that we allow oracle access to an additional *proof of proximity* $\pi \in \Sigma^\ell$ of length ℓ, and restrict the total number of queries that can be made to both w and π. A randomized query-restricted algorithm deciding the property testing problem is called a *tester* and when we allow oracle access to a proof we call it a *verifier*. The formal definition follows. (See [BGH+06] for more information on PCPPs.)

To simplify exposition we view w, π as functions from $[n] = \{1, \ldots, n\}$ and from $[n+1, n+\ell] = \{n+1, \ldots, n+\ell\}$ respectively to Σ and define the *word-proof pair* as the function $(w \circ \pi) : [n+\ell] \to \Sigma$ that is the concatenation of w and π. We call $(w \circ \pi)[i]$ a *word-symbol* whenever $i \leq n$ and a *proof symbol* when $i \in \{n+1, \ldots, n+\ell\}$. For a set of indices $I \subseteq [n+\ell]$ let $(w \circ \pi)|_I : I \to \Sigma$ denote the restriction of $w \circ \pi$ to I.

Definition 2.1 (Verifier, Tester). *A* query *of size q into a word of length n and proof of length ℓ is a pair $Q = (I, C)$ where $I \subseteq [n+\ell], |I| \leq q$ denotes the query's* index-set *and $C : \Sigma^I \to \{\mathsf{accept}, \mathsf{reject}\}$ is the query's* constraint. *Given word w and proof π let $Q(w \circ \pi) = C((w \circ \pi)|_I)$. A (q, n, ℓ)-*verifier *for a property of length n is a pair $\mathcal{V} = \langle \mathcal{Q}, D \rangle$ where*

- *$\mathcal{Q}$ is a finite set of queries of size at most q into a word of length n and proof of length ℓ.*

– *D is a distribution over $\mathcal{Q}$. We use $Q \sim_D \mathcal{Q}$ to denote that Q is sampled from $\mathcal{Q}$ according to distribution D.*

A q-tester *is a $(q, n, 0)$-verifier, i.e., a verifier that queries only the input.*

Often we will restrict our attention to a subclass of verifiers that use special kinds of constraints. In particular, we will be interested in *unique* and *linear* verifiers, defined next.

Definition 2.2 (Linear verifiers). *A query $Q = (I, C)$ is called $\mathbb{F}$-linear if $\Sigma = \mathbb{F}$ is a finite field and the set of assignments accepted by the query-constraint C forms an $\mathbb{F}$-linear space.*

A verifier is called $\mathbb{F}$-linear *if all its queries are $\mathbb{F}$-linear. Let $\mathbb{F}$-**linV** denote the set of $\mathbb{F}$-linear verifiers.*

Informally, if a (q, ℓ)-verifier solves the promise problem associated with P "with high probability" then we say P "has a PCPP" (with query complexity q and length ℓ). The *completeness* and *soundness* parameters quantify the success probability of the verifier. The formal definition follows.

Definition 2.3 (PCPP, Testability). *A property $P \subset \Sigma^n$ is said to have a PCPP of* length ℓ, query complexity q, completeness *parameter c and* soundness function $s : (0, 1] \to [0, 1]$ *if there exists a (q, n, ℓ)-verifier for the property satisfying the following pair of requirements.*

– **Completeness:** *For all $w \in P$, $\max_{\pi \in \Sigma^\ell} \Pr_{Q \sim_D \mathcal{Q}}[Q(w \circ \pi) = \mathsf{accept}] \geq c$. If $c = 1$, we say the verifier has* perfect *completeness.*
– **Soundness:** *For all $w \in \Sigma^n \setminus P$, $\min_{\pi \in \Sigma^\ell} \Pr_{Q \sim_D \mathcal{Q}}[Q(w \circ \pi) = \mathsf{reject}] \geq s(\delta(w, P))$, where $\delta(w, P)$ denotes the minimal fractional Hamming distance between w and an element of P.*

If P has a PCPP of length 0, query complexity q, completeness parameter c and soundness function s, we say that P is q-testable *with completeness c and soundness s.*

A verifier is said to be *adaptive* if its query indices depend on answers given to previous queries. The verifier defined above is *nonadaptive*. All results in this paper refer to nonadaptive verifiers with perfect completeness. We point out that all known PCPP constructions use nonadaptive verifiers and achieve perfect completeness so our deficiency bounds, stated next, apply to all of them (see [BHLM07] for further discussion).

2.2 Soundness Deficiency

We study the tradeoff between *proof length* and *soundness*. Our aim is to show that short PCPPs cannot attain the same soundness as long ones. To quantify this tradeoff we start by defining the *best soundness* that can be obtained by a class of verifiers with restricted proof length.

Definition 2.4 (Best Soundness). *Let $P \subseteq \Sigma^n$ be a property. For integers q, ℓ and $\delta \in [0,1]$, define the* best soundness $\mathcal{S}^P(q,\ell,\delta)$ *to be the maximum — taken over all (q,n,ℓ)-verifiers $\mathcal{V}$ — of the soundness of $\mathcal{V}$ with respect to inputs that are δ-far from P. Formally,*

$$\mathcal{S}^P(q,\ell,\delta) = \max_{(q,n,\ell)\text{-verifiers}} \quad \min_{w\circ\pi\in\Sigma^{n+\ell},\ \delta(w,P)=\delta} \quad \Pr_{Q\sim_D \mathcal{Q}}[Q(w\circ\pi)=\mathsf{reject}].$$

The best tester soundness *is $\mathcal{S}^P(q,0,\delta)$.*

The best soundness with respect to a class of verifiers $\mathbf{V}$, denoted $\mathcal{S}^P_{\mathbf{V}}(q,\ell,\delta)$, is defined by taking the maximum above over all (q,n,ℓ)-verifiers in $\mathbf{V}$. Notice that $\mathcal{S}^P_{\mathbf{V}}(q,\ell,\delta) \le \mathcal{S}^P(q,\ell,\delta)$.

The *soundness-deficiency*, defined next, is the reduction in best soundness incurred by 3-query verifiers limited to using short proofs. As customary in computational complexity, we measure the asymptotic deficiency over a family of properties of increasing length. In the remark following the definition, we further explain the need for complexity assumptions.

Definition 2.5 (Soundness deficiency). *For $\mathcal{P} = \{P \subseteq \Sigma^n \mid n \in \mathbb{Z}^+\}$ a family of properties, $\mathbf{V}$ a class of verifiers and $\ell : \mathbb{Z}^+ \to \mathbb{Z}^+$ a function measuring proof length, let the* soundness-deficiency *be the function measuring the decrease in soundness due to limited proof length. Formally, it is a function from $(0,1]$ to $[0,1]$ defined by*

$$\text{s-Def.}_{\mathbf{V}}[\mathcal{P},\ell](\delta) = \liminf_{n\to\infty} \mathcal{S}^{P_n}_{\mathbf{V}}(3,\infty,\delta) - \mathcal{S}^{P_n}_{\mathbf{V}}(3,\ell(n),\delta)\,.$$

For $\mathcal{C}$ a complexity class and $\mathcal{L}$ a family of complexity functions, we denote by $\text{s-Def.}_{\mathbf{V}}[\mathcal{C},\mathcal{L}](\delta)$ the soundness deficiency function taken over all $\mathcal{P} \in \mathcal{C}$ and $\ell \in \mathcal{L}$. Let in addition $\text{max-s-Def.}_{\mathbf{V}}[\mathcal{C},\mathcal{L}] = \max_{\delta\in(0,1]} \text{s-Def.}_{\mathbf{V}}[\mathcal{C},\mathcal{L}](\delta)$ be the maximal value that this function obtains over all $\delta \in (0,1]$. As before, whenever there is no restriction to a specific class of verifiers, the subscript $\mathbf{V}$ is omitted.

Remark 2.1 (Complexity restrictions). If no restriction is placed on the complexity of $\mathcal{P}$, then one may end up with trivial and uninteresting results. For instance, if $P_n \subset \{0,1\}^n$ is random, then with high probability any nondeterministic circuit deciding the promise problem associated with P_n requires size $2^{\Omega(n/\log n)}$. This implies that there are no constant query PCPPs with positive soundness and proof length $2^{o(n/\log n)}$. Thus, to get meaningful results, we focus on properties $\mathcal{P} \in \mathbf{P}/\text{poly}$ for which the existence of polynomial-length PCPPs is guaranteed.

2.3 Summary of Results

In this section, we summarize our main results bounding the maximum soundness deficiency for three different classes of verifiers – general verifiers, linear verifiers and unique verifiers. Deficiency bounds are obtained by bounding from

below the soundness of inspective verifiers that have access to long proofs and then bounding from above the soundness obtained by verifiers limited to short proofs. The next theorem shows the first bound, namely, that large soundness is obtainable if no restriction is placed on proof length. Its proof is based on the Fourier analytic approach introduced in [BCH+96] and appears in [BHLM07].

Theorem 2.1 (Best soundness with unbounded proof length). *Let $\mathbb{F}_p$ be a prime field. Every $\mathbb{F}_p$-linear property $P \subseteq \mathbb{F}_p^n$ has a 3-query $\mathbb{F}_p$-linear verifier using a proof of length $\leq |\mathbb{F}|^{\dim(P)} \leq |\mathbb{F}|^n$ that achieves soundness function $s(\delta) \geq \delta$. Formally, $\mathcal{S}^P_{\mathbf{linV}}\left(3, |\mathbb{F}_p|^{\dim(P)}, \delta\right) \geq \delta$.*

Deficiency of Short PCPPs. Our first main theorem says that for some properties, proofs of sub-exponential length incur constant soundness-deficiency. This deficiency can be reduced, but only at the expense of using exponentially long proofs.

Theorem 2.2 (Main). *Let $\alpha \in (0,1)$ be a positive constant and let $\mathcal{P} \triangleq \{P_n \subseteq \{0,1\}^n : n \in \mathbb{Z}^+\}$ be a family of binary linear properties (codes) with dual distance[4] at least αn. The properties in $\mathcal{P}$ have no sub-exponential PCPP's achieving soundness larger than $1/3$. Namely, for every $\varepsilon > 0$ there are $\beta > 0$ and $n_0 \in \mathbb{N}$ such that for any property $P_n \in \mathcal{P}$, $n > n_0$ the following is satisfied for all $\delta \in [0,1]$: $\mathcal{S}^{P_n}\left(3, 2^{\beta n}, \delta\right) \leq \frac{1}{3} + \varepsilon$.*

We show in Theorem 2.1 that every (in particular) binary linear property $P \subseteq \{0,1\}^n$ of dimension $k \leq n$ has a $(3, 2^k)$-verifier with soundness function $s(\delta) \geq \delta$. This implies constant deficiency for short PCPPs over the binary alphabet as formalized in the following corollary.

Corollary 2.1 (Soundness deficiency). *Let* **SUBEXP** *denote the set of sub-exponential functions, i.e., functions satisfying $f(n) = 2^{o(n)}$. There exists a family $\mathcal{P}$ of linear properties over $\{0,1\}$ such that* s-Def.$[\mathcal{P}, \mathbf{SUBEXP}](\delta) \geq \delta - \frac{1}{3}$. *Consequently, since there are words that are roughly $\frac{1}{2}$-far from the property $\mathcal{P}$, the maximal deficiency with sub-exponential proof length is at least $\frac{1}{6}$, i.e.,* max-s-Def.$[\mathbf{P}/poly, \mathbf{SUBEXP}] \geq \frac{1}{6}$.

Deficiency of Short Linear PCPPs. Our next main theorem presents stronger deficiency bounds for linear PCPPs and states the following intuitively appealing implication: Let p be a prime. Every $\mathbb{F}_p$-linear property that is "untestable" — in the sense that testers with small query complexity for it have low soundness — is also "unverifiable", i.e., 3-query $\mathbb{F}_p$-linear verifiers with short proofs must incur a large loss in soundness. Limiting our attention to linear verifiers seems natural in light of the fact that all current PCPP constructions produce linear verifiers for linear properties, as argued in [BHLM07].

[4] The dual distance of a linear property P is defined to be the minimal support-size of a nonzero vector in the space dual to P.

Theorem 2.3 (Main, linear case). *Let $P \subseteq \mathbb{F}^n$ be a $\mathbb{F}$-linear property. Let $s[\ell](\delta)$ denote the best soundness of a $(3,\ell)$-linear verifier for P, i.e., $s[\ell](\delta) = \mathcal{S}^P_{\mathbf{linV}}(3,\ell,\delta)$. Let $t[q](\delta)$ denote the best soundness of a q-tester for P, i.e., $t[q](\delta) = \mathcal{S}^P(q,0,\delta)$. Then*

$$s[\ell](\delta) \leq \min_{\varepsilon>0}\left\{t\left[\frac{36\log\ell}{\varepsilon}\right](\delta) + \frac{1}{2}\cdot\left(1-\frac{1}{|\mathbb{F}|}+\varepsilon\right)\right\}.$$

Using Theorem 2.1 again for arbitrary prime p we get the following bound on the deficiency of linear verifiers.

Corollary 2.2 (Soundness deficiency, linear case). *Let* **SUBEXP** *denote the set of subexponential functions, i.e., functions satisfying $f(n) = 2^{o(n)}$. For every prime field $\mathbb{F}_p$ there exists a family of $\mathbb{F}_p$-linear properties $\mathcal{P}$ such that* $\text{s-Def.}_{\mathbb{F}_p-\mathbf{linV}}[\mathcal{P},\mathbf{SUBEXP}](\delta) \geq \delta - \frac{1}{2}\cdot\left(1-\frac{1}{p}\right)$. *Consequently, the maximal deficiency of linear verifiers with sub-exponential proofs is at least $\frac{1}{2}\cdot(1-1/p)$. In other words,* $\text{max-s-Def.}_{\mathbb{F}_p-\mathbf{linV}}[\mathbb{F}_p - \textit{linear},\mathbf{SUBEXP}] \geq \frac{1}{2}\cdot\left(1-\frac{1}{p}\right)$.

We point out that even if we restrict our attention to families of linear properties with constant dual distance, the soundness deficiency can be very large. This last point is explained in detail in the proof of Corollary 2.2.

3 Proof of Length-Soundness Tradeoff for Linear PCPPs over $\mathbb{F}_2$

In this final section we sketch the proof of length-soundness tradeoff for linear PCPPs over the binary field. Although we consider only a special case, this section captures most of the ideas that are used for proving our main results.

Theorem 3.1 (Special case of Theorems 2.2 and 2.3). *Given any $\varepsilon > 0$ and a sub-exponential function $\ell \in 2^{o(n)}$, let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be an $\mathbb{F}_2$-linear property (code) with dual distance*[5] *at least $\left(\frac{3\log\ell}{\varepsilon}\right)$. Let $s[\ell](\delta)$ denote the best soundness of a $(3,n,\ell)$-linear verifier for C, i.e., $s[\ell](\delta) = \mathcal{S}^C_{\mathbf{linV}}(3,\ell,\delta)$. Then for every $\delta \leq 1/2$,*

$$s[\ell](\delta) \leq \frac{1}{3} + \varepsilon.$$

Consequently, the soundness deficiency of linear verifiers with sub-exponential proofs is $1/6 - o(1)$.

The following lemma is the main ingredient in the proof of Theorem 3.1.

Lemma 3.1 (Inspective Verifiability implies Testability). *Let $\mathcal{C}$ be any $[n,k,d]_2$-linear code. For every $q,\ell \in \mathbb{Z}^+$, $\varepsilon \in (0,1]$ and $s : (0,1] \to [0,1]$, the following holds. If $\mathcal{C}$ has a linear q-query inspective PCPP of length ℓ and soundness function s, then $\mathcal{C}$ has a $(\frac{q\log\ell}{\varepsilon})$-tester with soundness function $s-\varepsilon$.*

[5] See a detailed discussion about such codes in [BHLM07].

Before proving the lemma, we outline the proof of Theorem 3.1. Fix a $(3, n, \ell)$-linear verifier $\mathcal{V} = \langle \mathcal{Q}, D \rangle$ for $\mathcal{C}$, and denote by s the soundness function of $\mathcal{V}$. Let μ denote the probability that $\mathcal{V}$ makes an inspective query, namely, the probability (with respect to distribution D) that $\mathcal{V}$ probes at least one word bit.

Now, for two possible ranges of μ and for any $\delta \leq 1/2$ we design a "fooling" word-proof pair $(w \circ \pi)$, such that: (i) $\delta(w, \mathcal{C}) \geq \delta$; and (ii) $\mathcal{V}$ accepts $(w \circ \pi)$ with probability $2/3 - \varepsilon$, concluding the proof.

Case $\mu \leq 2/3$: Fix any $\delta \leq 1/2$ and consider the distribution over word-proof pairs $(w \circ \pi)$, where π is a legitimate proof of some fixed codeword $w' \in \mathcal{C}$ and $w = (w_\delta + r)$ is a sum of a fixed δ-far (from $\mathcal{C}$) word w_δ and a randomly chosen codeword $r \in \mathcal{C}$. It is not hard to show (see [BHLM07]) that

1. w is δ-far from $\mathcal{C}$
2. for any subset $I \in [n]$ of at most 3 coordinates of w, the $2^{|I|}$ possible values of $w \cap I$ are distributed uniformly (over the choices of r).

Clearly, all non-inspective queries are accepted with probability 1 since π is a legitimate proof. In addition, item 2 implies that the inspective queries are accepted with probability 1/2, since the constraints are all linear. Therefore, $s(\delta) \leq \mu/2 \leq 1/3$.

Case $\mu > 2/3$: Let $\mathcal{V}_i$ be a $(3, n, \ell)$-linear inspective verifier derived from $\mathcal{V}$ as follows: $\mathcal{V}_i$ picks a query $Q \in \mathcal{Q}$ according to distribution D. If Q is inspective then $\mathcal{V}_i$ proceeds exactly as $\mathcal{V}$. Otherwise, $\mathcal{V}_i$ immediately accepts (without making the query Q). Let s_i denote the soundness functions of $\mathcal{V}_i$. Since $\mathcal{V}_i$ is inspective, by Lemma 3.1 we get that for any $(\frac{3 \log \ell}{\varepsilon})$-query tester T for $\mathcal{C}$ having soundness function s_T, $s_i \leq s_T + \varepsilon$. We also know (by definition of $\mathcal{C}$) that any such tester has soundness $s_T \equiv 0$. Therefore, $s(\delta) \leq \mu \cdot (0 + \varepsilon) + 1 - \mu \leq 1/3 + \varepsilon$ for any $\delta \leq 1/2$.

3.1 Proof of Lemma 3.1

As mentioned in the introduction, we are going to view an inspective q-query linear verifier as a graph. An inspective graph (defined below) is a representation of an inspective verifier in the sense that a single invocation of the verifier corresponds to picking a random edge in the graph and making the set of queries given by the names of the end-vertices and the edge-label.

Definition 3.1 (Inspective Graphs). *A (q, n, ℓ)-inspective graph is a triplet $\mathcal{G} = (V, E, L_E)$ where $(V = \{0, 1, \ldots, l\}, E)$ is an undirected multigraph graph and $L_E : E \to (\mathbb{F}_2)^n_{\leq q}$ is a mapping of the edges to $\mathbb{F}_2$-vectors of dimension n and weight at most q*[6].

A word $w \in \mathbb{F}_2^n$ induces a labeling $L_E^{(w)} : V \to \mathbb{F}_2$ as follows: $L_E^{(w)}(e) = \langle L_E(e), w \rangle = \sum_{i=1}^n L_E(e)[i] \cdot w[i]$.

[6] The weight of a vector $v \in \mathbb{F}_2^n$ is the number of non-zero entries in the vector.

A labeling $\pi : V \to \mathbb{F}_2$ is said to satisfy edge $e = (u, v)$ with respect to w if $\pi(u) + \pi(v) + L_E^{(w)} = \pi(u) + \pi(v) + \langle L_E(e), w \rangle = 0$. A labeling $\pi : V \to \mathbb{F}_2$ is said to α-satisfy $\mathcal{G}$ with respect to w if it satisfies an α-fraction of the edges with respect to w.

$\mathcal{G}$ is said to be a (q, n, ℓ)-inspective proof graph for property P with soundness function $s : (0, 1] \to [0, 1]$ if the following two conditions are satisfied.

Perfect completeness: *For all $w \in P$ there exists a a labeling $\pi : V \to \mathbb{F}_2$ such that π 1-satisfies $\mathcal{G}$ with respect to w.*
Soundness: *For all $w \in \Sigma^n$, no labeling $\pi : V \to \mathbb{F}_2$ $(1 - s(\delta(w, P))$-satisfies $\mathcal{G}$ with respect to w where $\delta(w, P)$ denotes the minimal fractional Hamming distance between w and an element of P.*

The correspondence between linear inspective PCPPs and inspective graphs is as follows. First, assume without loss of generality that the verifier $\mathcal{V} = \langle \mathcal{Q}, D \rangle$ has uniform distribution D over $\mathcal{Q}$. This can be assumed by replacing $\mathcal{Q}$ with a multiset of queries where the number of "copies" of a query Q reflects the probability with which the Q is performed. The vertices $V \setminus \{0\}$ corresponds to the ℓ locations of the proof. The vertex 0 is a special vertex which corresponds to the bit 0. Any labeling of the vertices V (that satisfies $\pi(0) = 0$) corresponds to a proof. However, we may assume $\pi(0) = 0$ without loss of generality for the following reason. If a labeling π α-satisfies $\mathcal{G}$, so does the labeling $\pi + b$ defined as follows: $(\pi + b)(v) = \pi(v) + b$. Hence, we might assume that the labeling π satisfies $\pi(0) = 0$.

The edges of the graph correspond to the (inspective) tests of the verifier. Non self-loop Edges in $E \cap (V \setminus \{0\} \times V \setminus \{0\})$ correspond to inspective queries of i-size[7] 2, non self-loop edges in $E \cap (V \setminus \{0\} \times \{0\})$ to inspective queries of i-size 1 while the self-loop edges correspond to inspective queries of i-size 0. On input w, the verifier chooses an edge of the graph uniformly at random and checks if the labeling π satisfies the edge with respect to w. The multiplicity of an edge is proportional to the probability with which the PCPP verifier chooses the corresponding test.

Claim. Let $\mathcal{G} = (V, E, L_E)$ be a (q, n, ℓ)-inspective proof graph for some linear code $\mathcal{C}$. Suppose the vertices $v_1, v_2, \ldots, v_k, v_{k+1} = v_1$ form a cycle in the graph (V, E), then the vector $\sum_{i=1}^{k} L_E(v_i, v_{i+1})$ is a member of the dual code $\mathcal{C}^{\perp}$. □

Before proceeding we need some notation. For any graph G, let $V(G)$ and $E(G)$ denote the set of vertices and set of edges respectively of the graph G. For any subset $V' \subseteq V$ of vertices, let $G(V')$ denote the induced subgraph of G on the vertex set V'. Also, let $E(V') = E(G(V'))$. Similarly, let $E(V', V \setminus V')$ denote the set of edges between V' and $V \setminus V'$ (i.e., $E(V', V \setminus V') = E \cap (V' \times (V \setminus V'))$. For any connected graph G, define the radius of G ($rad(G)$) as follows:

$$rad(G) = \min_{v \in V} \max_{u \in V} d(u, v),$$

[7] The i-size of a query is the number of proof bits read by that query.

where $d(u,v)$ denotes the length of the shortest path between vertices u and v. Notice that for any connected graph, the distance between any two vertices is at most twice the radius of the graph.

Lemma 3.1 is proved by first showing that the inspective proof graph can be decomposed into components with small radii, and then transforming any inspective proof graph with components of small radii into a tester (Lemma 3.3)

Lemma 3.2 (Decomposition Lemma [LR99]). *For every $\varepsilon \in (0,1)$ and every multigraph $G = (V,E)$, there exists a subset of edges $E' \subseteq E$ of size at most $\varepsilon|E|$, such that every component of the graph $G_{\mathsf{Decomp.}} = (V, E \setminus E')$ has radius strictly less than $\log|V|/\varepsilon$.* □

We now show how to convert an inspective graph into a tester. The query complexity of the tester will be bounded by the length of the cycles in the graph. Thus, a graph with small radius will result in a tester of low query complexity.

Lemma 3.3 (low radius implies testability). *Let $\mathcal{C}$ be a $[n,k,d]_2$-code and $\mathcal{G} = (V,E,L)$ be a (q,n,ℓ)-inspective proof graph for the code $\mathcal{C}$ with soundness function s. If each of the components of the graph (V,E) have radius at most r, then $\mathcal{C}$ is $2qr$-testable with soundness function s.*

Proof. Let $\mathcal{G} = (V,E,L)$ be a (q,n,ℓ)-inspective proof graph for $\mathcal{C}$ with soundness s such that each component of the graph $G = (V,E)$ has radius at most r. Having radius at most r implies that there exists a spanning forest $F = (V,E')$ of G such that the height of each tree in F is at most r.

Consider the mapping $\tau : E \to \mathbb{F}^n$ of the edges to $\mathbb{F}_2$-vectors of length n defined as follows: If $e \in E(F)$, then $\tau(e) = 0$. Otherwise, $E(F) \cup \{e\}$ contains a unique cycle C. Then, define $\tau(e) = \sum_{e \in C} L(e)$. Since each tree of F is of height at most r, any such cycle C is of length at most $2r$. Also, from the definition of inspective proof graphs we have that $L(e)$ is a vector of weight at most q. Hence, $\tau(e)$ is a vector of weight at most $2qr$ and τ is a mapping from E to $(\mathbb{F}^n)_{\leq 2qr}$.

We define a tester $\mathcal{T}_{\mathcal{G}}$ based on the graph $\mathcal{G}$ as follows: On input $w \in \mathbb{F}^n$, it selects an edge e uniformly at random from $E(G)$ and checks if $\langle \tau(e), w \rangle = 0$. If yes it accepts, else it rejects. We now prove that $\mathcal{T}_{\mathcal{G}}$ is a $2qr$-tester for $\mathcal{C}$ with soundness function s.

Query Complexity. Since each $\tau(e)$ is of weight at most $2qr$, the tester queries at most $2qr$ locations of the word w.

Completeness. Suppose $w \in \mathcal{C}$. We have by Claim 3.1, that for each cycle C in G, we have $\sum_{e \in C} L(e)$ is a member of the dualcode $\mathcal{C}^{\perp}$. Therefore, $\langle \tau(e), w \rangle = 0$ for all edges e in G. In other words, the tester $\mathcal{T}_{\mathcal{G}}$ accepts with probability 1.

Soundness. Let w be any word. Consider the labeling $\pi : V \to \mathbb{F}$ defined as follows: For each tree T in the forest F, chose an arbitrary vertex v in T and set $\pi(v) = 0$. For any other vertex u in the tree, let $v = v_0, v_1, \ldots, v_k = u$ be the unique path in the tree T from v to u. Define $\pi(u) = \langle \sum L(v_i, v_{i+1}), w \rangle$. It is easy to check that if the labeling π α-satisfies $\mathcal{G}$ with respect to π, then the tester $\mathcal{T}_{\mathcal{G}}$ accepts w with probability exactly α. The soundness of the tester now follows from the soundness of the inspective proof graph $\mathcal{G}$

Lemma 3.1 now easily follows from the Decomposition Lemma and Lemma 3.3.

References

[ALM+98] Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. Journal ACM 45(3), 501–555 (1998)

[AS98] Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of NP. Journal ACM 45(1), 70–122 (1998)

[BCH+96] Bellare, M., Coppersmith, D., Håstad, J., Kiwi, M.A., Sudan, M.: Linearity testing in characteristic two. IEEE Transactions on Information Theory 42(6), 1781–1795 (1996)

[BGH+06] Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.: Robust PCPs of proximity, shorter PCPs and applications to coding. SICOMP 36(4), 889–974 (2006)

[BHLM07] Ben-Sasson, E., Harsha, P., Lachish, O., Matsliah, A.: Sound 3-query PCPPs are long. Technical Report TR07-127, ECCC (2007)

[BS05] Ben-Sasson, E., Sudan, M.: Simple PCPs with poly-log rate and query complexity. In: Proc. 37th ACM STOC, Baltimore, Maryland, pp. 266–275.

[Din07] Dinur, I.: The PCP theorem by gap amplification. Journal ACM 54(3), 12 (2007)

[DR06] Dinur, I., Reingold, O.: Assignment testers: Towards a combinatorial proof of the PCP Theorem. SICOMP 36, 975–1024 (2006)

[EKR04] Ergün, F., Kumar, R., Rubinfeld, R.: Fast approximate probabilistically checkable proofs. Information and Computation 189(2), 135–159 (2004)

[FF05] Fischer, E., Fortnow, L.: Tolerant versus intolerant testing for boolean properties. In: Proc. 20th IEEE CCC, San Jose, California, pp. 135–140.

[Fis01] Fischer, E.: The art of uninformed decisions: A primer to property testing. Bulletin of the European Association for Theoretical Computer Science 75, 97–126 (2001); The Computational Complexity Column

[GGR98] Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connection to learning and approximation. Journal ACM 45(4), 653–750 (1998)

[GR05] Guruswami, V., Rudra, A.: Tolerant locally testable codes. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 306–317. Springer, Heidelberg (2005)

[LR99] Leighton, F.T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal ACM 46(6), 787–832 (1999)

[PRR06] Parnas, M., Ron, D., Rubinfeld, R.: Tolerant property testing and distance approximation. Journal of Computer and System Sciences 72(6), 1012–1042 (2006)

[Sze99] Szegedy, M.: Many-valued logics and holographic proofs. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 676–686. Springer, Heidelberg (1999)

[Tre05] Trevisan, L.: Approximation algorithms for unique games. In: Proc. 46th IEEE FOCS, Pittsburgh, Pennsylvania, pp. 197–205

Approximative Methods for Monotone Systems of Min-Max-Polynomial Equations*

Javier Esparza, Thomas Gawlitza, Stefan Kiefer, and Helmut Seidl

Institut für Informatik
Technische Universität München, Germany
{esparza,gawlitza,kiefer,seidl}@in.tum.de

Abstract. A *monotone system of min-max-polynomial equations* (min-max-MSPE) over the variables $X_1, \ldots, X_n$ has for every i exactly one equation of the form $X_i = f_i(X_1, \ldots, X_n)$ where each $f_i(X_1, \ldots, X_n)$ is an expression built up from polynomials with non-negative coefficients, minimum- and maximum-operators. The question of computing least solutions of min-max-MSPEs arises naturally in the analysis of *recursive stochastic games* [5,6,14]. Min-max-MSPEs generalize MSPEs for which convergence speed results of Newton's method are established in [11,3]. We present the first methods for approximatively computing least solutions of min-max-MSPEs which converge at least linearly. Whereas the first one converges faster, a single step of the second method is cheaper. Furthermore, we compute ϵ-optimal positional strategies for the player who wants to maximize the outcome in a recursive stochastic game.

1 Introduction

In this paper we study *monotone systems of min-max polynomial equations* (min-max-MSPEs). A min-max-MSPE over the variables $X_1, \ldots, X_n$ contains for every $1 \leq i \leq n$ exactly one equation of the form $X_i = f_i(X_1, \ldots, X_n)$ where every $f_i(X_1, \ldots, X_n)$ is an expression built up from polynomials with non-negative coefficients, minimum- and maximum-operators. An example of such an equation is $X_1 = 3X_1X_2 + 5X_1^2 \wedge 4X_2$ (where $\wedge$ is the minimum-operator). The variables range over non-negative reals. Min-max-MSPEs are called monotone because f_i is a monotone function in all arguments.

Min-max-MSPEs naturally appear in the study of two-player stochastic games and competitive Markov decision processes, in which, broadly speaking, the next move is decided by one of the two players or by tossing a coin, depending on the game's position (see e.g. [12,7]). The min and max operators model the competition between the players. The product operator, which leads to non-linear equations, allows to deal with recursive stochastic games [5,6], a class of games with an infinite number of positions, and having as special case *extinction games*, games in which players influence with their actions the development of a population whose members reproduce and die, and the player's goals are to extinguish the population or keep it alive (see Section 3).

Min-max-MSPEs generalize several other classes of equation systems. If product is disallowed, we obtain systems of min-max *linear* equations, which appear in classical

* This work was in part supported by the DFG project *Algorithms for Software Model Checking*.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 698–710, 2008.

two-person stochastic games with a finite number of game positions. The problem of solving these systems has been thoroughly studied [1,8,9]. If both $\min$ and $\max$ are disallowed, we obtain monotone systems of polynomial equations, which are central to the study of recursive Markov chains and probabilistic pushdown systems, and have been recently studied in [4,11,3]. If only one of $\min$ or $\max$ is disallowed, we obtain a class of systems corresponding to recursive Markov decision processes [5]. All these models have applications in the analysis of probabilistic programs with procedures [14].

In vector form we denote a min-max-MSPE by $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ where $\boldsymbol{X}$ denotes the vector $(X_1, \ldots, X_n)$ and $\boldsymbol{f}$ denotes the vector $(f_1, \ldots, f_n)$. By Kleene's theorem, if a min-max-MSPE has a solution then it also has a *least* one, denoted by $\mu\boldsymbol{f}$, which is also the relevant solution for the applications mentioned above. Kleene's theorem also ensures that the iterative process $\boldsymbol{\kappa}^{(0)} = \mathbf{0}$, $\boldsymbol{\kappa}^{(k+1)} = \boldsymbol{f}(\boldsymbol{\kappa}^{(k)})$, $k \in \mathbb{N}$, the so-called Kleene sequence, converges to $\mu\boldsymbol{f}$. However, this procedure can converge very slowly: in the worst case, the number of accurate bits of the approximation grows with the *logarithm* of the number of iterations (cf. [4]). Thus, the goal is to replace the function $\boldsymbol{f}$ by an operator $G : \mathbb{R}^n \to \mathbb{R}^n$ such that the respective iterative process also converges to $\mu\boldsymbol{f}$ but faster. In [4,11,3] this problem was studied for min-max-MSPEs without the $\min$ and $\max$ operator. There, G was chosen as one step of the well-known Newton's method (cf. for instance [13]). This means that, for a given approximate $\boldsymbol{x}^{(k)}$, the next approximate $\boldsymbol{x}^{(k+1)} = G(\boldsymbol{x}^{(k)})$ is determined by the *unique solution* of a linear equation system which is obtained from the first order Taylor approximation of $\boldsymbol{f}$ at $\boldsymbol{x}^{(k)}$. It was shown that this choice guarantees *linear convergence*, i.e., the number of accurate bits grows *linearly* in the number of iterations. Notice that when characterizing the convergence behavior the term linear does not refer to the size of $\boldsymbol{f}$.

However, this technique no longer works for arbitrary min-max-MSPEs. If we approximate $\boldsymbol{f}$ at $\boldsymbol{x}^{(k)}$ through its first order Taylor approximation at $\boldsymbol{x}^{(k)}$ there is no guarantee that the next approximate still lies below the least solution, and the sequence of approximants may even diverge. For this reason, the PReMo tool [14] uses round-robin iteration for min-max-MSPEs, an optimization of Kleene iteration. Unfortunately, this technique also exhibits "logarithmic" convergence behavior in the worst case.

In this paper we overcome the problem of Newton's method. Instead of approximating $\boldsymbol{f}$ (at the current approximate $\boldsymbol{x}^{(k)}$) by a linear function, both of our methods approximate $\boldsymbol{f}$ by a *piecewise* linear function. In contrast to the applications of Newton's method in [4,11,3], this approximation may not have a *unique fixpoint*, but it has a *least fixpoint* which we use as the next approximate $\boldsymbol{x}^{(k+1)} = G(\boldsymbol{x}^{(k)})$. Our first method uses an approximation of $\boldsymbol{f}$ at $\boldsymbol{x}^{(k)}$ whose least fixpoint can be determined using the algorithm for *systems of rational equations* from [9]. The approximation of $\boldsymbol{f}$ at $\boldsymbol{x}^{(k)}$ used by our second method allows to use linear programming to compute $\boldsymbol{x}^{(k+1)}$. Our methods are the first algorithms for approximatively computing $\mu\boldsymbol{f}$ which converge at least linearly, provided that $\boldsymbol{f}$ is quadratic, an easily achievable normal form.

The rest of the paper is organized as follows. In Section 2 we introduce basic concepts and state some important facts about min-max-MSPEs. A class of games which can be analyzed using our techniques is presented in Section 3. Our main contribution, the two approximation methods, is presented and analyzed in Sections 4 and 5. In Section 6 we study the relation between our two approaches and compare them to

previous work. We conclude in Section 7. Missing proofs can be found in a technical report [2].

2 Notations, Basic Concepts and a Fundamental Theorem

As usual, $\mathbb{R}$ and $\mathbb{N}$ denote the set of real and natural numbers. We assume $0 \in \mathbb{N}$. We write $\mathbb{R}_{\geq 0}$ for the set of non-negative real numbers. We use bold letters for vectors, e.g. $\boldsymbol{x} \in \mathbb{R}^n$. In particular $\mathbf{0}$ denotes the vector $(0, \ldots, 0)$. The transpose of a matrix or a vector is indicated by the superscript $^\top$. We assume that the vector $\boldsymbol{x} \in \mathbb{R}^n$ has the components $x_1, \ldots, x_n$. Similarly, the i-th component of a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is denoted by f_i. As in [3], we say that $\boldsymbol{x} \in \mathbb{R}^n$ has $i \in \mathbb{N}$ valid bits of $\boldsymbol{y} \in \mathbb{R}^n$ iff $|x_j - y_j| \leq 2^{-i}|y_j|$ for $j = 1, \ldots, n$. We identify a linear function from $\mathbb{R}^n$ to $\mathbb{R}^m$ with its representation as a matrix from $\mathbb{R}^{m \times n}$. The identity matrix is denoted by I. The *Jacobian* of a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ at $\boldsymbol{x} \in \mathbb{R}^n$ is the matrix of all first-order partial derivatives of $\boldsymbol{f}$ at $\boldsymbol{x}$, i.e., the $m \times n$-matrix with the entry $\frac{\partial f_i}{\partial X_j}(\boldsymbol{x})$ in the i-th row and the j-th column. We denote it by $\boldsymbol{f}'(\boldsymbol{x})$.

The partial order $\leq$ on $\mathbb{R}^n$ is defined by setting $\boldsymbol{x} \leq \boldsymbol{y}$ iff $x_i \leq y_i$ for all $i = 1, \ldots, n$. We write $\boldsymbol{x} < \boldsymbol{y}$ iff $\boldsymbol{x} \leq \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$. The operators $\wedge$ and $\vee$ are defined by $x \wedge y := \min\{x, y\}$ and $x \vee y := \max\{x, y\}$ for $x, y \in \mathbb{R}$. These operators are also extended component-wise to $\mathbb{R}^n$ and point-wise to $\mathbb{R}^n$-valued functions. A function $\boldsymbol{f} : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ it called *monotone* on $M \subseteq D$ iff $\boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{f}(\boldsymbol{y})$ for every $\boldsymbol{x}, \boldsymbol{y} \in M$ with $\boldsymbol{x} \leq \boldsymbol{y}$. Let $X \subseteq \mathbb{R}^n$ and $\boldsymbol{f} : X \to X$. A vector $\boldsymbol{x} \in X$ is called *fixpoint* of $\boldsymbol{f}$ iff $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$. It is the *least fixpoint* of $\boldsymbol{f}$ iff $\boldsymbol{y} \geq \boldsymbol{x}$ for every fixpoint $\boldsymbol{y} \in X$ of $\boldsymbol{f}$. If it exists we denote the least fixpoint of $\boldsymbol{f}$ by $\mu\boldsymbol{f}$. We call $\boldsymbol{f}$ *feasible* iff $\boldsymbol{f}$ has some fixpoint $\boldsymbol{x} \in X$.

Let us fix a set $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ of variables. We call a vector $\boldsymbol{f} = (f_1, \ldots, f_m)$ of polynomials $f_1, \ldots, f_m$ in the variables $X_1, \ldots, X_n$ a *system of polynomials*. $\boldsymbol{f}$ is called *linear* (resp. *quadratic*) iff the degree of each f_i is at most 1 (resp. 2), i.e., every monomial contains at most one variable (resp. two variables). As usual, we identify $\boldsymbol{f}$ with its interpretation as a function from $\mathbb{R}^n$ to $\mathbb{R}^m$. As in [11,3] we call $\boldsymbol{f}$ a *monotone system of polynomials* (MSP for short) iff all coefficients are non-negative.

Min-max-MSPs. Given polynomials $f_1, \ldots, f_k$ we call $f_1 \wedge \cdots \wedge f_k$ a *min-polynomial* and $f_1 \vee \cdots \vee f_k$ a *max-polynomial*. A function that is either a min- or a max-polynomial is also called *min-max-polynomial*. We call $\boldsymbol{f} = (f_1, \ldots, f_n)$ a *system of min-polynomials* iff every component f_i is a min-polynomial. The definition of *systems of max-polynomials* and *systems of min-max-polynomials* is analogous. A system of min-max-polynomials is called *linear* (resp. *quadratic*) iff all occurring polynomials are linear (resp. *quadratic*). By introducing auxiliary variables every system of min-max-polymials can be transformed into a *quadratic* one in time linear in the size of the system (cf. [11]). A system of min-max-polynomials where all coefficients are from $\mathbb{R}^n_{\geq 0}$ is called a *monotone system of min-max-polynomials* (*min-max-MSP*) for short. The terms *min-MSP* and *max-MSP* are defined analogously.

Example 1. $\boldsymbol{f}(x_1, x_2) = (\frac{1}{2}x_2^2 + \frac{1}{2} \wedge 3, x_1 \vee 2)^\top$ is a quadratic min-max-MSP.

A min-max-MSP $\boldsymbol{f} = (f_1, \ldots, f_n)^\top$ can be considered as a mapping from $\mathbb{R}^n_{\geq 0}$ to $\mathbb{R}^n_{\geq 0}$. The Kleene sequence $(\boldsymbol{\kappa}_{\boldsymbol{f}}^{(k)})_{k\in\mathbb{N}}$ is defined by $\boldsymbol{\kappa}_{\boldsymbol{f}}^{(k)} := \boldsymbol{f}^k(\mathbf{0})$, $k \in \mathbb{N}$. We have:

Lemma 1. *Let $\boldsymbol{f} : \mathbb{R}^n_{\geq 0} \to \mathbb{R}^n_{\geq 0}$ be a min-max-MSP. Then: (1) $\boldsymbol{f}$ is monotone and continuous on $\mathbb{R}^n_{\geq 0}$; and (2) If $\boldsymbol{f}$ is feasible (i.e., $\boldsymbol{f}$ has some fixpoint), then $\boldsymbol{f}$ has a least fixpoint $\mu\boldsymbol{f}$ and $\mu\boldsymbol{f} = \lim_{k\to\infty} \boldsymbol{\kappa}_{\boldsymbol{f}}^{(k)}$.*

Strategies. Assume that $\boldsymbol{f}$ denotes a system of min-max-polynomials. A $\vee$-*strategy* σ for $\boldsymbol{f}$ is a function that maps every max-polynomial $f_i = f_{i,1} \vee \cdots \vee f_{i,k_i}$ occurring in $\boldsymbol{f}$ to one of the $f_{i,j}$'s and every min-polynomial f_i to f_i. We also write f_i^σ for $\sigma(f_i)$. Accordingly, a $\wedge$-*strategy* π for $\boldsymbol{f}$ is a function that maps every min-polynomial $f_i = f_{i,1} \wedge \cdots \wedge f_{i,k}$ occurring in $\boldsymbol{f}$ to one of the $f_{i,j}$'s and every max-polynomial f_i to f_i. We denote the set of $\vee$-strategies for $\boldsymbol{f}$ by $\Sigma_{\boldsymbol{f}}$ and the set of $\wedge$-strategies for $\boldsymbol{f}$ by $\Pi_{\boldsymbol{f}}$. For $s \in \Sigma_{\boldsymbol{f}} \cup \Pi_{\boldsymbol{f}}$, we write $\boldsymbol{f}^s$ for $(\boldsymbol{f}_1^s, \ldots, \boldsymbol{f}_n^s)^\top$. We define $\Pi^*_{\boldsymbol{f}} := \{\pi \in \Pi_{\boldsymbol{f}} \mid \boldsymbol{f}^\pi \text{ is feasible}\}$. We drop the subscript whenever it is clear from the context.

Example 2. Consider $\boldsymbol{f}$ from Example 1. Then $\pi : \frac{1}{2}x_2^2 + \frac{1}{2} \wedge 3 \mapsto 3,\ x_1 \vee 2 \mapsto x_1 \vee 2$ is a $\wedge$-strategy. The max-MSP $\boldsymbol{f}^\pi$ is given by $\boldsymbol{f}^\pi(x_1, x_2)^\top = (3, x_1 \vee 2)^\top$. □

We collect some elementary facts concerning strategies.

Lemma 2. *Let $\boldsymbol{f}$ be a feasible min-max-MSP. Then (1) $\mu\boldsymbol{f}^\sigma \leq \mu\boldsymbol{f}$ for every $\sigma \in \Sigma$; (2) $\mu\boldsymbol{f}^\pi \geq \mu\boldsymbol{f}$ for every $\pi \in \Pi^*$; (3) $\mu\boldsymbol{f}^\pi = \mu\boldsymbol{f}$ for some $\pi \in \Pi^*$.*

In [5] the authors consider a subclass of recursive stochastic games for which they prove that a positional optimal strategy exists for the player who wants to maximize the outcome (Theorem 2). The outcome of such a game is the least fixpoint of some min-max-MSP $\boldsymbol{f}$. In our setting, Theorem 2 of [5] implies that there exists a $\vee$-strategy σ such that $\mu\boldsymbol{f}^\sigma = \mu\boldsymbol{f}$ — provided that $\boldsymbol{f}$ is derived from such a recursive stochastic game. Example 3 shows that this property does not hold for arbitrary min-max-MSPs.

Example 3. Consider $\boldsymbol{f}$ from Example 1. Let $\sigma_1, \sigma_2 \in \Sigma$ be defined by $\sigma_1(x_1 \vee 2) = x_1$ and $\sigma_2(x_1 \vee 2) = 2$. Then $\mu\boldsymbol{f}^{\sigma_1} = (1, 1)^\top$, $\mu\boldsymbol{f}^{\sigma_2} = (\frac{5}{2}, 2)^\top$ and $\mu\boldsymbol{f} = (3, 3)^\top$. □

The proof of the following fundamental result is inspired by the proof of Theorem 2 in [5]. Although the result looks very natural it is non-trivial to prove.

Theorem 1. *Let $\boldsymbol{f}$ be a feasible max-MSP. Then $\mu\boldsymbol{f}^\sigma = \mu\boldsymbol{f}$ for some $\sigma \in \Sigma$.*

3 A Class of Applications: Extinction Games

In order to illustrate the interest of min-max-MSPs we consider *extinction games*, which are special stochastic games. Consider a world of n different species $s_1, \ldots, s_n$. Each species s_i is controlled by one of two adversarial players. For each s_i there is a non-empty set A_i of actions. An action $a \in A_i$ replaces a single individual of species s_i by other individuals specified by the action a. The actions can be probabilistic. E.g., an action could transform an adult rabbit to zero individuals with probability 0.2, to an adult rabbit with probability 0.3 and to an adult and a baby rabbit with probability 0.5.

Another action could transform an adult rabbit to a fat rabbit. The max-player (min-player) wants to maximize (minimize) the probability that some initial population is extinguished. During the game each player continuously chooses an individual of a species s_i controlled by her/him and applies an action from A_i to it. Note that actions on different species are never in conflict and the execution order is irrelevant. What is the probability that the population is extinguished if the players follow optimal strategies?

To answer those questions we set up a min-max-MSP $\boldsymbol{f}$ with one min-max-polynomial for each species, thereby following [10,5]. The variables X_i represent the probability that a population with only a single individual of species s_i is extinguished. In the rabbit example we have $X_{\text{adult}} = 0.2 + 0.3X_{\text{adult}} + 0.5X_{\text{adult}}X_{\text{baby}} \vee X_{\text{fat}}$, assuming that the adult rabbits are controlled by the max-player. The probability that an initial population with p_i individuals of species s_i is extinguished is given by $\prod_{i=1}^{n}((\mu \boldsymbol{f})_i)^{p_i}$. The stochastic termination games of [5,6,14] can be considered as extinction games. In the following we present another instance.

The primaries game. Hillary Clinton has to decide her strategy in the primaries. Her team estimates that undecided voters have not yet decided to vote for her for three possible reasons: they consider her (a) cold and calculating, (b) too much part of Washington's establishment, or (c) they listen to Obama's campaign. So the team decides to model those problems as species in an extinction game. The larger the population of a species, the more influenced is an undecided voter by the problem. The goal of Clinton's team is to maximize the extinction probabilities.

Clinton's possible actions for problem (a) are *showing emotions* or *concentrating on her program*. If she shows emotions, her team estimates that the individual of problem (a) is removed with probability 0.3, but with probability 0.7 the action backfires and produces yet another individual of (a). This and the effect of concentrating on her program can be read off from Equation (1) below. For problem (b), Clinton can choose between concentrating on her voting record or her statement "I'll be ready from day 1". Her team estimates the effect as given in Equation (2). Problem (c) is controlled by Obama, who has the choice between his "change" message, or attacking Clinton for her position on Iraq, see Equation (3).

$$X_a = 0.3 + 0.7X_a^2 \quad \vee \quad 0.1 + 0.9X_c \tag{1}$$

$$X_b = 0.1 + 0.9X_c \quad \vee \quad 0.4X_b + 0.3X_c + 0.3 \tag{2}$$

$$X_c = 0.5X_b + 0.3X_b^2 + 0.2 \quad \wedge \quad 0.5X_a + 0.4X_aX_b + 0.1X_b \tag{3}$$

What should Clinton and Obama do? What are the extinction probabilities, assuming perfect strategies? In the next sections we show how to efficiently solve these problems.

4 The τ-Method

Assume that $\boldsymbol{f}$ denotes a feasible min-max-MSP. In this section we present our first method for computing $\mu\boldsymbol{f}$ approximatively. We call it τ-method. This method computes, for each approximate $\boldsymbol{x}^{(i)}$, the next approximate $\boldsymbol{x}^{(i+1)}$ as the least fixpoint of a piecewise linear approximation $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{x}^{(i)}) \vee \boldsymbol{x}^{(i)}$ (see below) of $\boldsymbol{f}$ at $\boldsymbol{x}^{(i)}$. This approximation is a system of *linear* min-max-polynomials where all coefficients of monomials

of degree 1 are non-negative. Here, we call such a system a *monotone linear min-max-system* (*min-max-MLS* for short). Note that a min-max-MLS $\boldsymbol{f}$ is not necessarily a min-max-MSP, since negative coefficients of monomials of degree 0 are allowed, e.g. the min-max-MLS $f(x_1) = x_1 - 1$ is not a min-max-MSP.

In [9] a min-max-MLS $\boldsymbol{f}$ is considered as a system of equations (called *system of rational equations* in [9]) which we denote by $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ in vector form. We identify a min-max-MLS $\boldsymbol{f}$ with its interpretation as a function from $\overline{\mathbb{R}}^n$ to $\overline{\mathbb{R}}^n$ ($\overline{\mathbb{R}}$ denotes the complete lattice $\mathbb{R} \cup \{-\infty, \infty\}$). Since $\boldsymbol{f}$ is monotone on $\overline{\mathbb{R}}^n$, it has a least fixpoint $\mu\boldsymbol{f} \in \overline{\mathbb{R}}^n$ which can be computed using the strategy improvement algorithm from [9].

We now define the min-max-MLS $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{y})$, a piecewise linear approximation of $\boldsymbol{f}$ at $\boldsymbol{y}$. As a first step, let us consider a monotone polynomial $f : \mathbb{R}^n_{\geq 0} \to \mathbb{R}_{\geq 0}$. Given some approximate $\boldsymbol{y} \in \mathbb{R}^n_{\geq 0}$, a linear approximation $\mathcal{L}(f, \boldsymbol{y}) : \mathbb{R}^n \to \mathbb{R}$ of f at $\boldsymbol{y}$ is given by the first order Taylor approximation at $\boldsymbol{y}$, i.e.,

$$\mathcal{L}(f, \boldsymbol{y})(\boldsymbol{x}) := f(\boldsymbol{y}) + f'(\boldsymbol{y})(\boldsymbol{x} - \boldsymbol{y}), \qquad \boldsymbol{x} \in \mathbb{R}^n.$$

This is precisely the linear approximation which is used for Newton's method. Now consider a max-polynomial $f = f_1 \vee \cdots \vee f_k : \mathbb{R}^n \to \mathbb{R}$. We define the approximation $\mathcal{L}(f, \boldsymbol{y}) : \mathbb{R}^n \to \mathbb{R}$ of f at $\boldsymbol{y}$ by $\mathcal{L}(f, \boldsymbol{y}) := \mathcal{L}(f_1, \boldsymbol{y}) \vee \cdots \vee \mathcal{L}(f_k, \boldsymbol{y})$. We emphasize that in this case, $\mathcal{L}(f, \boldsymbol{y})$ is in general not a linear function but a linear max-polynomial. Accordingly, for a min-MSP $f = f_1 \wedge \cdots \wedge f_k : \mathbb{R}^n \to \mathbb{R}$, we define $\mathcal{L}(f, \boldsymbol{y}) := \mathcal{L}(f_1, \boldsymbol{y}) \wedge \cdots \wedge \mathcal{L}(f_k, \boldsymbol{y})$. In this case $\mathcal{L}(f, \boldsymbol{y})$ is a linear min-polynomial. Finally, for a min-max-MSP $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$, we define the approximation $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{y}) : \mathbb{R}^n \to \mathbb{R}^n$ of $\boldsymbol{f}$ at $\boldsymbol{y}$ by $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{y}) := (\mathcal{L}(f_1, \boldsymbol{y}), \ldots, \mathcal{L}(f_n, \boldsymbol{y}))^\top$ which is a min-max-MLS.

Example 4. Consider the min-max-MSP $\boldsymbol{f}$ from Example 1. The approximation $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, (\frac{1}{2}, \frac{1}{2}))$ is given by $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, (\frac{1}{2}, \frac{1}{2}))(x_1, x_2) = \big(\frac{1}{2}x_2 + \frac{3}{8} \wedge 3,\ x_1 \vee 2\big)$. □

Using the approximation $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{x}^{(i)})$ we define the operator $\mathcal{N}_{\boldsymbol{f}} : \mathbb{R}^n_{\geq 0} \to \mathbb{R}^n_{\geq 0}$ which gives us, for an approximate $\boldsymbol{x}^{(i)}$, the next approximate $\boldsymbol{x}^{(i+1)}$ by

$$\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x}) := \mu(\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{x}) \vee \boldsymbol{x}), \qquad \boldsymbol{x} \in \mathbb{R}^n_{\geq 0}.$$

Observe that $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{x}) \vee \boldsymbol{x}$ is still a min-max-MLS (at least after introducing auxiliary variables in order to eliminate components which contain $\vee$- and $\wedge$-operators).

Example 5. In Example 4 we have: $\mathcal{N}_{\boldsymbol{f}}(\frac{1}{2}, \frac{1}{2}) = \mu(\boldsymbol{\mathcal{L}}(\boldsymbol{f}, (\frac{1}{2}, \frac{1}{2})) \vee (\frac{1}{2}, \frac{1}{2})^\top) = (\frac{11}{8}, 2)^\top$.

We collect basic properties of $\mathcal{N}_{\boldsymbol{f}}$ in the following lemma:

Lemma 3. *Let $\boldsymbol{f}$ be a feasible min-max-MSP and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n_{\geq 0}$. Then:*

1. $\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}) \leq \mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x})$;
2. $\boldsymbol{x} = \mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x})$ *whenever* $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$;
3. *(Monotonicity of $\mathcal{N}_{\boldsymbol{f}}$)* $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x}) \leq \mathcal{N}_{\boldsymbol{f}}(\boldsymbol{y})$ *whenever* $\boldsymbol{x} \leq \boldsymbol{y}$;
4. $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x}) \leq \boldsymbol{f}(\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x}))$ *whenever* $\boldsymbol{x} \leq \boldsymbol{f}(\boldsymbol{x})$;
5. $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x}) \geq \mathcal{N}_{\boldsymbol{f}^\sigma}(\boldsymbol{x})$ *for every $\vee$-strategy* $\sigma \in \Sigma$;
6. $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x}) \leq \mathcal{N}_{\boldsymbol{f}^\pi}(\boldsymbol{x})$ *for every $\wedge$-strategy* $\pi \in \Pi$;
7. $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x}) = \mathcal{N}_{\boldsymbol{f}^\pi}(\boldsymbol{x})$ *for some $\wedge$-strategy* $\pi \in \Pi$.

In particular Lemma 3 implies that the least fixpoint of $\mathcal{N}_{\boldsymbol{f}}$ is equal to the least fixpoint of $\boldsymbol{f}$. Moreover, iteration based on $\mathcal{N}_{\boldsymbol{f}}$ is at least as fast as Kleene iteration. We therefore use this operator for computing approximates to the least fixpoint. Formally, we define:

Definition 1. *We call the sequence* $(\boldsymbol{\tau}_{\boldsymbol{f}}^{(k)})$ *of approximates defined by* $\boldsymbol{\tau}_{\boldsymbol{f}}^{(k)} := \mathcal{N}_{\boldsymbol{f}}^{k}(\mathbf{0})$ *for* $k \in \mathbb{N}$ *the* $\boldsymbol{\tau}$*-sequence for* $\boldsymbol{f}$*. We drop the subscript if it is clear from the context.*

Proposition 1. *Let* $\boldsymbol{f}$ *be a feasible min-max-MSP. The* $\boldsymbol{\tau}$*-sequence* $(\boldsymbol{\tau}^{(k)})$ *for* $\boldsymbol{f}$ *(see definition 1) is monotonically increasing, bounded from above by* $\mu\boldsymbol{f}$*, and converges to* $\mu\boldsymbol{f}$*. Moreover,* $\boldsymbol{\kappa}^{(k)} \leq \boldsymbol{\tau}^{(k)}$ *for all* $k \in \mathbb{N}$*.*

We now show that the new approximation method converges at least *linearly* to the least fixpoint. Theorem 6.2 of [3] implies the following lemma about the convergence of Newton's method for MSPs, i.e., systems without maxima and minima.

Lemma 4. *Let* $\boldsymbol{f}$ *be a feasible quadratic MSP. The sequence* $(\boldsymbol{\tau}^{(k)})_{k\in\mathbb{N}}$ *converges linearly to* $\mu\boldsymbol{f}$*. More precisely, there is a* $k_{\boldsymbol{f}} \in \mathbb{N}$ *such that* $\boldsymbol{\tau}^{(k_{\boldsymbol{f}}+i\cdot(n+1)\cdot 2^n)}$ *has at least* i *valid bits of* $\mu\boldsymbol{f}$ *for every* $i \in \mathbb{N}$*.*

We emphasize that linear convergence is the worst case. In many practical examples, in particular if the matrix $I - \boldsymbol{f}'(\mu\boldsymbol{f})$ is invertible, Newton's method converges *exponentially*. We mean by this that the number of accurate bits of the approximation grows *exponentially* in the number of iterations.

As a first step towards our main result for this section, we use Lemma 4 to show that our approximation method converges linearly whenever $\boldsymbol{f}$ is a max-MSPs. In this case we obtain the same convergence speed as for MSPs.

Lemma 5. *Let* $\boldsymbol{f}$ *be a feasible max-MSP. Let* $M := \{\sigma \in \Sigma \mid \mu\boldsymbol{f}^{\sigma} = \mu\boldsymbol{f}\}$*. The set* M *is non-empty and* $\boldsymbol{\tau}_{\boldsymbol{f}}^{(i)} \geq \boldsymbol{\tau}_{\boldsymbol{f}^{\sigma}}^{(i)}$ *for all* $\sigma \in M$ *and* $i \in \mathbb{N}$*.*

Proof. Theorem 1 implies that there exists a $\vee$-strategy $\sigma \in \Sigma$ such that $\mu\boldsymbol{f}^{\sigma} = \mu\boldsymbol{f}$. Thus M is non-empty. Let $\sigma \in M$. By induction on k Lemma 3 implies $\boldsymbol{\tau}_{\boldsymbol{f}}^{(k)} = \mathcal{N}_{\boldsymbol{f}}^{k}(\mathbf{0}) \geq \mathcal{N}_{\boldsymbol{f}^{\sigma}}^{k}(\mathbf{0}) = \boldsymbol{\tau}_{\boldsymbol{f}^{\sigma}}^{(k)}$ for every $k \in \mathbb{N}$. □

Combining Lemma 4 and Lemma 5 we get linear convergence for max-MSPs:

Theorem 2. *Let* $\boldsymbol{f}$ *be a feasible quadratic max-MSP. The* $\boldsymbol{\tau}$*-sequence* $(\boldsymbol{\tau}^{(k)})$ *for* $\boldsymbol{f}$ *(see definition 1) converges linearly to* $\mu\boldsymbol{f}$*. More precisely, there is a* $k_{\boldsymbol{f}} \in \mathbb{N}$ *such that* $\boldsymbol{\tau}^{(k_{\boldsymbol{f}}+i\cdot(n+1)\cdot 2^n)}$ *has at least* i *valid bits of* $\mu\boldsymbol{f}$ *for every* $i \in \mathbb{N}$*.*

A direct consequence of Lemma 5 is that the $\boldsymbol{\tau}$-sequence $(\boldsymbol{\tau}_{\boldsymbol{f}}^{(i)})$ converges exponentially if $(\boldsymbol{\tau}_{\boldsymbol{f}^{\sigma}}^{(i)})$ converges exponentially for some $\sigma \in \Sigma$ with $\mu\boldsymbol{f}^{\sigma} = \mu\boldsymbol{f}$. This is in particular the case if the matrix $I - (\boldsymbol{f}^{\sigma})'(\mu\boldsymbol{f})$ is invertible. In order to extend this result to min-max-MSPs we state the following lemma which enables us to relate the sequence $(\boldsymbol{\tau}_{\boldsymbol{f}}^{(i)})$ to the sequences $(\boldsymbol{\tau}_{\boldsymbol{f}^{\pi}}^{(i)})$ where $\mu\boldsymbol{f}^{\pi} = \mu\boldsymbol{f}$.

Lemma 6. *Let* $\boldsymbol{f}$ *be a feasible min-max-MSP and* m *denote the number of strategies* $\pi \in \Pi$ *with* $\mu\boldsymbol{f} = \mu\boldsymbol{f}^{\pi}$*. There is a constant* $k \in \mathbb{N}$ *such that for all* $i \in \mathbb{N}$ *there exists some strategy* $\pi \in \Pi$ *with* $\mu\boldsymbol{f} = \mu\boldsymbol{f}^{\pi}$ *and* $\boldsymbol{\tau}_{\boldsymbol{f}^{\pi}}^{(i)} \leq \boldsymbol{\tau}_{\boldsymbol{f}}^{(k+m\cdot i)}$*.*

We now present the main result of this section which states that our approximation method converges at least linearly also in the general case, i.e., for min-max-MSPs.

Theorem 3. *Let $\boldsymbol{f}$ be a feasible quadratic min-max-MSP and m denote the number of strategies $\pi \in \Pi$ with $\mu\boldsymbol{f} = \mu\boldsymbol{f}^{\pi}$. The $\boldsymbol{\tau}$-sequence $(\boldsymbol{\tau}^{(k)})$ for $\boldsymbol{f}$ (see definition 1) converges linearly to $\mu\boldsymbol{f}$. More precisely, there is a $k_{\boldsymbol{f}} \in \mathbb{N}$ such that $\boldsymbol{\tau}^{(k_{\boldsymbol{f}}+i\cdot m\cdot(n+1)\cdot 2^n)}$ has at least i valid bits of $\mu\boldsymbol{f}$ for every $i \in \mathbb{N}$.*

The upper bound on the convergence rate provided by Theorem 2 is by the factor m worse than the upper bound obtained for MSPs. Since m is the number of strategies $\pi \in \Pi$ with $\mu\boldsymbol{f}^{\pi} = \mu\boldsymbol{f}$, m is trivially bounded by $|\Pi|$ but is usually much smaller. The $\boldsymbol{\tau}$-sequence $(\boldsymbol{\tau}_{\boldsymbol{f}}^{(i)})$ converges exponentially whenever $(\boldsymbol{\tau}_{\boldsymbol{f}^{\pi}}^{(i)})$ converges exponentially for every π with $\mu\boldsymbol{f}^{\pi} = \mu\boldsymbol{f}$ (see [2]). The latter condition is typically satisfied (see the discussion after Theorem 2).

In order to determine the approximate $\boldsymbol{\tau}^{(i+1)} = \mathcal{N}_{\boldsymbol{f}}(\boldsymbol{\tau}^{(i)})$ from $\boldsymbol{\tau}^{(i)}$ we must compute the least fixpoint of the min-max-MLS $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{\tau}^{(i)})\vee\boldsymbol{\tau}^{(i)}$. This can be done by using the strategy improvement algorithm from [9]. The algorithm iterates over $\vee$-strategies. For each strategy it solves a linear program or alternatively iterates over $\wedge$-strategies. The number of $\vee$-strategies used by this algorithm is trivially bounded by the number of $\vee$-strategies for $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{\tau}^{(i)})\vee\boldsymbol{\tau}^{(i)}$ which is exponentially in the number of $\vee$-expressions occurring in $\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{\tau}^{(i)}) \vee \boldsymbol{\tau}^{(i)}$. However, we do not know an example for which the algorithm considers more than linearly many strategies.

5 The $\boldsymbol{\nu}$-Method

The $\boldsymbol{\tau}$-method, presented in the previous section, uses strategy iteration over $\vee$-strategies to compute $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{y})$. This could be expensive, as there may be exponentially many $\vee$-strategies. Therefore, we derive an alternative generalization of Newton's method that in each step picks the currently most promising $\vee$-strategy directly, without strategy iteration.

Consider again a fixed feasible min-max-MSP $\boldsymbol{f}$ whose least fixpoint we want to approximate. Assume that $\boldsymbol{y}$ is some approximation of $\mu\boldsymbol{f}$. Instead of applying $\mathcal{N}_{\boldsymbol{f}}$ to $\boldsymbol{y}$, as the $\boldsymbol{\tau}$-method, we now choose a strategy $\sigma \in \Sigma$ such that $\boldsymbol{f}(\boldsymbol{y}) = \boldsymbol{f}^{\sigma}(\boldsymbol{y})$, and compute $\mathcal{N}_{\boldsymbol{f}^{\sigma}}(\boldsymbol{y})$, where $\mathcal{N}_{\boldsymbol{f}^{\sigma}}$ was defined in Section 4 as $\mathcal{N}_{\boldsymbol{f}^{\sigma}}(\boldsymbol{y}) := \mu(\boldsymbol{\mathcal{L}}(\boldsymbol{f}^{\sigma}, \boldsymbol{y})\vee\boldsymbol{y})$. In the following we write $\mathcal{N}_{\sigma}$ instead of $\mathcal{N}_{\boldsymbol{f}^{\sigma}}$ if $\boldsymbol{f}$ is understood.

Assume for a moment that $\boldsymbol{f}$ is a max-MSP and that there is a unique $\sigma \in \Sigma$ such that $\boldsymbol{f}(\boldsymbol{y}) = \boldsymbol{f}^{\sigma}(\boldsymbol{y})$. The approximant $\mathcal{N}_{\sigma}(\boldsymbol{y})$ is the result of applying one iteration of Newton's method, because $\boldsymbol{\mathcal{L}}(\boldsymbol{f}^{\sigma}, \boldsymbol{y})$ is not only a linearization of $\boldsymbol{f}^{\sigma}$, but the first order Taylor approximation of $\boldsymbol{f}$ at $\boldsymbol{y}$. More precisely, $\boldsymbol{\mathcal{L}}(\boldsymbol{f}^{\sigma}, \boldsymbol{y})(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{y}) + \boldsymbol{f}'(\boldsymbol{y})\cdot(\boldsymbol{x}-\boldsymbol{y})$, and $\mathcal{N}_{\sigma}(\boldsymbol{y})$ is obtained by solving $\boldsymbol{x} = \boldsymbol{\mathcal{L}}(\boldsymbol{f}^{\sigma}, \boldsymbol{y})(\boldsymbol{x})$. In this sense, the $\boldsymbol{\nu}$-method is a more direct generalization of Newton's method than the $\boldsymbol{\tau}$-method. Formally, we define the $\boldsymbol{\nu}$-method by a sequence of approximates, the *$\boldsymbol{\nu}$-sequence*.

Definition 2 ($\boldsymbol{\nu}$-sequence). *A sequence $(\boldsymbol{\nu}_{\boldsymbol{f}}^{(k)})_{k\in\mathbb{N}}$ is called* $\boldsymbol{\nu}$-sequence *of a min-max-MSP $\boldsymbol{f}$ if $\boldsymbol{\nu}_{\boldsymbol{f}}^{(0)} = \mathbf{0}$ and for each k there is a strategy $\sigma_{\boldsymbol{f}}^{(k)} \in \Sigma$ with $\boldsymbol{f}(\boldsymbol{\nu}_{\boldsymbol{f}}^{(k)}) = \boldsymbol{f}^{\sigma_{\boldsymbol{f}}^{(k)}}(\boldsymbol{\nu}_{\boldsymbol{f}}^{(k)})$ and $\boldsymbol{\nu}_{\boldsymbol{f}}^{(k+1)} = \mathcal{N}_{\sigma_{\boldsymbol{f}}^{(k)}}(\boldsymbol{\nu}_{\boldsymbol{f}}^{(k)})$. We may drop the subscript if $\boldsymbol{f}$ is understood.*

Notice the nondeterminism here if there is more than one $\vee$-strategy that attains $\boldsymbol{f}(\boldsymbol{\nu}^{(k)})$. The following proposition is analogous to Proposition 1 and states some basic properties of $\boldsymbol{\nu}$-sequences.

Proposition 2. *Let $\boldsymbol{f}$ be a feasible min-max-MSP. The sequence $(\boldsymbol{\nu}^{(k)})$ is monotonically increasing, bounded from above by $\mu\boldsymbol{f}$, and converges to $\mu\boldsymbol{f}$. More precisely, we have $\boldsymbol{\kappa}^{(k)} \leq \boldsymbol{\nu}^{(k)} \leq \boldsymbol{f}(\boldsymbol{\nu}^{(k)}) \leq \boldsymbol{\nu}^{(k+1)} \leq \mu\boldsymbol{f}$ for all $k \in \mathbb{N}$.*

The goal of this section is again to strengthen Proposition 2 towards quantitative convergence results for $\boldsymbol{\nu}$-sequences. To achieve this goal we again relate the convergence of $\boldsymbol{\nu}$-sequences to the convergence of Newton's method for MSPs. If $\boldsymbol{f}$ is an MSP, Lemma 4 allows to argue about the Newton operator $\mathcal{N}_{\boldsymbol{f}}$ when applied to approximates $\boldsymbol{x} \leq \mu\boldsymbol{f}$. To transfer this result to min-max-MSPs $\boldsymbol{f}$ we need an invariant like $\boldsymbol{\nu}^{(k)} \leq \mu\boldsymbol{f}^{\sigma^{(k)}}$ for $\boldsymbol{\nu}$-sequences. As a first step to such an invariant we further restrict the selection of the $\sigma^{(k)}$. Roughly speaking, the strategy in a component i is only changed when it is immediate that component i has not yet reached its fixpoint.

Definition 3 (lazy strategy update). *Let $\boldsymbol{x} \leq \boldsymbol{f}^{\sigma}(\boldsymbol{x})$ for a $\sigma \in \Sigma$. We say that $\sigma' \in \Sigma$ is obtained from $\boldsymbol{x}$ and σ by a* lazy strategy update *if $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}^{\sigma'}(\boldsymbol{x})$ and $\sigma'(f_i) = \sigma(f_i)$ holds for all components i with $f_i(\boldsymbol{x}) = x_i$. We call a $\boldsymbol{\nu}$-sequence $(\boldsymbol{\nu}^{(k)})_{k\in\mathbb{N}}$* lazy *if for all k, the strategy $\sigma^{(k)}$ is obtained from $\boldsymbol{\nu}^{(k)}$ and $\sigma^{(k-1)}$ by a lazy strategy update.*

The key property of lazy $\boldsymbol{\nu}$-sequences is the following non-trivial invariant.

Lemma 7. *Let $(\boldsymbol{\nu}^{(k)})_{k\in\mathbb{N}}$ be a lazy $\boldsymbol{\nu}$-sequence. Then $\boldsymbol{\nu}^{(k)} \leq \mu\boldsymbol{f}^{\sigma^{(k)}\pi}$ holds for all $k \in \mathbb{N}$ and all $\pi \in \Pi^*$.*

The following example shows that lazy strategy updates are essential to Lemma 7 even for max-MSPs.

Example 6. Consider the MSP $\boldsymbol{f}(x,y) = (\frac{1}{2} \vee x, xy + \frac{1}{2})$. Let $\sigma^{(0)}(\frac{1}{2} \vee x) = \frac{1}{2}$ and $\sigma^{(1)}(\frac{1}{2} \vee x) = x$. Then there is a $\boldsymbol{\nu}$-sequence $(\boldsymbol{\nu}^{(k)})$ with $\boldsymbol{\nu}^{(0)} = \mathbf{0}$, $\boldsymbol{\nu}^{(1)} = \mathcal{N}_{\sigma^{(0)}}(\mathbf{0}) = (\frac{1}{2}, 0)$, $\boldsymbol{\nu}^{(2)} = \mathcal{N}_{\sigma^{(1)}}(\boldsymbol{\nu}^{(1)})$. However, the conclusion of Lemma 7 does not hold, because $(\frac{1}{2}, 0) = \boldsymbol{\nu}^{(1)} \not\leq \mu\boldsymbol{f}^{\sigma^{(1)}} = (0, \frac{1}{2})$. Notice that $\sigma^{(1)}$ is not obtained by a lazy strategy update, as $f_1(\boldsymbol{\nu}^{(1)}) = \nu_1^{(1)}$. □

Lemma 7 falls short of our subgoal to establish $\boldsymbol{\nu}^{(k)} \leq \mu\boldsymbol{f}^{\sigma^{(k)}}$, because $\Pi \setminus \Pi^*$ might be non-empty. In fact, we provide an example in [2] showing that $\boldsymbol{\nu}^{(k)} \leq \mu\boldsymbol{f}^{\sigma^{(k)}\pi}$ does not always hold for all $\pi \in \Pi$, even when $\boldsymbol{f}^{\sigma^{(k)}\pi}$ is feasible. Luckily, Lemma 7 will suffice for our convergence speed result.

The left procedure of Algorithm 1 summarizes the *lazy $\boldsymbol{\nu}$-method* which works by computing lazy $\boldsymbol{\nu}$-sequences. The following lemma relates the $\boldsymbol{\nu}$-method for min-max-MSPs to Newton's method for MSPs.

Lemma 8. *Let $\boldsymbol{f}$ be a feasible min-max-MSP and $(\boldsymbol{\nu}^{(k)})$ a lazy $\boldsymbol{\nu}$-sequence. Let m be the number of strategy pairs $(\sigma,\pi) \in \Sigma \times \Pi$ with $\mu\boldsymbol{f} = \mu\boldsymbol{f}^{\sigma\pi}$. Then $m \geq 1$ and there is a constant $k_{as} \in \mathbb{N}$ such that, for all $k \in \mathbb{N}$, there exist strategies $\sigma \in \Sigma$, $\pi \in \Pi$ with $\mu\boldsymbol{f} = \mu\boldsymbol{f}^{\sigma\pi}$ and $\boldsymbol{\nu}_{\boldsymbol{f}}^{(k_{as}+m\cdot k)} \geq \boldsymbol{\tau}_{\boldsymbol{f}^{\sigma\pi}}^{(k)}$.*

Algorithm 1. lazy $\boldsymbol{\nu}$-method

procedure lazy-$\boldsymbol{\nu}(\boldsymbol{f}, k)$
assumes: $\boldsymbol{f}$ is a min-max-MSP
returns: $\boldsymbol{\nu}^{(k)}, \sigma^{(k)}$ obtained by k iterations of the lazy $\boldsymbol{\nu}$-method
  $\boldsymbol{\nu} \leftarrow \mathbf{0}$
  $\sigma \leftarrow$ any $\sigma \in \Sigma$ such that $\boldsymbol{f}(\mathbf{0}) = \boldsymbol{f}^{\sigma}(\mathbf{0})$
  for i **from** 1 **to** k **do**
    $\boldsymbol{\nu} \leftarrow \mathcal{N}_{\boldsymbol{f}^{\sigma}}(\boldsymbol{\nu})$
    $\sigma \leftarrow$ lazy strategy update from $\boldsymbol{\nu}$ and σ
  od
  return $\boldsymbol{\nu}, \sigma$

procedure $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{y})$
assumes: $\boldsymbol{f}$ is a min-MSP, $\boldsymbol{y} \in \mathbb{R}^n_{\geq 0}$
returns: $\mu(\boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{y}) \vee \boldsymbol{y})$
  $\boldsymbol{g} \leftarrow$ linear min-MSP with $\boldsymbol{g}(\boldsymbol{d}) = \boldsymbol{\mathcal{L}}(\boldsymbol{f}, \boldsymbol{y})(\boldsymbol{y} + \boldsymbol{d}) - \boldsymbol{y}$
  $\boldsymbol{u} \leftarrow \boldsymbol{\kappa}^{(n)}_{\boldsymbol{g}}$
  $\widetilde{\boldsymbol{g}} \leftarrow (\widetilde{g}_1, \ldots, \widetilde{g}_n)^{\top}$ where $\widetilde{g}_i = \begin{cases} 0 & \text{if } u_i = 0 \\ g_i & \text{if } u_i > 0 \end{cases}$
  $\boldsymbol{d}^* \leftarrow$ maximize $x_1 + \cdots + x_n$ subject to $\mathbf{0} \leq \boldsymbol{x} \leq \widetilde{\boldsymbol{g}}(\boldsymbol{x})$ by 1 LP
  return $\boldsymbol{y} + \boldsymbol{d}^*$

In typical cases, i.e., if $I - (\boldsymbol{f}^{\sigma\pi})'(\mu\boldsymbol{f})$ is invertible for all $\sigma \in \Sigma$ and $\pi \in \Pi$ with $\mu\boldsymbol{f}^{\sigma\pi} = \mu\boldsymbol{f}$, Newton's method converges exponentially. The following theorem captures the worst-case, in which the lazy $\boldsymbol{\nu}$-method still converges linearly.

Theorem 4. *Let $\boldsymbol{f}$ be a quadratic feasible min-max-MSP. The lazy $\boldsymbol{\nu}$-sequence $(\boldsymbol{\nu}^{(k)})_{k\in\mathbb{N}}$ converges linearly to $\mu\boldsymbol{f}$. More precisely, let m be the number of strategy pairs $(\sigma, \pi) \in \Sigma \times \Pi$ with $\mu\boldsymbol{f} = \mu\boldsymbol{f}^{\sigma\pi}$. Then there is a $k_{\boldsymbol{f}} \in \mathbb{N}$ such that $\boldsymbol{\nu}^{(k_{\boldsymbol{f}} + i \cdot m \cdot (n+1) \cdot 2^n)}$ has at least i valid bits of $\mu\boldsymbol{f}$ for every $i \in \mathbb{N}$.*

Next we show that $\mathcal{N}_{\boldsymbol{f}^{\sigma}}(\boldsymbol{y})$ can be computed exactly by solving a single LP. The right procedure of Algorithm 1 accomplishes this by taking advantage of the following proposition which states that $\mathcal{N}_{\boldsymbol{f}^{\sigma}}(\boldsymbol{y})$ can be determined by computing the least fixpoint of some linear min-MSP $\boldsymbol{g}$.

Proposition 3. *Let $\boldsymbol{y} \leq \boldsymbol{f}^{\sigma}(\boldsymbol{y}) \leq \mu\boldsymbol{f}$. Then $\mathcal{N}_{\boldsymbol{f}^{\sigma}}(\boldsymbol{y}) = \boldsymbol{y} + \mu\boldsymbol{g}$ for the linear min-MSP $\boldsymbol{g}$ with $\boldsymbol{g}(\boldsymbol{d}) = \boldsymbol{\mathcal{L}}(\boldsymbol{f}^{\sigma}, \boldsymbol{y})(\boldsymbol{y} + \boldsymbol{d}) - \boldsymbol{y}$.*

After having computed the linear min-MSP $\boldsymbol{g}$, Algorithm 1 determines the 0-components of $\mu\boldsymbol{g}$. This can be done by performing n Kleene steps, since $(\mu\boldsymbol{g})_i = 0$ whenever $(\boldsymbol{\kappa}^{(n)}_{\boldsymbol{g}})_i = 0$. Let $\widetilde{\boldsymbol{g}}$ be the linear min-MSP obtained from $\boldsymbol{g}$ by substituting the constant 0 for all components g_i with $(\mu\boldsymbol{g})_i = 0$. The least fixpoint of $\widetilde{\boldsymbol{g}}$ can be computed by solving a single LP, as implied by the following lemma. The correctness of Algorithm 1 follows.

Lemma 9. *Let $\boldsymbol{g}$ be a linear min-MSP such that $g_i = 0$ whenever $(\mu\boldsymbol{g})_i = 0$ for all components i. Then $\mu\boldsymbol{g}$ is the greatest vector $\boldsymbol{x}$ with $\boldsymbol{x} \leq \boldsymbol{g}(\boldsymbol{x})$.*

The following theorem is a direct consequence of Lemma 7 for the case where $\Pi = \Pi^*$. It shows the second major advantage of the lazy $\boldsymbol{\nu}$-method, namely, that that the strategies $\sigma^{(k)}$ are meaningful in terms of games.

Theorem 5. *Let $\Pi = \Pi^*$. Let $(\boldsymbol{\nu}^{(k)})_{k\in\mathbb{N}}$ be a lazy $\boldsymbol{\nu}$-sequence. Then $\boldsymbol{\nu}^{(k)} \leq \mu\boldsymbol{f}^{\sigma^{(k)}}$ holds for all $k \in \mathbb{N}$.*

As $(\boldsymbol{\nu}^{(k)})$ converges to $\mu\boldsymbol{f}$, the max-strategy $\sigma^{(k)}$ can be considered ϵ-*optimal*. In terms of games, Theorem 5 states that the strategy $\sigma^{(k)}$ guarantees the max-player an outcome of at least $\boldsymbol{\nu}^{(k)}$. It is open whether an analogous theorem holds for the $\boldsymbol{\tau}$-method.

Application to the primaries example. We solved the equation system of Section 3 approximatively by performing 5 iterations of the lazy $\boldsymbol{\nu}$-method. Using Theorem 5 we found that Clinton can extinguish a problem (a) individual with a probability of at least $X_a = 0.492$ by concentrating on her program and her "ready from day 1" message. (More than 70 Kleene iterations would be needed to infer that X_a is at least 0.49.) As $\boldsymbol{\nu}^{(5)}$ seems to solve above equation system quite well in the sense that $\left\|\boldsymbol{f}(\boldsymbol{\nu}^{(5)}) - \boldsymbol{\nu}^{(5)}\right\|$ is small, we are pretty sure about Obama's optimal strategy: he should talk about Iraq. As $\nu^{(2)}_{X_1} > 0.38$ and $\sigma^{(2)}$ maps f_1 to $0.3 + 0.7X_1^2$, Clinton's team can use Theorem 5 to infer that $X_a \geq 0.38$ by showing emotions and using her "ready from day 1" message.

6 Discussion

In order to compare our two methods in terms of convergence speed, assume that $\boldsymbol{f}$ denotes a feasible min-max-MSP. Since $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x}) \geq \mathcal{N}_{\boldsymbol{f}^\sigma}$ (Lemma 3.5), it follows that $\boldsymbol{\tau}^{(i)}_{\boldsymbol{f}} \geq \boldsymbol{\nu}^{(i)}_{\boldsymbol{f}}$ holds for all $i \in \mathbb{N}$. This means that the $\boldsymbol{\tau}$-method is as least as fast as the $\boldsymbol{\nu}$-method if one counts the number of approximation steps. Next, we construct an example which shows that the number of approximation steps needed by the lazy $\boldsymbol{\nu}$-method can be much larger than the respective number needed by the $\boldsymbol{\tau}$-method. It is parameterized with an arbitrary $k \in \mathbb{N}$ and given by $\boldsymbol{f}(x_1, x_2) = \left(x_2 \wedge 2,\ \ x_1^2 + 0.25 \vee x_1 + 2^{-2(k+1)}\right)^\top$. Since the constant $2^{-2(k+1)}$ is represented using $\mathcal{O}(k)$ bits, it is of size linear in k. It can be shown (see [2]) that the lazy $\boldsymbol{\nu}$-method needs at least k steps. More precisely, $\nu\boldsymbol{f} - \boldsymbol{\tau}^{(k)} \geq (1.5, 1.95)$. The $\boldsymbol{\tau}$-method needs exactly 2 steps.

We now compare our approaches with the tool PReMo [14]. PReMo employs 4 different techniques to approximate $\mu\boldsymbol{f}$ for min-max-MSPs $\boldsymbol{f}$: It uses Newton's method only for MSPs without min or max. In this case both of our methods coincide with Newton's method. For min-max-MSPs, PReMo uses Kleene iteration, round-robin iteration (called Gauss-Seidel in [14]), and an "optimistic" variant of Kleene which is not guaranteed to converge. In the following we compare our algorithms only with Kleene iteration, as our algorithms are guaranteed to converge and a round-robin step is not faster than n Kleene steps.

Our methods improve on Kleene iteration in the sense that $\boldsymbol{\kappa}^{(i)} \leq \boldsymbol{\tau}^{(i)}, \boldsymbol{\nu}^{(i)}$ holds for all $i \in \mathbb{N}$, and our methods converge linearly, whereas Kleene iteration does not converge linearly in general. For example, consider the MSP $g(x) = \frac{1}{2}x^2 + \frac{1}{2}$ with $\mu g = 1$. Kleene iteration needs exponentially many iterations for j bits [4], whereas Newton's method gives exactly 1 bit per iteration. For the slightly modified MSP $\tilde{g}(x) = g(x) \wedge 1$ which has the same fixpoint, PReMo no longer uses Newton's method, as $\tilde{g}$ contains a minimum. Our algorithms still produce exactly 1 bit per iteration.

In the case of linear min-max systems our methods compute the precise solution and not only an approximation. This applies, for example, to the max-linear system of [14] describing the expected time of termination of a nondeterministic variant of Quicksort.

Notice that Kleene iteration does not compute the precise solution (except for trivial instances), even for linear MSPs without min or max.

We implemented our algorithms prototypically in Maple and ran them on the quadratic nonlinear min-max-MSP describing the termination probabilities of a recursive simple stochastic game. This game stems from the example suite of PReMo (`rssg2.c`) and we used PReMo to produce the equations. Both of our algorithms reached the least fixpoint after 2 iterations. So we could compute the precise $\mu \boldsymbol{f}$ and optimal strategies for both players, whereas PReMo computes only approximations of $\mu \boldsymbol{f}$.

7 Conclusion

We have presented the first methods for approximatively computing the least fixpoint of min-max-MSPs, which are guaranteed to converge at least linearly. Both of them are generalizations of Newton's method. Whereas the $\boldsymbol{\tau}$-method converges faster in terms of number of approximation steps, one approximation step of the $\boldsymbol{\nu}$-method is cheaper. Furthermore, we have shown that the $\boldsymbol{\nu}$-method computes ϵ-optimal strategies for games. Whether such a result can also be established for the $\boldsymbol{\tau}$-method is still open. A direction for future research is to evaluate our methods in practice. In particular, the influence of imprecise computation through floating point arithmetic should be studied. It would also be desirable to find a bound on the "threshold" $k_{\boldsymbol{f}}$.

Acknowledgement. We thank the anonymous referees for valuable comments.

References

1. Condon, A.: The complexity of stochastic games. Inf. and Comp. 96(2), 203–224 (1992)
2. Esparza, J., Gawlitza, T., Kiefer, S., Seidl, H.: Approximative methods for monotone systems of min-max-polynomial equations. Technical report, Technische Universität München, Institut für Informatik (February 2008)
3. Esparza, J., Kiefer, S., Luttenberger, M.: Convergence thresholds of Newton's method for monotone polynomial equations. In: Proceedings of STACS, pp. 289–300 (2008)
4. Etessami, K., Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 340–352. Springer, Heidelberg (2005)
5. Etessami, K., Yannakakis, M.: Recursive Markov decision processes and recursive stochastic games. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580. Springer, Heidelberg (2005)
6. Etessami, K., Yannakakis, M.: Efficient qualitative analysis of classes of recursive Markov decision processes and simple stochastic games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 634–645. Springer, Heidelberg (2006)
7. Filar, J., Vrieze, K.: Competitive Markov Decision processes. Springer, Heidelberg (1997)
8. Gawlitza, T., Seidl, H.: Precise fixpoint computation through strategy iteration. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 300–315. Springer, Heidelberg (2007)
9. Gawlitza, T., Seidl, H.: Precise relational invariants through strategy iteration. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 23–40. Springer, Heidelberg (2007)
10. Harris, T.E.: The Theory of Branching Processes. Springer, Heidelberg (1963)

11. Kiefer, S., Luttenberger, M., Esparza, J.: On the convergence of Newton's method for monotone systems of polynomial equations. In: STOC, pp. 217–226. ACM, New York (2007)
12. Neyman, A., Sorin, S.: Stochastic Games and Applications. Kluwer Academic Press, Dordrecht (2003)
13. Ortega, J., Rheinboldt, W.: Iterative solution of nonlinear equations in several variables. Academic Press, London (1970)
14. Wojtczak, D., Etessami, K.: PReMo: An analyzer for probabilistic recursive models. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 66–71. Springer, Heidelberg (2007)

Recursive Stochastic Games with Positive Rewards

K. Etessami[1], D. Wojtczak[1], and M. Yannakakis[2]

[1] LFCS, School of Informatics, University of Edinburgh
[2] Dept. of Computer Science, Columbia University

Abstract. We study the complexity of a class of Markov decision processes and, more generally, stochastic games, called 1-exit Recursive Markov Decision Processes (1-RMDPs) and Simple Stochastic Games (1-RSSGs) with strictly positive rewards. These are a class of finitely presented countable-state zero-sum stochastic games, with total expected reward objective. They subsume standard finite-state MDPs and Condon's simple stochastic games and correspond to optimization and game versions of several classic stochastic models, with rewards. Such stochastic models arise naturally as models of probabilistic procedural programs with recursion, and the problems we address are motivated by the goal of analyzing the optimal/pessimal expected running time in such a setting.

We give polynomial time algorithms for 1-exit Recursive Markov decision processes (1-RMDPs) with positive rewards. Specifically, we show that the exact optimal value of both maximizing and minimizing 1-RMDPs with positive rewards can be computed in polynomial time (this value may be ∞). For two-player 1-RSSGs with positive rewards, we prove a "stackless and memoryless" determinacy result, and show that deciding whether the game value is at least a given value r is in NP $\cap$ coNP. We also prove that a simultaneous strategy improvement algorithm converges to the value and optimal strategies for these stochastic games. We observe that 1-RSSG positive reward games are "harder" than finite-state SSGs in several senses.

1 Introduction

Markov decision processes and stochastic games are fundamental models in stochastic optimization and game theory (see, e.g., [25,23,13]). In this paper, motivated by the goal of analyzing the optimal/pessimal expected running time of probabilistic procedural programs, we study the complexity of a reward-based stochastic game, called *1-exit recursive simple stochastic games* (1-RSSGs), and its 1-player version, *1-exit recursive Markov decision processes* (1-RMDPs). These form a class of (finitely presented) countable-state turn-based zero-sum stochastic games (and MDPs) with strictly positive rewards, and with an undiscounted expected total reward objective.

Intuitively, a 1-RSSG (1-RMDP) consists of a collection of finite-state component SSGs (MDPs), each of which can be viewed as an abstract finite-state

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 711–723, 2008.

procedure (subroutine) of a probabilistic program with potential recursion. Each component procedure has some nodes that are probabilistic and others that are controlled by one or the other of the two players. The component SSGs can call each other in a recursive manner, generating a potentially unbounded call stack, and thereby an infinite state space. The "1-exit" restriction essentially restricts these finite-state subroutines so they do not return a value, unlike multi-exit RSSGs and RMDPs in which they can return distinct values. (We shall show that the multi-exit version of these reward games are undecidable.) 1-RMDPs and 1-RSSGs were studied in [8,9] in a setting without rewards, where the goal of the players was to maximize/minimize the probability of termination. Such termination probabilities can be irrational, and quantitative decision problems for them subsume long standing open problems in exact numerical computation. Here we extend 1-RSSGs and 1-RMDPs to a setting with positive rewards. Note that much of the literature on MDPs and games is based on a reward structure. This paper is a first step toward extending these models to the recursive setting. Interestingly, we show that the associated problems actually become more benign in some respects in this strictly positive reward setting. In particular, the values of our games are either rational, with polynomial bit complexity, or ∞.

The 1-RMDP and 1-RSSG models can also be described as optimization and game versions of several classic stochastic models, including stochastic context-free grammars (SCFGs) and (multi-type) branching processes. These have applications in many areas, including natural language processing [21], biological sequence analysis ([4]), and population biology [17,16]. Another model that corresponds to a strict subclass of SCFGs is "random walks with back-buttons" studied in [12] as a model of web surfing. See [7] for details on the relationships between these various models. A 1-RSSG with positive rewards, can be equivalently reformulated as the following game played on a stochastic context-free grammar (see full version [11] for details). We are given a context-free grammar where non-terminals are partitioned into three disjoint sets: `random`, `player-1`, and `player-2`. Starting from a designated start non-terminal, S_{init}, we proceed to generate a derivation by choosing a remaining *left-most* non-terminal, S, and expanding it. The precise derivation law (left-most, right-most, etc.) doesn't effect the game value in our strictly positive reward setting, but does if we allow 0 rewards. If S belongs to `random`, it is expanded randomly by choosing a rule $S \to \alpha$, according to a given probability distribution over the rules whose left hand side is S. If S belongs to `player-i`, then player i chooses which grammar rule to use to expand this S. Each grammar rule also has an associated (strictly positive) *reward* for player 1, and each time a rule is used during the derivation, player 1 accumulates this associated reward. Player 1 wants to maximize total expected reward (which may be ∞), and player 2 wants to minimize it. When we have only one player it is a minimizing or maximizing 1-RMDP.

We assume strictly positive rewards on all transitions (rules) in this paper. This assumption is very natural for modeling optimal/pessimal expected running time in probabilistic procedural programs: each discrete step of the program is assumed to cost some non-zero amount of time. Strictly positive rewards also

endow our games with a number of important robustness properties. In particular, in the above grammar presentation, with strictly positive rewards these games have the same value regardless of what derivation law is imposed. This is not the case if we also allow 0 rewards on grammar rules. In that case, even in the single-player setting, the game value can be wildly different (e.g., 0 or ∞) depending on the derivation law (e.g., left-most or right-most). Moreover, for 1-RMDPs, if we allow 0 rewards, then there may not even exist any ϵ-optimal strategies. Furthermore, even in a purely probabilistic setting without players (1-RMCs), with 0 rewards the expected reward can be irrational. Even the decidability of determining whether the supremum expected reward for 1-RMDPs is greater than a given rational value is open, and subsumes other open decidability questions, e.g., for optimal reachability probabilities in non-reward 1-RMDPs ([8,1]). (See the full version [11] for elaboration on these issues.) As we shall show, none of these pathologies arise in our setting with strictly positive rewards.

We show that 1-RMDPs and 1-RSSGs with strictly positive rewards have a value which is either rational (with polynomial bit complexity) or ∞, and which arises as the least fixed point solution (over the extended reals) of an associated system of linear-min-max equations. Both players do have optimal strategies in these games, and in fact we show the much stronger fact that both players have *stackless and memoryless* (SM) optimal strategies: deterministic strategies that depend only on the current state of the running component, and not on the history or even the stack of pending recursive calls.

We provide polynomial-time algorithms for computing the exact value for both the maximizing and minimizing 1-RMDPs. The two cases are not equivalent and require separate treatment. We show that for the 2-player games (1-RSSGs) deciding whether the game has value at least a given $r \in \mathbb{Q} \cup \{\infty\}$ is in NP $\cap$ coNP. We also describe a practical simultaneous strategy improvement algorithm, analogous to similar algorithms for finite-state stochastic games, and show that it converges to the game value (even if it is ∞) in a finite number of steps. A corollary is that computing the game value and optimal strategies for these games is contained in the class PLS of polynomial local search problems ([19]). Whether this strategy improvement algorithm runs in worst-case P-time is open, just like its version for finite-state SSGs.

We observe that these games are essentially "harder" than Condon's finite-state SSG games in the following senses. We reduce Condon's quantitative decision problem for finite-state SSGs to a special case of 1-RSSG games with strictly positive rewards: namely to deciding whether the game value is ∞. By contrast, if finite-state SSGs are themselves equipped with strictly positive rewards, we can decide in P-time whether their value is ∞. Moreover, it has recently been shown that computing the value of Condon's SSG games is in the complexity class PPAD (see [10] and [20]). The same proof however does not work for 1-RSSG positive reward games, and we do not know whether these games are contained in PPAD. Technically, the problem is that in the expected reward setting the domain of the fixed point equations is not compact, and indeed the expected

reward is potentially ∞. In these senses, the 1-RSSG reward games studied in this paper appear to be "harder" than Condon's SSGs, and yet as we show their quantitative decision problems remain in NP $\cap$ coNP. Finally, we show that the more general multi-exit RSSG model is undecidable. Namely, even for single-player multi-exit RMDPs with strictly positive rewards, it is undecidable whether the optimal reward value is ∞.

The tool PReMo [28] implements a number of analyses for RMCs, 1-RMDPs, and 1-RSSGs. Most recently, the strategy improvement algorithm of this paper was implemented and incorporated in the tool. See the PReMo web page ([28]) for very encouraging experimental results based on the algorithms of this paper. *Due to space constraints proofs and discussions are omitted. See full paper [11].*

Related work. Two (equivalent) purely probabilistic recursive models, Recursive Markov chains and probabilistic Pushdown Systems (pPDSs) were introduced in [7] and [5], and have been studied in several papers recently. These models were extended to the optimization and game setting of (1)-RMDPs and (1)-RSSGs in [8,9], and studied further in [1]. As mentioned earlier, the games considered in these earlier papers had the goal of maximizing/minimizing termination or reachability probability, which can be irrational, and for which quantitative decision problems encounter long standing open problems in numerical computation, even to place their complexity in NP. On the other hand, the qualitative termination decision problem ("is the termination game value exactly 1?") for 1-RMDPs was shown to be in P, and for 1-RSSGs in NP $\cap$ coNP in [9]. These results are related to the results in the present paper as follows. If termination occurs with probability strictly less than 1 in a strictly positive reward game, then the expected total reward is ∞. But the converse does not hold: the expected reward may be ∞ even when the game terminates with probability 1, because there can be *null recurrence* in these infinite-state games. Thus, not only do we have to address this discrepancy, but also our goal in this paper is quantitative computation (compute the optimal reward), whereas in [9] it was purely qualitative (almost sure termination).

Condon [2] originally studied finite-state SSGs with termination objectives (no rewards), and showed that the quantitative termination decision problem is in NP $\cap$ coNP; it is a well-known open problem whether it is in P. In [3] strategy improvement algorithms for SSGs were studied, based on variants of the classic Hoffman-Karp algorithm [18]. It remains open whether the simultaneous version of strategy improvement runs in P-time. This is also the case for our simultaneous strategy improvement algorithm for 1-RSSGs with positive rewards. (Single-vertex updates per step in strategy improvement is known to require exponentially many steps in the worst case.)

There has been some recent work on augmenting purely probabilistic multi-exit RMCs and pPDSs with rewards in [6]. These results however are for RMCs without players. We in fact show in Theorem 8 that the basic questions about multi-exit RMDPs and RSSGs are undecidable.

A full tech report of this paper appeared in [11]. A recent and independent paper by Gawlitza and Seidl [14] considers monotone linear-min-max equations

with potentially negative constant terms (with entirely different motivation from abstract interpretation), and studies a different kind of strategy improvement algorithm for computing their least fixed point solution over the *full* extended reals. Their work is related to ours, but in rather subtle ways. In particular their notion of LFP over the extended reals may yield negative values or even $-\infty$, and they assume that "strategies" (choices for the max and min operators) are memoryless, rather than proving a (memoryless) determinacy result. Moreover, their strategy improvement algorithm requires a particular initial strategy (otherwise it can fail) and thus is not directly formulable as a local search. Unlike our results, their results apparently do not yield [15] containment in NP∩coNP, nor in PLS, for the relevant decision and search problems. Nevertheless, there are connections between their work and ours that need to be explored further. In particular, Gawlitza [15] informs us that a modified version of their strategy improvement algorithm can also be used to obtain our P-time upper bound for the LFP, over the non-negative extended reals, for the linear-min and linear-max equations that arise for 1-RMDPs.

Models related to 1-RMDPs have been studied in OR, under the name Branching Markov Decision Chains (a controlled version of multi-type Branching processes). These are close to the single-player SCFG model, with non-negative rewards, but simultaneous derivation law. They were studied by Pliska [24], in a related form by Veinott [27], and extensively by Rothblum and co-authors (e.g., [26]). Besides the restriction to simultaneous derivation, these models were restricted to the single-player MDP case, and to simplify their analysis they were typically assumed to be "transient" (i.e., the expected number of visits to a node was assumed to be finite under all strategies). None of these works yield a P-time algorithm for optimal expected rewards for 1-RMDPs with positive rewards.

2 Definitions and Background

Let $\mathbb{R}_{>0} = (0, \infty)$ denote the positive real numbers, $\mathbb{R}_{\geq 0} \doteq [0, \infty)$, $\overline{\mathbb{R}} \doteq [-\infty, \infty]$, $\mathbb{R}^{\infty}_{>0} \doteq (0, \infty]$, and $\mathbb{R}^{\infty}_{\geq 0} \doteq [0, \infty]$. The extended reals $\overline{\mathbb{R}}$ have the natural total order. We assume the following usual arithmetic conventions on the non-negative extended reals $\mathbb{R}^{\infty}_{\geq 0}$: $a \cdot \infty = \infty$, for any $a \in \mathbb{R}^{\infty}_{>0}$; $0 \cdot \infty = 0$; $a + \infty = \infty$, for any $a \in \mathbb{R}^{\infty}_{\geq 0}$. This extends naturally to matrix arithmetic over $\mathbb{R}^{\infty}_{\geq 0}$. We first define general multi-exit RSSGs (for which basic reward problems turn out to be undecidable). Later, we will confine these to the 1-exit case, 1-RSSGs.

A *Recursive Simple Stochastic Game (RSSG) with positive rewards* is a tuple $A = (A_1, \ldots, A_k)$, where each *component* $A_i = (N_i, B_i, Y_i, En_i, Ex_i, \mathtt{pl}_i, \delta_i, \xi_i)$ consists of:

- A set N_i of *nodes*, with a distinguished subset En_i of *entry* nodes and a (disjoint) subset Ex_i of *exit* nodes.
- A set B_i of *boxes*, and a mapping $Y_i : B_i \mapsto \{1, \ldots, k\}$ that assigns to every box (the index of) a component. To each box $b \in B_i$, we associate a set of *call ports*, $Call_b = \{(b, en) \mid en \in En_{Y(b)}\}$, and a set of *return ports*, $Ret_b = \{(b, ex) \mid ex \in Ex_{Y(b)}\}$. Let $Call^i = \cup_{b \in B_i} Call_b$, $Ret^i = \cup_{b \in B_i} Ret_b$,

and let $Q_i = N_i \cup Call^i \cup Ret^i$ be the set of all nodes, call ports and return ports; we refer to these as the *vertices* of component A_i.

- A mapping $\mathtt{pl}_i : Q_i \mapsto \{0, 1, 2\}$ that assigns to every vertex a *player* (Player 0 represents "chance" or "nature"). We assume $\mathtt{pl}_i(ex) = 0$ for all $ex \in Ex_i$.
- A transition relation $\delta_i \subseteq (Q_i \times (\mathbb{R}_{>0} \cup \{\bot\}) \times Q_i \times \mathbb{R}_{>0})$, where for each tuple $(u, x, v, c_{u,v}) \in \delta_i$, the source $u \in (N_i \setminus Ex_i) \cup Ret^i$, the destination $v \in (N_i \setminus En_i) \cup Call^i$, and x is either (i) $p_{u,v} \in (0, 1]$ (the transition probability) if $\mathtt{pl}_i(u) = 0$, or (ii) $x = \bot$ if $\mathtt{pl}_i(u) = 1$ or 2; and $c_{u,v} \in \mathbb{R}_{>0}$ is the positive reward associated with this transition. We assume for vertices u and v there is at most one transition in δ from u to v. For computational purposes we assume the given probabilities $p_{u,v}$ and rewards $c_{u,v}$ are rational. Probabilities must also satisfy consistency: for every $u \in \mathtt{pl}_i^{-1}(0)$, $\sum_{\{v' | (u,p_{u,v'},v',c_{u,v}) \in \delta_i\}} p_{u,v'} = 1$, unless u is a call port or exit node, neither of which have outgoing transitions, in which case by default $\sum_{v'} p_{u,v'} = 0$.
- Finally, the mapping $\xi_i : Call_i \mapsto \mathbb{R}_{>0}$ maps each call port u in the component to a positive rational value $c_u = \xi(u)$. (This mapping reflects the "cost" of a function call, but is not strictly necessary. This cost can be 0 and all our results would still hold.)

We use the symbols $(N, B, Q, \delta$, etc.) without a subscript, to denote the union over all components. Thus, e.g., $N = \cup_{i=1}^{k} N_i$ is the set of all nodes of A, $\delta = \cup_{i=1}^{k} \delta_i$ the set of all transitions, etc. Let $n(u) = \{v \mid (u, \bot, v, c_{u,v}) \in \delta\}$ denote the neighbors of u if u is a player 1 or player 2 node and $n(u) = \{v \mid (u, p_{u,v}, v, c_{u,v}) \in \delta\}$ otherwise. An RSSG A defines a global denumerable simple stochastic game, with rewards, $M_A = (V = V_0 \cup V_1 \cup V_2, \Delta, \mathtt{pl})$ as follows. The global *states* $V \subseteq B^* \times Q$ of M_A are pairs of the form $\langle \beta, u \rangle$, where $\beta \in B^*$ is a (possibly empty) sequence of boxes and $u \in Q$ is a *vertex* of A. The states $V \subseteq B^* \times Q$ and transitions Δ are defined inductively as follows:

1. $\langle \epsilon, u \rangle \in V$, for $u \in Q$. (ϵ denotes the empty string.)
2. if $\langle \beta, u \rangle \in V$ & $(u, x, v, c) \in \delta$, then $\langle \beta, v \rangle \in V$ and $(\langle \beta, u \rangle, x, \langle \beta, v \rangle, c) \in \Delta$.
3. if $\langle \beta, (b, en) \rangle \in V$ & $(b, en) \in Call_b$, then $\langle \beta b, en \rangle \in V$ & $(\langle \beta, (b, en) \rangle, 1, \langle \beta b, en \rangle, \xi((b, en))) \in \Delta$.
4. if $\langle \beta b, ex \rangle \in V$ & $(b, ex) \in Ret_b$, then $\langle \beta, (b, ex) \rangle \in V$ & $(\langle \beta b, ex \rangle, 1, \langle \beta, (b, ex) \rangle, 0) \in \Delta$.

The mapping $\mathtt{pl} : V \mapsto \{0, 1, 2\}$ is given as follows: $\mathtt{pl}(\langle \beta, u \rangle) = \mathtt{pl}(u)$ if u is in $Q \setminus (Call \cup Ex)$, and $\mathtt{pl}(\langle \beta, u \rangle) = 0$ if $u \in Call \cup Ex$. The set of states V is partitioned into V_0, V_1, and V_2, where $V_i = \mathtt{pl}^{-1}(i)$. We consider M_A with various *initial states* of the form $\langle \epsilon, u \rangle$, denoting this by M_A^u. Some states of M_A are *terminating states* and have no outgoing transitions. These are states $\langle \epsilon, ex \rangle$, where ex is an exit node. An RSSG where $V_2 = \emptyset$ ($V_1 = \emptyset$) is called a maximizing (minimizing, respectively) *Recursive Markov Decision Process* (RMDP); an RSSG where $V_1 \cup V_2 = \emptyset$ is called a *Recursive Markov Chain* (RMC) ([7]); A *1-RSSGs* is a RSSG where every component has one exit, and we likewise define *1-RMDPs* and *1-RMCs*. This entire paper is focused on 1-RSSGs and 1-RMDPs, except for Theorem 8, where we show that multi-exit RMDP reward games are undecidable. In a (1-)RSSG with positive rewards the goal of player

1 (maximizer) is to maximize the total expected reward gained during a play of the game, and the goal of player 2 (minimizer) is to minimize this. A *strategy* σ for player i, $i \in \{1,2\}$, is a function $\sigma : V^*V_i \mapsto V$, where, given the history $ws \in V^*V_i$ of play so far, with $s \in V_i$ (i.e., it is player i's turn to play a move), $\sigma(ws) = s'$ determines the next move of player i, where $(s, \bot, s', c) \in \Delta$. (We could also allow randomized strategies, but this won't be necessary, as we shall see.) Let Ψ_i denote the set of all strategies for player i. A pair of strategies $\sigma \in \Psi_1$ and $\tau \in \Psi_2$ induce in a straightforward way a Markov chain $M_A^{\sigma,\tau} = (V^*, \Delta')$, whose set of states is the set V^* of histories. Let $r_u^{k,\sigma,\tau}$ denote the expected reward in k steps in $M_A^{\sigma,\tau}$, starting at initial state $\langle \epsilon, u \rangle$. Formally, we can define the total expected reward gained during the i'th transition, starting at $\langle \epsilon, u \rangle$ to be given by a random variable Y_i. The total k-step expected reward is simply $r_u^{k,\sigma,\tau} = E[\sum_{i=1}^{k} Y_i]$. When $k = 0$, we of course have $r_u^{0,\sigma,\tau} = 0$. Given an initial vertex u, let $r_u^{*,\sigma,\tau} = \lim_{k\to\infty} r^{k,\sigma,\tau} = E[\sum_{i=1}^{\infty} Y_i] \in [0,\infty]$ denote the total expected reward obtained in a run of $M_A^{\sigma,\tau}$, starting at initial state $\langle \epsilon, u \rangle$. Clearly, this sum may diverge, thus $r^{*,\sigma,\tau} \in [0,\infty]$. Note that, because of the positive constraint on the rewards out of all transitions, the sum will be finite if and only if the expected number of steps until the run terminates is finite.

We now want to associate a "value" to 1-RSSG games. Unlike 1-RSSGs with termination probability objectives, it unfortunately does not follow directly from general determinacy results such as Martin's Blackwell determinacy ([22]) that these games are determined, because those determinacy results require a Borel payoff function to be bounded, whereas the payoff function for us is unbounded. Nevertheless, we will establish that determinacy does hold for 1-RSSG positive reward games, as part of our proof of Stackless & Memoryless determinacy. For all vertices u, let $r_u^* \doteq \sup_{\sigma\in\Psi_1} \inf_{\tau\in\Psi_2} r_u^{*,\sigma,\tau}$. We show $r_u^* = \inf_{\tau\in\Psi_2} \sup_{\sigma\in\Psi_1} r_u^{*,\sigma,\tau}$, and thus r_u^* is the *value* of the game starting at vertex u. We are interested in the following problem: *Given A, a 1-RSSG (or 1-RMDP), and given a vertex u in A, compute r_u^* if it is finite, or else declare that $r_u^* = \infty$. Also, compute optimal strategies for both players.* For a strategy $\sigma \in \Psi_1$, let $r_u^{*,\sigma} = \inf_{\tau\in\Psi_2} r_u^{*,\sigma,\tau}$, and for $\tau \in \Psi_2$, let $r_u^{*,\cdot,\tau} = \sup_{\sigma\in\Psi_1} r_u^{*,\sigma,\tau}$. Call a deterministic strategy *Stackless & Memoryless (SM)* if it depends neither on the history of the game nor on the current call stack, i.e., only depends on the current vertex. Such strategies, for player i, can be given by a map $\sigma : V_i \mapsto V$. We call a game *SM-determined* if both players have optimal SM strategies.

In ([8]) we defined a monotone system of *nonlinear* min-max equations for the value of the termination probability game on 1-RSSGs, and showed that its *Least Fixed Point* solution yields the desired probabilities. Here we show we can adapt this to obtain analogous *linear* min-max systems in the setting of positive reward 1-RSSGs. We use a variable x_u for each unknown r_u^*. Let $\mathbf{x}$ be the vector of all $x_u, u \in Q$. The system has one equation of the form $x_u = P_u(\mathbf{x})$ for each vertex u. Suppose that u is in component A_i with (unique) exit ex. There are 5 cases based on the "*Type*" of u.

1. $Type_0$: $u = ex$. In this case: $x_u = 0$.
2. $Type_{rand}$: $\mathtt{pl}(u) = 0$ & $u \in (N_i \setminus \{ex\}) \cup Ret^i$: $x_u = \sum_{v\in n(u)} p_{u,v}(x_v + c_{u,v})$.

3. $Type_{call}$: $u = (b, en)$ is a call port: $x_{(b,en)} = x_{en} + x_{(b,ex')} + c_u$, where $ex' \in Ex_{Y(b)}$ is the unique exit of $A_{Y(b)}$.
4. $Type_{max}$: $\mathtt{pl}(u) = 1$ and $u \in (N_i \setminus \{ex\}) \cup Ret^i$: $x_u = \max_{v \in n(u)}(x_v + c_{u,v})$
5. $Type_{min}$: $\mathtt{pl}(u) = 2$ and $u \in (N_i \setminus \{ex\}) \cup Ret^i$: $x_u = \min_{v \in n(u)}(x_v + c_{u,v})$

We denote the system in vector form by $\mathbf{x} = P(\mathbf{x})$. Given a 1-RSSG, we can easily construct its associated system in linear time. For vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$ means $x_j \leq y_j$ for every j. Let $\mathbf{r}^* \in \mathbb{R}^n$ denote the n-vector of r_u^*'s. Let $\mathbf{0}$ denote an all 0 vector, and define $\mathbf{x}^0 = \mathbf{0}$, $\mathbf{x}^{k+1} = P^{k+1}(\mathbf{0}) = P(\mathbf{x}^k)$, for $k \geq 0$.

Theorem 1. *(1) The map $P : \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^n$ is monotone on $\mathbb{R}^{\infty}_{\geq 0}$ and $\mathbf{0} \leq \mathbf{x}^k \leq \mathbf{x}^{k+1}$ for $k \geq 0$. (2) $\mathbf{r}^* = P(\mathbf{r}^*)$. (3) For all $k \geq 0$, $\mathbf{x}^k \leq \mathbf{r}^*$. (4) For all $\mathbf{r}' \in \mathbb{R}^{\infty}_{\geq 0}$, if $\mathbf{r}' = P(\mathbf{r}')$, then $\mathbf{r}^* \leq \mathbf{r}'$. (5) For all vertices u, $r_u^* \doteq \sup_{\sigma \in \Psi_1} \inf_{\tau \in \Psi_2} r_u^{*,\sigma,\tau} = \inf_{\tau \in \Psi_2} \sup_{\sigma \in \Psi_1} r_u^{*,\sigma,\tau}$ (i.e., these games are determined). (6) $\mathbf{r}^* = \lim_{k \to \infty} \mathbf{x}^k$.*

The following is a simple corollary of the proof.

Corollary 1. *In 1-RSSG positive reward games, the* minimizer *has an optimal deterministic Stackless and Memoryless (SM) strategy.*

3 SM-Determinacy and Strategy Improvement

We now prove SM-determinacy, and also show that strategy improvement can be used to compute the values and optimal strategies for 1-RSSG positive reward games. Consider the following *(simultaneous) strategy improvement* algorithm.

Initialization: Pick some (any) SM strategy, σ, for player 1 (maximizer).
Iteration step: First compute the optimal value, $r_u^{*,\sigma}$, starting from every vertex, u, in the resulting minimizing 1-RMDP. (We show in Theorem 3 that this can be done in P-time.) Then, update σ to a new SM strategy, σ', as follows. For each vertex $u \in Type_{max}$, if $\sigma(u) = v$ and u has a neighbor $w \neq v$, such that $r_w^{*,\sigma} + c_{u,w} > r_v^{*,\sigma} + c_{u,v}$, let $\sigma'(u) := w$ (e.g., choose a w that maximizes $r_w^{*,\sigma} + c_{u,w}$). Otherwise, let $\sigma'(u) := \sigma(u)$.

Repeat the iteration step, using the new σ' in place of σ, until no further local improvement is possible, i.e., stop when $\sigma' = \sigma$.

Theorem 2 shows that this algorithm always halts, and produces an optimal final SM strategy for player 1. (The proof shows it works even if we switch any non-empty subset of improvable vertices in each iteration.) Combined with Corollary 1, both players have optimal SM strategies, i.e., the games are SM-determined.

Theorem 2. *(1)* SM-determinacy. *In 1-RSSG positive reward games, both players have optimal SM strategies. (2)* Strategy Improvement. *Moreover, we can compute the value and optimal SM strategies using the above simultaneous strategy improvement algorithm. (3) Computing the value and optimal strategies in these games is contained in the class PLS.*

The proof is intricate, and is given in the full version ([11]). Here we briefly sketch the approach. Fix a SM strategy σ for player 1. It can be shown that if $\mathbf{x} = P(\mathbf{x})$ is the linear-min-max equation system for this 1-RSSG, then $\mathbf{r}^{*,\sigma}_u \leq P_u(\mathbf{r}^{*,\sigma})$, for all vertices u, and equality fails only on vertices u_i belonging to player 1 such that $\sigma(u_i) = v_i$ is not "locally optimal", i.e., such that there exists some neighbor w_i such that $r^{*,\sigma}_{w_i} + c_{u_i,w_i} > r^{*,\sigma}_{v_i} + c_{u_i,v_i}$. Let $u_1, \ldots, u_n$ be all such vertices belonging to player 1. Associate a parameter $t_i \in \mathbb{R}^{\infty}_{\geq 0}$ with each such vertex u_i, creating a parametrized game $A(\mathbf{t})$, in which whenever the vertex u_i is encountered player 1 gains additional reward t_i and the game then terminates. Let $g_{u,\tau}(\mathbf{t})$ denote the expected reward of this parametrized game starting at vertex u, when player 1 uses SM strategy σ and player 2 uses SM strategy τ. Let $f_u(\mathbf{t}) = \min_\tau g_{u,\tau}(\mathbf{t})$. The vector $\mathbf{t}^\sigma$, where $t^\sigma_i = r^{*,\sigma}_{u_i}$, is a fixed point of $f_u(\mathbf{t})$, for every vertex u, and so is $\mathbf{t}^{\sigma'}$ where σ' is any SM strategy consistent with σ on all vertices other than the u_i's. The functions $g_{u,\tau}(\mathbf{t})$ is continuous and nondecreasing over $[0,\infty]^n$, and expressible as an infinite sum of *linear* terms with non-negative coefficients. Using these properties of $g_{u,\tau}$, and their implications for f_u, we show that if σ' is the SM strategy obtained by locally improving the strategy σ at the u_i's, by letting $\sigma'(u_i) := w_i$, then $t^\sigma_i = \mathbf{r}^{*,\sigma}_{u_i} < \mathbf{r}^{*,\sigma'}_{u_i} = \mathbf{t}^{\sigma'}_i$, and thus also $\mathbf{r}^{*,\sigma}_z = f_z(\mathbf{t}^\sigma) \leq f_z(\mathbf{t}^{\sigma'}) = \mathbf{r}^{*,\sigma'}_z$, for any vertex z. Thus, switching to σ' does not decrease the value at any vertex, and increases it on all the switched vertices u_i. There are only finitely many SM strategies, thus after finitely many iterations we reach a SM strategy, σ, where no improvement is possible. This σ must be optimal. Since each local improvement step can be done in P-time and increases sum total reward, the problem is in PLS. □

4 The Complexity of Reward 1-RMDPs and 1-RSSGs

Theorem 3. *There is a P-time algorithm for computing the exact optimal value (including the possible value ∞) of a 1-RMDP with positive rewards, in both the case where the single player aims to maximize, or to minimize, the total reward.*

We consider maximizing and minimizing 1-RMDPs separately.

Maximizing reward 1-RMDPs

We are given a maximizing reward 1-RMDP (i.e., no $Type_{\min}$ nodes in the 1-RSSG). Let us call the following LP "*max-LP* ":

Minimize $\sum_{u \in Q} x_u$
Subject to:

$x_u = 0$	for all $u \in Type_0$
$x_u \geq \sum_{v \in n(u)} p_{u,v}(x_v + c_{u,v})$	for all $u \in Type_{rand}$
$x_u \geq x_{en} + x_{(b,ex')} + c_u$	for all $u = (b, en) \in Type_{call}$; ex' is the exit of $Y(b)$.
$x_u \geq (x_v + c_{u,v})$	for all $u \in Type_{\max}$ and all $v \in n(u)$
$x_u \geq 0$	for all vertices $u \in Q$

We show that, when the value vector $\mathbf{r}^*$ is finite, it is precisely the optimal solution to the above max-LP, and furthermore that we can use this LP to find

and eliminate vertices u for which $r^*_u = \infty$. Note that if $\mathbf{r}^*$ is finite then it fulfills all the constraints of the max-LP, and thus it is a feasible solution. We will show that it must then also be an optimal feasible solution. We first have to detect vertices u such that $\mathbf{r}^*_u = \infty$. For the max-linear equation system P, we define the underlying directed dependency graph G, where the nodes are the set of vertices, Q, and there is an edge in G from u to v if and only if the variable x_v occurs on the right hand side in the equation defining variable x_u in P. We can decompose this graph in linear time into strongly connected components(SCCs) and get an SCC DAG $SCC(G)$, where the set of nodes are SCCs of G, and an edge goes from one SCC A to another B, iff there is an edge in G from some node in A to some node in B. We will call a subset $U \subseteq Q$ of vertices *proper* if all vertices reachable in G from the vertices in U are already in U. We also use U to refer to the corresponding set of variables. Clearly, such a proper set U must be a union of SCCs, and the equations restricted to variables in U do not use any variables outside of U, so they constitute a proper equation system on their own. For any proper subset U of G, we will denote by max-LP$|_U$ a subset of equations of max-LP, restricted to the constraints corresponding to variables in U and with new objective $\sum_{u \in U} x_u$. Analogously we define $P|_U$, and let $\mathbf{x}|_U$ be the vector $\mathbf{x}$ with entries indexed by any $v \notin U$ removed.

Proposition 1. *Let U be any proper subset of vertices. (I) The vector $\mathbf{r}^*|_U$ is the LFP of $P|_U$. (II) If $r^*_u = \infty$ for some vertex u in an SCC S of G, then $r^*_v = \infty$ for all $v \in S$. (III) If $\mathbf{r}'$ is an optimal bounded solution to* max-LP$|_U$*, then $\mathbf{r}'$ is a fixed point of $P|_U$. (IV) If max-LP$|_U$ has a bounded optimal feasible solution $\mathbf{r}'$, then $\mathbf{r}' = \mathbf{r}^*|_U$.*

Theorem 4. *We can compute $\mathbf{r}^*$ for the max-linear equation system P, including the values that are infinite, in time polynomial in the size of the 1-RMDP.*

Proof. Build dependency graph G of P and decompose it into SCC DAG $SCC(G)$. We will find the LFP solution to P, bottom-up starting at a bottom SCC, S_1. We solve max-LP$|_{S_1}$ using a P-time LP algorithm. If the LP is feasible then the optimal (minimum) value is bounded, and we plug in the values of the (unique) optimal solution as constants in all other constraints of max-LP. We know this optimal solution is equal to $\mathbf{r}^*|_{S_1}$, since S_1 is *proper*. We do the same, in bottom-up order, for remaining SCCs $S_2, \ldots, S_l$. If at any point after adding the new constraints corresponding to the variables in an SCC S_i, the LP is *infeasible*, we know from Proposition 1 (IV), that at least one of the values of $\mathbf{r}^*|_{S_i}$ is ∞. So by Proposition 1 (II), all are. We can then mark all variables in S_i as ∞, and also mark all variables in the SCCs that can reach S_i in $SCC(G)$ as ∞. Also, at each step we add to a set U the SCCs that have finite optimal values. At the end we have a maximal *proper* such set U, i.e., every variable outside of U has value ∞. We label the variables not in U with ∞, obtaining the vector $\mathbf{r}^*$. □

Minimizing reward 1-RMDPs

Given a minimizing reward 1-RMDP (i.e., no $Type_{\max}$ nodes) we want to compute $\mathbf{r}^*$. Call the following LP "*min-LP:* "

Maximize $\sum_{u \in Q} x_u$
Subject to:

$x_u = 0$	for all $u \in Type_0$
$x_u \leq \sum_{v \in n(u)} p_{u,v}(x_v + c_{u,v})$	for all $u \in Type_{rand}$
$x_u \leq x_{en} + x_{(b,ex')} + c_u$	for all $u = (b, en) \in Type_{call}$; ex' is the exit of $Y(b)$.
$x_u \leq (x_v + c_{u,v})$	for all $u \in Type_{\min}$ and all $v \in n(u)$
$x_u \geq 0$	for all vertices $u \in Q$

Lemma 1. *For any* proper *set U, if an optimal solution $\mathbf{x}$ to* min-LP$|_U$ *is bounded, it is a fixed point of the min-linear operator $P|_U$. Thus, if min-LP$|_U$ has a bounded optimal feasible solution then $\mathbf{r}^*|_U$ is bounded (i.e., is a real vector).*

From min-LP we can remove variables $x_u \in Type_0$, by substituting their occurrences with 0. *Assume, for now, that we can also find and remove all variables x_u such that $r^*_u = \infty$.* By removing these 0 and ∞ variables from P we obtain a new system P', and a new LP, min-LP$'$.

Lemma 2. *If ∞ and 0 nodes have been removed, i.e., if $\mathbf{r}^* \in (0, \infty)^n$, then $\mathbf{r}^*$ is the unique optimal feasible solution of min-LP$'$.*

Proof. By Corollary 1, player 2 has an optimal SM strategy, call it τ, which yields the finite optimal reward vector r^*. Once strategy τ is fixed, we can define a new equation system $P'_\tau(\mathbf{x}) = A_\tau \mathbf{x} + b_\tau$, where A_τ is a nonnegative matrix and b_τ is a vector of average rewards per single step from each node, obtained under strategy τ. We then have $\mathbf{r}^* = \lim_{k \to \infty} (P'_\tau)^k(0)$, i.e., $\mathbf{r}^*$ is the LFP of $x = P'(x)$.

Proposition 2. *(I)* $\mathbf{r}^* = (\sum_{k=0}^{\infty} A_\tau^k) b_\tau$. *(II) If $\mathbf{r}^*$ is finite, then* $\lim_{k \to \infty} A_\tau^k = 0$, *and thus* $(I - A_\tau)^{-1} = \sum_{i=0}^{\infty} (A_\tau)^i$ *exists (i.e., is a finite real matrix).*

Now pick an optimal *SM* strategy τ for player 2 that yields the finite $\mathbf{r}^*$. We know that $\mathbf{r}^* = (I - A_\tau)^{-1} b_\tau$. Note that $\mathbf{r}^*$ is a feasible solution of the min-LP$'$. We show that for any feasible solution $\mathbf{r}$ to min-LP$'$, $\mathbf{r} \leq \mathbf{r}^*$. From the LP we can see that $\mathbf{r} \leq A_\tau \mathbf{r} + b_\tau$ (because this is just a subset of the constraints) and in other words $(I - A_\tau)\mathbf{r} \leq b_\tau$. We know that $(I - A_\tau)^{-1}$ exists and is non-negative (and finite), so multiply both sides by $(I - A_\tau)^{-1}$ to get $\mathbf{r} \leq (I - A_\tau)^{-1} b_\tau = \mathbf{r}^*$. Thus $\mathbf{r}^*$ is the optimal feasible solution of min-LP$'$. □

For $u \in Q$, consider the LP: **Maximize** x_u, **subject to:** the same constraints as min-LP, except, again, remove all variables $x_v \in Type_0$. Call this u-min-LP$'$.

Theorem 5. *In a minimizing 1-RMDP, for all vertices u, value $\mathbf{r}^*_u$ is finite iff u-min-LP$'$ is feasible and bounded. Thus, combined with Lemma 2, we can compute the exact value (even if ∞) of minimizing reward 1-RMDPs in P-time.*

Complexity of (1-)RSSGs with positive rewards

Theorem 6. *Deciding whether the value r^*_u of a 1-RSSG positive reward game is $\geq a$ for a given $a \in [0, \infty]$, is in NP $\cap$ coNP.*

This is immediate from P-time upper bounds for 1-RMDPs, and SM-determinacy: guess a player's SM strategy, and compute the value for the remaining 1-RMDP.

Theorem 7. *Condon's quantitative termination problem for finite SSGs reduces in P-time to the problem of deciding whether* $r_u^* = \infty$.

By contrast, for finite-state SSGs with strictly positive rewards, we can decide in P-time whether the value is ∞, because this is the case iff the value of the corresponding termination game is not 1. Deciding whether an SSG termination game has value 1 is in P-time (see, e.g., [9]).

Finally, we show undecidability for multi-exit RMDPs and RSSGs.

Theorem 8. *For multi-exit positive reward RMDPs it is undecidable to distinguish whether the optimal expected reward for a node is finite or* ∞.

Acknowledgement. Research partly supported by NSF grant CCF-0728736.

References

1. Brázdil, T., Brozek, V., Forejt, V., Kucera, A.: Reachability in recursive markov decision processes. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137. Springer, Heidelberg (2006)
2. Condon, A.: The complexity of stochastic games. Inf.&Comp. 96, 203–224 (1992)
3. Condon, A., Melekopoglou, M.: On the complexity of the policy iteration algorithm for stochastic games. ORSA Journal on Computing 6(2) (1994)
4. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological Sequence Analysis: Probabilistic models of Proteins and Nucleic Acids. Cambridge U. Press, Cambridge (1999)
5. Esparza, J., Kučera, A., Mayr, R.: Model checking probabilistic pushdown automata. In: LICS, pp. 12–21 (2004)
6. Esparza, J., Kučera, A., Mayr, R.: Quantitative analysis of probabilistic pushdown automata: expectations and variances. In: Proc. of 20th IEEE LICS 2005 (2005)
7. Etessami, K., Yannakakis, M.: Recursive markov chains, stochastic grammars, and monotone systems of non-linear equations. In: Proc. of 22nd STACS (2005), http://homepages.inf.ed.ac.uk/kousha/bib_index.html
8. Etessami, K., Yannakakis, M.: Recursive markov decision processes and recursive stochastic games. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580. Springer, Heidelberg (2005)
9. Etessami, K., Yannakakis, M.: Efficient qualitative analysis of classes of recursive markov decision processes and simple stochastic games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884. Springer, Heidelberg (2006)
10. Etessami, K., Yannakakis, M.: On the complexity of Nash equilibria and other fixed points. In: Proc. of 48th IEEE FOCS (2007)
11. Etessami, K., Wojtczak, D., Yannakakis, M.: Recursive stochastic games with positive rewards. Tech report EDI-INF-RR-1224 (July 2007)
12. Fagin, R., Karlin, A., Kleinberg, J., Raghavan, P., Rajagopalan, S., Rubinfeld, R., Sudan, M., Tomkins, A.: Random walks with "back buttons". In: STOC (2000)
13. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Heidelberg (1997)
14. Gawlitza, T., Seidl, H.: Precise relational invariants through strategy iteration. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646. Springer, Heidelberg (2007)

15. Gawlitza, T.: Personal communication (April 2008)
16. Haccou, P., Jagers, P., Vatutin, V.A.: Branching Processes: Variation, Growth, and Extinction of Populations. Cambridge U. Press, Cambridge (2005)
17. Harris, T.E.: The Theory of Branching Processes. Springer, Heidelberg (1963)
18. Hoffman, A., Karp, R.: On nonterminating stochastic games. Manag. Sci. 12, 359–370 (1966)
19. Johnson, D.S., Papadimitriou, C., Yannakakis, M.: How easy is local search? J. Comput. Syst. Sci. 37(1), 79–100 (1988)
20. Juba, B.: On the hardness of simple stochastic games. Master's thesis, CMU (2006)
21. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge (1999)
22. Martin, D.A.: Determinacy of Blackwell games. J. Sym. Log. 63, 1565–1581 (1998)
23. Neyman, A., Sorin, S. (eds.): Stochastic Games and Applications. Kluwer, Dordrecht (2003)
24. Pliska, S.: Optimization of multitype branching processes. Management Sci. 23, 117–124 (1976)
25. Puterman, M.L.: Markov Decision Processes. Wiley, Chichester (1994)
26. Rothblum, U., Whittle, P.: Growth optimality for branching Markov decision chains. Math. Oper. Res. 7(4), 582–601 (1982)
27. Veinott, A.F.: Discrete dynamic programming with sensitive discount optimality criteria. Ann. Math. Statist. 40, 1635–1660 (1969)
28. Wojtczak, D., Etessami, K.: Premo: an analyzer for probabilistic recursive models. In: Proc. of TACAS (2007), http://groups.inf.ed.ac.uk/premo/

Complementation, Disambiguation, and Determinization of Büchi Automata Unified

Detlef Kähler and Thomas Wilke

Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany

Abstract. We present a uniform framework for (1) complementing Büchi automata, (2) turning Büchi automata into equivalent unambiguous Büchi automata, and (3) turning Büchi automata into equivalent deterministic automata. We present the first solution to (2) which does not make use of McNaughton's theorem (determinization) and an intuitive and conceptually simple solution to (3).

Our results are based on Muller and Schupp's procedure for turning alternating tree automata into non-deterministic ones.

1 Introduction

As Büchi automata play a crucial role when it comes to designing decision procedures for mathematical theories (such as S1S and S2S) and solving verification problems, see, for instance, [3,16,21], much work has been devoted to fundamental constructions for Büchi automata, in particular, numerous complementation and determinization[1] procedures have been suggested and lower bounds have been proved, see [3,18,8,19,9,7,22,5] and [13,11,4,12,17,14,15], respectively. The known constructions for complementation have nothing in common with the known constructions for determinization, unless one considers the following detour: To complement a given Büchi automaton, one first determinizes it, then complements the deterministic automaton (which is, in general, straightforward), and finally reconverts the complemented deterministic automaton into a Büchi automaton. This, however, yields much more complex constructions than the ones that aim directly at complementation, not only on an informal level, but also on a formal one: When the number of states of the resulting automata are used for comparing them, then direct complementation constructions are by far superior. For turning a Büchi automaton into an equivalent unambiguous Büchi automaton, the situation is worse. The only disambiguation procedure that has been published [2] goes via determinization and therefore is complex.

In this paper, we establish a uniform framework for complementation, disambiguation, and determinization, where complementation and disambiguation avoid determinization. This framework heavily builds on the work by Muller and Schupp described in [14], where they present a procedure that turns alternating tree automata into non-deterministic ones and point out that, as a byproduct, one obtains a determinization procedure for Büchi (word) automata. From their work we identify a characteristic

[1] Recall that there are Büchi automata for which no equivalent deterministic Büchi automata exist. Determinization in the context of Büchi automata therefore means to turn a Büchi automaton into an equivalent deterministic automaton with an appropriate acceptance condition.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 724–735, 2008.

structure which is associated with a given Büchi automaton and a given word over the same alphabet—the so-called skeleton. We first show that this tree can be produced level by level by an unambiguous Büchi automaton—the so-called slice automaton. We then show that by forming a product of the slice automaton with small Büchi automata (i) an equivalent unambiguous Büchi automaton and (ii) an unambiguous Büchi automaton for the complement can be obtained. The number of states of the complement automaton obtained is at most $4(3n)^n$, which is comparable to other constructions. The upper bound for the size of the unambiguous automaton is the same, which is much smaller than anything one obtains by first determinizing the given automaton, in particular, it is much smaller than the automaton described in [2]. It should be noted that after simple optimizations, the tighter analysis from [5] applies in both cases, (i) and (ii).

We also show how the branching structure of the skeleton can be approximated by a conceptually simple deterministic automaton, which, by forming a product with a generic parity automaton (latest appearance record with hit), can be turned into an equivalent deterministic parity automaton.

Related work. The first comprehensive description of Muller and Schupp's determinization construction for Büchi word automata and a comparison with Safra's determinization construction was given by the first author in his diploma thesis [10]; further data on the relation between these two determinization constructions was presented in [1]. A preliminary version of the complementation construction presented here was described in [20]. The exposition in the present paper is more modular, in particular, [20] did not have the intermediate slice automaton, which is the basis for both complementation and disambiguation.

2 Basic Notation and Definitions

We use standard notation for alphabets and words. A *Büchi automaton* is a tuple $\mathscr{A} = (A, Q, q_I, \Delta, F)$ where A is a finite alphabet, Q is a finite state set, $q_I \in Q$ is an initial state, $\Delta \subseteq Q \times A \times Q$ is a transition relation, and $F \subseteq Q$ is a final state set. A run of $\mathscr{A}$ on a word $u \in A^\omega$ is a word $\rho \in Q^\omega$ satisfying $\rho(0) = q_I$ and $(\rho(i), u(i), \rho(i+1)) \in \Delta$ for all $i < \omega$. Such a run is *accepting* if there exist infinitely many i such that $\rho(i) \in F$. The set of all words accepted by $\mathscr{A}$ is denoted $L(\mathscr{A})$. A Büchi automaton $\mathscr{A}$ as above is *unambiguous* if for every $u \in A^\omega$ there is at most one accepting run of $\mathscr{A}$ on u.

Henceforth, we assume, without loss of generality, that every Büchi automaton is complete, that is, for every $q \in Q$, $a \in A$, there exists $q' \in Q$ such that $(q, a, q') \in \Delta$.

All trees in this paper are binary trees with ordered successors. One way to represent such a tree is by a prefix-closed non-empty subset of $\{0,1\}^*$. If t denotes such a tree and $v0 \in t$, then $v0$ is the left successor or 0-successor of v. Symmetrically, if $v1 \in t$, then $v1$ is the right successor or 1-successor of v. A vertex $v' \in t$ is a descendant of a vertex $v \in t$ if $v \leq_{\mathrm{prf}} v'$, where $\leq_{\mathrm{prf}}$ denotes the prefix order. The set of inner vertices and leafs of a tree t are denoted by $\mathrm{inner}(t)$ and $\mathrm{leafs}(t)$, respectively.

As usual, a tree t is called full binary tree if for every $v \in t$ either $\{v0, v1\} \cap t = \emptyset$ or $\{v0, v1\} \subseteq t$, that is, if every vertex is a leaf or has two successors.

A path through a tree t is a word $\pi \in t^+ \cup t^\omega$ such that $\pi(i+1) \in \{\pi(i)0, \pi(i)1\}$ for all i with $i + 1 < |\pi|$. A branch is a maximum path starting in the root, that is, starting

with ϵ, which denotes the empty word. By abuse of notation, when π is a path we will write $v \in \pi$ to denote that there exists $i < |\pi|$ such that $v = \pi(i)$.

Another way to represent a binary tree is as a tuple $T = (V, s_0, s_1)$ where V is an arbitrary set of vertices and $s_0 \colon V \to V$ and $s_1 \colon V \to V$ are partial functions describing the 0- and 1-successors of the vertices, respectively. The root of such a tree is denoted v_T. As the two representations of trees are interchangeable, in concrete circumstances we will use the representation which is most convenient. The first is referred to as *implicit notation* while the second is referred to as *explicit notation*.

A vertex v of a tree T as above is on level 0 if $v = v_T$, that is, if v is the root of T. It is on level $i + 1$ if it is a successor of a vertex on level i. The width of a tree is the supremum of the number of vertices on any level.

For a set L, an L-labeled tree in implicit notation is a function $t \colon V \to L$ where V is a tree in implicit notation as defined above. For every vertex $v \in V$, the value $t(v)$ is the label of v. By abuse of notation, we will write $v \in t$ instead of $v \in V$. An L-labeled tree in explicit notation is a tuple $T = (V, s_0, s_1, l)$ where (V, s_0, s_1) is a tree in explicit notation as defined above and $l \colon V \to L$ is the labeling function which assigns to each $v \in V$ its label $l(v)$. By abuse of notation, we will write $T(v)$ for $l(v)$. The i-th *slice* of an L-labeled tree is the word which is obtained by reading the labels of the vertices on level i from left to right, that is, it is a word over L.

3 Split Trees and Skeletons

In the rest of this paper, $\mathscr{A}$ denotes a fixed Büchi automaton as above with n states and u an infinite word over the same alphabet.

The split tree for u with respect to $\mathscr{A}$ is a tree which can be thought of as refining the power set construction. The root is labeled with the singleton set consisting of the initial state only. For any vertex on level i, we consider the set of states which can be reached from the states the vertex is labeled with by reading the letter at position i, split this set into final and non-final states, and label the left successor of the vertex with the final states and the right successor with the non-final states. If there are no final or non-final states the respective successor does not exist.

To describe this formally, we introduce more notation. For a set of states $Q' \subseteq Q$ and a letter $a \in A$, we define $\Delta(Q', a) = \{q \in Q \colon \exists q' (q' \in Q' \wedge (q', a, q) \in \Delta)\}$, and, similarly, $\Delta_F(Q', a) = \Delta(Q', a) \cap F$ and $\Delta_{\overline{F}}(Q', a) = \Delta(Q', a) \setminus F$.

The split tree for u with respect to $\mathscr{A}$, denoted t_u^{sp}, is a binary tree in implicit form with labels in 2^Q, inductively defined as follows. For the basis, $\epsilon \in t_u^{\mathrm{sp}}$ and $t_u^{\mathrm{sp}}(\epsilon) = \{q_I\}$. For the inductive step, assume $v \in t_u^{\mathrm{sp}}$ and let $Q' = t_u^{\mathrm{sp}}(v)$, $i = |v|$, and $a = u(i)$. Then:

- If $\Delta_F(Q', a) \neq \emptyset$, then $v0 \in t_u^{\mathrm{sp}}$ and $t_u^{\mathrm{sp}}(v0) = \Delta_F(Q', a)$.
- If $\Delta_{\overline{F}}(Q', a) \neq \emptyset$, then $v1 \in t_u^{\mathrm{sp}}$ and $t_u^{\mathrm{sp}}(v1) = \Delta_{\overline{F}}(Q', a)$.

If ρ is a run of $\mathscr{A}$ on u, then β inductively defined by $\beta(0) = \epsilon$ and $\beta(i+1) = \beta(i)0$ if $\rho(i+1) \in F$ and else $\beta(i+1) = \beta(i)1$ is a branch of t_u^{sp}. Conversely, using König's lemma, one can easily prove that for every infinite branch β of t_u^{sp} there exists a run ρ of $\mathscr{A}$ on u such that $\rho(i) \in t_u^{\mathrm{sp}}(\beta(i))$ for every $i < \omega$. We say that ρ is a *witness* for β.

Using the next definition, it is straightforward to express in terms of t_u^{sp} whether $u \in L(\mathscr{A})$. We say a branch β of a tree is *left-recurring* if it is infinite and $\beta(i)$ is a left successor for infinitely many i. This implies:

Remark 1. $u \in L(\mathscr{A})$ iff in t_u^{sp} there exists a left-recurring branch.

Assume t_u^{sp} has a left-recurring branch. Consider the following inductive process for constructing the "left-most left-recurring branch", denoted λ_u. For the basis, let $\lambda_u(0) = \epsilon$. For the inductive step, assume $\lambda_u(i)$ is defined and let $v = \lambda_u(i)$.

- If $v0$ is on a left-recurring branch of t, then $\lambda_u(i+1) = v0$.
- If $v0$ is not on a left-recurring branch, then $\lambda_u(i+1) = v1$.

That this process yields indeed a left-recurring branch is stated in the following lemma, but it also states another interesting property of λ_u. The lemma uses the following notation: $v \leq_{\mathrm{lft}} v'$ if $|v| = |v|'$ and $v \leq_{\mathrm{lex}} v'$, where $\leq_{\mathrm{lex}}$ denotes the lexicographical ordering.

Lemma 1. *Let $\mathscr{A}$ be a Büchi automaton and u an infinite word over the same alphabet. Assume t_u^{sp} has a left-recurring branch. Then:*

1. *λ_u is left-recurring.*
2. *$\lambda_u \leq_{\mathrm{lex}} \beta$ for every left-recurring branch β of t_u^{sp}.*
3. *Let ρ be a witness for λ_u. Then $\rho(i) \notin t_u^{\mathrm{sp}}(v)$ for all $i < \omega$ and $v \in t_u^{\mathrm{sp}}$ with $v <_{\mathrm{lft}} \lambda_u(i)$.*

One property of the split tree is that, in general, it has unbounded width, which makes it difficult to handle. Since we know from the previous lemma that there is a left-most left-recurring path, it suggests itself to remove a state from a labeling of a vertex if it occurs in a vertex on the same level which is to the left of it. This leads to the following inductive definition of the *reduced split tree* for a word u with respect to $\mathscr{A}$, denoted t_u^{rs}, where the induction is on the lexicographic ordering of the vertices. For the basis, $\epsilon \in t_u^{\mathrm{rs}}$ and $t_u^{\mathrm{rs}}(\epsilon) = \{q_I\}$. For the inductive step, assume $v \in t_u^{\mathrm{rs}}$ and let $Q' = t_u^{\mathrm{rs}}(v)$, $i = |v|$, and $a = u(i)$. Further, let $Q'' = \bigcup\{\Delta(t_u^{\mathrm{rs}}(v'), a) \colon v' \in t_u^{\mathrm{rs}} \wedge v' <_{\mathrm{lft}} v\}$. Observe that Q'' is the set of states which have been assigned to vertices to the left of $v0$. Then:

- If $\Delta_F(Q', a) \setminus Q'' \neq \emptyset$, then $v0 \in t_u^{\mathrm{rs}}$ and $t_u^{\mathrm{rs}}(v0) = \Delta_F(Q', a) \setminus Q''$.
- If $\Delta_{\overline{F}}(Q', a) \setminus Q'' \neq \emptyset$, then $v1 \in t_u^{\mathrm{rs}}$ and $t_u^{\mathrm{rs}}(v1) = \Delta_{\overline{F}}(Q', a) \setminus Q''$.

From Remark 1 and Lemma 1, we can conclude:

Lemma 2. *Let $\mathscr{A}$ be a Büchi automaton and u an infinite word over the same alphabet. Then $u \in L(\mathscr{A})$ iff in t_u^{rs} there exists a left-recurring branch.*

The other important property of t_u^{rs} is:

Remark 2. The tree t_u^{rs} has width at most n.

As we are only interested in the infinite branches of t_u^{rs}, we prune away the finite branches. The *skeleton* for u with respect to $\mathscr{A}$, denoted t_u^{sk}, is the subtree of t_u^{rs} which contains a vertex v if there exists an infinite branch β of t_u^{rs} with $v \in \beta$. By König's lemma, this is equivalent to having an infinite number of descendants.

Remark 3. The skeleton t_u^{sk} has an infinite left-recurring branch iff $u \in L(\mathscr{A})$.

4 Disambiguation and Complementation

The complementation and disambiguation construction to be presented make use of the fact that one can construct an unambiguous automaton that produces the slices of the skeleton. We first describe what exactly we mean by this and how this automaton can be used for disambiguation and complementation. We then turn to the construction of this automaton.

4.1 Slice Automaton and Its Applications

The *i-th skeleton slice* of u with respect to $\mathscr{A}$, denoted $\mathrm{skelslice}_i(u)$, is the i-th slice (see Sect. 2) of the skeleton for u. So $\mathrm{skelslice}_i(u) \in (2^Q)^+$. We say that a skeleton slice $Q_0 \dots Q_{s-1}$ is accepting if there exists $i < s$ such that $Q_i \subseteq F$.

Using the above definition and Remark 3 we can state:

Remark 4. 1. $u \in L(\mathscr{A})$ iff there are infinitely many i such that $\mathrm{skelslice}_i(u)$ is accepting.
2. $u \notin L(\mathscr{A})$ iff there is some i such that $\mathrm{skelslice}_j(u)$ is not accepting for all $j \geq i$.

We say a Büchi automaton $\mathscr{S} = (A, S, s_I, \Delta', F')$ is a *slice automaton* for $\mathscr{A}$ with respect to some function $h\colon S \to (2^Q)^+$ if for every $u \in A^\omega$ there exists exactly one accepting run ρ_u of $\mathscr{S}$ on u and this run has the property that $h(\rho(i)) = \mathrm{skelslice}_i(u)$ for every $i < \omega$.

Proposition 1. *For every Büchi automaton $\mathscr{A}$ there exists a slice automaton with at most $(3n)^n$ states.*

Before we sketch the construction, we explain how this automaton can be used for disambiguation and complementation.

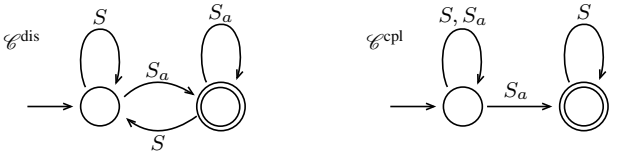
The first part of Remark 4 tells us how to disambiguate: We simply check—using a Büchi condition and in a deterministic fashion—that infinitely accepting slices are produced by the slice automaton. The second part tells us how to complement: We simply guess a point from which onwards all skeleton slices produced by the slice automaton are not accepting.

To turn this into a construction, we use the following product definition. Let $\mathscr{A}$ be as above, $h\colon Q \to B$ a function, and $\mathscr{B} = (B, S, s_I, \Delta', F')$ a Büchi automaton. Then the Büchi automaton $\mathscr{A} \times_h \mathscr{B} = (A, Q \times S \times \{0,1\}, (q_I, s_I, 0), \Delta'', F \times S \times \{0\})$ is defined by: $((q,s,b), a, (q',s',b')) \in \Delta''$ for $(q,a,q') \in \Delta$ and $(s, h(q), s') \in \Delta'$, and where

$$b' = \begin{cases} 0 & \text{if } b = 0 \text{ and } q \notin F, \text{ or } b = 1 \text{ and } s \in F, \\ 1 & \text{if } b = 1 \text{ and } s \notin F', \text{ or } b = 0 \text{ and } q \in F. \end{cases}$$

Remark 5. 1. For every $u \in A^\omega$, $u \in L(\mathscr{A} \times_h \mathscr{B})$ iff there exists an accepting run ρ of $\mathscr{A}$ on u and an accepting run of $\mathscr{B}$ on $h(\rho(0))h(\rho(1)) \dots$.
2. If $\mathscr{A}$ and $\mathscr{B}$ are unambiguous, then so is $\mathscr{A} \times_h \mathscr{B}$.

Let $\mathscr{A}$ be a Büchi automaton, and $\mathscr{S}$ a slice automaton for $\mathscr{A}$ with respect to h. Let $\mathscr{C}^{\mathrm{dis}}$ and $\mathscr{C}^{\mathrm{cpl}}$ be defined by



where S_a and S stand for accepting and non-accepting slices, respectively. The automata $\mathscr{A}^{\mathrm{dis}}$ and $\mathscr{A}^{\mathrm{cpl}}$ are defined by

$$\mathscr{A}^{\mathrm{dis}} = \mathscr{S} \times_h \mathscr{C}^{\mathrm{dis}} \qquad \mathscr{A}^{\mathrm{cpl}} = \mathscr{S} \times_h \mathscr{C}^{\mathrm{cpl}} \ ,$$

respectively.

Fig. 1. Disambiguation and Complementation Construction

From this remark and the above observations, see Remark 4, it is now easy to derive disambiguation and complementation constructions. These are described in Figure 1, where, for complementation, w. l. o. g., we assume that every skeleton has at least one accepting slice.

Theorem 1. *Let $\mathscr{A}$ be a Büchi automaton.*

1. *The automaton $\mathscr{A}^{\mathrm{dis}}$ is an equivalent unambiguous Büchi automaton with at most $4(3n)^n$ states.*
2. *The automaton $\mathscr{A}^{\mathrm{cpl}}$ is a Büchi automaton with at most $4(3n)^n$ states satisfying $L(\mathscr{A}^{\mathrm{cpl}}) = A^\omega \setminus L(\mathscr{A})$ where A is the alphabet of $\mathscr{A}$.*

4.2 Construction of a Slice Automaton

The slice automaton we present produces, in a deterministic fashion, the slices of the reduced split tree. In addition, it guesses which vertices of each level are vertices of the skeleton and which are not.

The problem is that the automaton needs to verify that its guesses are correct. To achieve this, the automaton proceeds in phases as follows. At the beginning of a phase there are certain vertices which have been guessed to be skeleton vertices, and the others have been guessed to be non-skeleton vertices. The automaton follows the descendants of the vertices which have been guessed to be skeleton vertices and for each descendant it decides whether it is a skeleton vertex or not. The latter are put on hold for the next phase. Recall that by definition the non-skeleton vertices are the ones which have only finitely many descendants. So the automaton follows these descendants and ends the current phase as soon as there are no more such descendants. At this point, it reaches a final state and the states on hold take over the role of the states guessed to be non-skeleton vertices. If infinitely many phases are gone through, the guesses were correct.

We first study how the slices of the reduced split tree evolve. So assume $S = Q_0 \dots Q_{s-1}$ is slice i of the reduced split tree for u and $a = u(i)$. For every $j < s$, consider Q'_{2j} and Q'_{2j+1} defined by

$$Q'_{2j} = \Delta_F(Q_j, a) \ , \qquad Q'_{2j+1} = \Delta_{\overline{F}}(Q_j, a) \ .$$

Further, for every $j < s$, let $\tilde{Q}_j = \bigcup_{k<2j} Q'_k$ and set

$$Q''_{2j} = Q'_{2j} \setminus \tilde{Q}_j \ , \qquad Q''_{2j+1} = Q'_{2j+1} \setminus \tilde{Q}_j \ .$$

Slice $i+1$ of the reduced split tree is obtained by removing from $Q''_0 \dots Q''_{2s-1}$ all occurrences of $\emptyset$. We write $\Delta(S, a)$ for this word and $f_{S,a}$ for the partial function $\{0, \dots, 2s-1\} \to \{0, \dots, 2s-1\}$ which tells us where the Q''_j's are moved to because of removing $\emptyset$: If Q''_j is non-empty, then $f_{S,a}(j)$ is the number of non-empty sets in the sequence $Q''_0, \dots, Q''_{j-1}$, and undefined otherwise.

We now describe the state set of our slice automaton. A *decorated slice* is a word $(Q_0, b_0) \dots (Q_{s-1}, b_{s-1})$ where the Q_j's are pairwise disjoint, non-empty subsets of Q and where $b_j \in \{0, *, 1\}$ for $j < s$. The b_j's are meant to indicate whether the corresponding vertex of the respective level is guessed to belong to the skeleton (1), is being checked to be a non-skeleton vertex (0), or is put on hold ($*$). We say such a decorated slice is a reset slice if $b_j \neq 0$ for all $j < s$.

Let $D = (Q_0, b_0) \dots (Q_{s-1}, b_{s-1})$ be a decorated slice, $a \in A$, $S = Q_0 \dots Q_{s-1}$, $f = f_{S,a}$, and $P_0 \dots P_{t-1} = \Delta(S, a)$. An a-successor of D is a decorated slice of the form $(P_0, c_0) \dots (P_{t-1}, c_{t-1})$ where the c_j's satisfy the four conditions below. In what follows, assume $j < s$ and let $b = *$ if D is a reset slice and else $b = 0$.

[D1] If $b_j = 1$, then $f(2j)$ is defined and $c_{f(2j)} = 1$, or $f(2j+1)$ is defined and $c_{f(2j+1)} = 1$.

[D2] If $b_j = 1$, then $c_{f(2j)} \in \{*, 1\}$ and $c_{f(2j+1)} \in \{*, 1\}$, provided $f(2j)$ and $f(2j+1)$, respectively, are defined.

[D3] If $b = *$ and $b_j = *$, then $c_{f(2j)} = 0$ and $c_{f(2j+1)} = 0$, provided $f(2j)$ and $f(2j+1)$, respectively, are defined.

[D4] If $b = 0$ and $b_j \in \{0, *\}$, then $c_{f(2j)} = b_j$ and $c_{f(2j+1)} = b_j$, provided $f(2j)$ and $f(2j+1)$, respectively, are defined.

Observe that these conditions exactly reflect the informal description above. For instance, [D3] says that when a phase is over—we have a reset slice—then the successors of all states on hold are marked as non-skeleton vertices and will be checked to have only a finite number of descendants.

The entire automaton is described in Figure 2. The function h postulated by Proposition 1 is defined by $h((Q_0, b_0) \dots (Q_{s-1}, b_{s-1})) = Q_{i_0} \dots Q_{i_{t-1}}$ where $i_0 < \dots < i_{t-1}$ and $\{i_0, \dots, i_{t-1}\} = \{j < s \colon b_j = 1\}$.

5 Determinization

To motivate our determinization construction, we start with a few definitions, assuming, w. l. o. g., that each skeleton has at least two branches.

We write $\mathrm{branches}(u)$ for the set of branches of the skeleton t_u^{sk}. A *fork* of a binary tree is a vertex which has exactly two successors. The set of all forks of a tree t is denoted by $\mathrm{forks}(t)$.

Our determinization construction is based on the fact that u is accepted by $\mathscr{A}$ iff there exists a left-recurring $\beta \in \mathrm{branches}(u)$. Observe that for each $\beta \in \mathrm{branches}(u)$

Let $\mathscr{A}$ be a Büchi automaton. The *slice automaton* for $\mathscr{A}$ is the Büchi automaton denoted $\mathscr{A}^{\mathrm{slc}}$ and defined by

$$\mathscr{A}^{\mathrm{slc}} = (A, Q', q_I', \Delta', F')$$

where
- Q' is the set of all decorated slices over Q,
- q_I' is the decorated slice $(\{q_I\}, 1)$,
- Δ' contains all transitions (D, a, D') such that D' is an a-successor of D, and
- F' contains all reset slices.

Fig. 2. Definition of Slice Automaton

there exists a unique $v \in \mathrm{forks}(t_u^{\mathrm{sk}})$ and $d \in \{0,1\}$ such that $vd \in \beta$ and $vd \notin \beta'$ for every $\beta' \in \mathrm{branches}(u) \setminus \{\beta\}$. That is, v is the last fork in t_u^{sk} lying on β. We write y_β and d_β for v and d, respectively. Our deterministic automaton tries to identify y_β and d_β for each β and to check that a left-recurring path of t_u^{sk} starts in some $y_\beta d_\beta$. To this end, the automaton approximates the skeleton t_u^{sk} as explained in what follows.

Approximation i of u is the tree consisting of all vertices of t_u^{rs} which potentially belong to t_u^{sk} given only the prefix of length i of u. Formally, *approximation i (of the skeleton) of u* with respect to $\mathscr{A}$ is the tree $a_u^i \subseteq \{0,1\}^*$ defined by $v \in a_u^i$ iff there exists $v' \in t_u^{\mathrm{rs}} \cap \{0,1\}^i$ such that $v \leq_{\mathrm{prf}} v'$.

We prove that the forks of the skeleton can be deduced from the sequence of the approximations of t_u^{sk}:

Lemma 3. *Let $\mathscr{A}$ be a Büchi automaton, u an infinite word over the same alphabet, and $v \in \{0,1\}^*$. Then $v \in \mathrm{forks}(t_u^{\mathrm{sk}})$ iff $v \in \mathrm{forks}(a_u^i)$ for all but finitely many i.*

To be able to handle approximations by a finite automaton, we use $\mathrm{forks}(a_u^i) \cup \mathrm{leafs}(a_u^i)$ as the vertex set of a tree—called contraction—which represents the branching structure of a_u^i. When t is a tree, v and v' are vertices of t, $v_0 \dots v_s$ is the path from v to v', $v_i \notin \mathrm{forks}(t)$ for $i < s$, and $v_s \in \mathrm{forks}(t) \cup \mathrm{leafs}(t)$, we write $v \rightsquigarrow v'$. In other words, $v \rightsquigarrow v'$ if v' is the next descendant of v which is a fork or a leaf.

The *contraction* of a tree t, denoted $C(t)$, is the tree $(\mathrm{forks}(t) \cup \mathrm{leafs}(t), s_0, s_1)$ where for $v, v' \in \mathrm{forks}(t) \cup \mathrm{leafs}(t)$ and $d \in \{0,1\}$, $s_d(v) = v'$ if $vd \rightsquigarrow v'$. The contraction of approximation i of u is denoted C_u^i and called *contraction i of u*. Similarly, $C(t_u^{\mathrm{sk}})$, the contraction of t_u^{sk}, is denoted C_u.

Corollary 1. *Let $\mathscr{A}$ be a Büchi automaton, u an infinite word over the same alphabet, and $v \in \{0,1\}^*$. Then $v \in C_u$ iff $v \in C_u^i$ for all but finitely many i.*

Further, there exists a number i_u such that $C_u \subseteq C_u^j$ for all $j \geq i_u$.

Next we derive a criterion that tells us whether a given $\beta \in \mathrm{branches}(u)$ is left-recurring. Observe that the vertex $y_\beta d_\beta$ is, in general, not an element of the C_u^j's. But, clearly, for $j > i_u$ the vertex y_β is an element of C_u^j and it has a d-successor in C_u^j, which we denote by w_j^β. This vertex is on β. The next lemma tells us that the w_j^β's move along β as j increases, and how one can check whether β is left-recurring or not. It uses the terminology introduced in the following definition.

A sequence $v_0, v_1, \ldots$ of vertices of a branch β *covers* this branch if $v_0 \leq_{\mathrm{prf}} v_1 \leq_{\mathrm{prf}} v_2 \leq_{\mathrm{prf}} \ldots$ and if, for every $i < \omega$, there exists some j such that $|v_j| \geq i$. For $v, w \in \{0,1\}^*$, we say that w is a *left descendant* of v and write $v \to^l w$ if there exists $v' \in \{0,1\}^*0\{0,1\}^*$ such that $w = vv'$, that is, if on the way from v to w there is a "left turn".

Using Corollary 1, we prove:

Lemma 4. *Let $\mathscr{A}$ be a Büchi automaton, u an infinite word over the same alphabet, and $\beta \in \mathrm{branches}(u)$.*

1. *The sequence $w^\beta_{i_u+1}, w^\beta_{i_u+2}, w^\beta_{i_u+3}, \ldots$ covers β.*
2. *The branch β is left-recurring iff there are infinitely many j such that $w^\beta_j \to^l w^\beta_{j+1}$.*

This motivates our determinization construction. The automaton produces isomorphic copies of the contractions, uses them to identify the forks of the skeleton, and checks whether there is one fork whose left or right successors in the approximations cover an infinite branch which is left-recurring. The latter is checked according to the second part of Lemma 4 by using an appropriate acceptance condition.

Note that a finite automaton cannot produce contractions directly, because they have vertices which are vertices of the skeleton t^{sk}_u and as such they are strings of unbounded length. Once an automaton works with isomorphic copies of contractions though, it is not obvious anymore how to identify vertices that cover a left-recurring branch. We solve this by (i) using the same vertex in two consecutive isomorphic copies if and only if it represents the same vertex of the original contractions and (ii) storing in each inner vertex of the isomorphic copies whether or not its right successor represents a left descendant of the vertex it represents. Observe that a left successor always represents a vertex which is a left descendant.

The details of our construction are as follows. Since each contraction has at most $2n-1$ vertices and in each step at most n leafs are added, the vertices for the isomorphic copies of the contractions are taken from the set $\{0, \ldots, 3n-2\}$, which we denote by U. A *contraction tree* is a full binary tree T with at most $2n-1$ vertices from U and labels from $2^Q \cup \{0,1\}$ such that $T(v) \in \{0,1\}$ for each inner vertex and $T(v) \subseteq Q$ for each leaf. Each inner vertex carries information about its right successor (see (ii) above), and each leaf has the same label as the vertex it represents.

We will use the following notation. Assume T is a tree where each inner vertex is labeled by 0 or 1, v' is a descendant of v, and $v_0 \ldots v_s$ the path from v to v'. We write $v \to^0 v'$ if $T(v_i) = 1$ for some $i < s$ or if v_{i+1} is the left successor of v_i for some $i < s-1$. This is the analogue of $\to^l$ for contraction trees.

The crucial part is to describe the transition function. To this end, let T be a contraction tree, assume $v_0, \ldots, v_{s-1}$ is a listing of $\mathrm{leafs}(T)$ in infix order (from left to right), let $S = T(v_0) \ldots T(v_{s-1})$, $a \in A$, $Q'_0 \ldots Q'_{t-1} = \Delta(S, a)$, and $f = f_{S,a}$. We want to define the *a-successor* of T, which we denote by T_a. This tree is obtained from T by a series of transformations modeling (i) extending the corresponding approximation by one level, (ii) removing superfluous vertices, (iii) contracting the tree, and (iv) adjusting the labeling.

We assume that $w_0, w_1, \ldots$ is the sequence of the elements of W defined by $W = U \setminus T$, say in increasing order. Note that W is the set of vertices which are not "used" in T.

Let $\mathscr{A}$ be a Büchi automaton. The *contraction automaton* for $\mathscr{A}$ is the deterministic automaton (without acceptance condition) denoted $\mathscr{A}^{\mathrm{ctr}}$ and defined by

$$\mathscr{A}^{\mathrm{ctr}} = (A, Q', T_I, \delta)$$

where
- Q' is the set of all contraction trees,
- T_I is the one-vertex tree with $v_T = 0$ and $T(v_T) = \{q_I\}$, and
- $\delta(T, a)$ is the a-successor of T for every $T \in Q'$ and $a \in A$.

Fig. 3. Definition of Contraction Automaton

1. *Extend T by one level.* For every $j < s$, first set $T(v_j) = 0$. Then,
   - if $f(2j)$ is defined, make $w_{f(2j)}$ the 0-successor of v_j and set $T(w_{f(2j)}) = Q'_{f(2j)}$, and
   - if $f(2j+1)$ is defined, make $w_{f(2j+1)}$ the 1-successor of v_j and $T(w_{f(2j+1)}) = Q'_{f(2j+1)}$.
2. *Remove superfluous vertices and relabel.* The tree T' is obtained from T by removing all vertices which do not have a descendant in W and changing the labeling as follows. If $v \in \mathrm{forks}(T)$, $d \in \{0, 1\}$, v' is the d-successor of v, and $v' \rightsquigarrow v''$, then $T'(v) = 1$ iff $v \to^0 v''$.
3. *Contract T'.* Finally, T_a is obtained by contracting T' to T_a. (Observe that the definition of contraction given above for trees in implicit notation can easily be adapted to trees in explicit notation.)

The full construction of the deterministic automaton which produces the isomorphic copies of the contractions, the so-called *contraction automaton*, is depicted in Figure 3.

Proposition 2. *Let $\mathscr{A}$ be a Büchi automaton. The contraction automaton $\mathscr{A}^{\mathrm{ctr}}$ has $\lessapprox (4.3n)^{4n}$ states and for every u over the same alphabet the run ρ_u of $\mathscr{A}^{\mathrm{ctr}}$ on u has the following properties:*

1. *For every i, the tree $\rho_u(i)$ is isomorphic with C^i_u, say via $\pi_i \colon \rho_u(i) \to C^i_u$.*
2. *For every $v \in \rho_u(i)$, the vertex v belongs to $\rho_u(i+1)$ iff the vertex $\pi_i(v)$ belongs to C^{i+1}_u, and if this is the case, then $\pi_i(v) = \pi_{i+1}(v)$.*
3. *For every $v \in \mathrm{inner}(\rho_u(i))$, if v' is the right successor of v in $\rho_u(i)$, then $\pi_i(v) \to^l \pi_i(v')$ iff $\rho_u(i)(v) = 1$.*

From the previous proposition we can conclude:

Corollary 2. *Let $\mathscr{A}$, u, and ρ_u as in Proposition 2. Then $u \in L(\mathscr{A})$ iff there is some $v \in U$ such that*

(a) $v \in \rho_u(i)$ for all but finitely many i and

(b) for infinitely many i, the vertex v has a d-successor v'' in $\rho_u(i+1)$ such that v'' is a leaf or $v' \to^0 v''$ in $\rho_u(i)$ for the d-successor v' of v.

To conclude, we explain how (a) and (b) from the corollary can be checked using appropriate acceptance conditions. We first use a custom acceptance condition. Let M be a finite set. A pair (α, β) of functions $Q \times A \to 2^M$ is called an *$\alpha\beta$-condition* over M

if $\alpha(q,a) \supseteq \beta(q,a)$ for all $q \in Q$ and $a \in A$. A run ρ on a word u is accepting with respect to such a condition if there exists some $m \in M$ such that $m \in \alpha(\rho(i), u(i))$ for all but finitely many i and $m \in \beta(\rho(i), u(i))$ for infinitely many i. The order of such a condition is the maximum of the values $|\alpha(q,a)|$.

For the contraction automaton, we choose $M = U$, let $\alpha(T,a)$ be the set of vertices of $\delta(T,a)$ (cf. (a)), and let $\beta(T,a)$ be the set of vertices v of $\delta(T,a)$ satisfying: for some $d \in \{0,1\}$, the vertex v has a d-successor v'' in $\delta(T,a)$ such that v'' is a leaf or $v' \to^0 v''$ in T for the d-successor v' of v (cf. (b)).

Any $\alpha\beta$-condition can be converted into ordinary acceptance conditions:

Lemma 5. *Let $\mathscr{A}$ be a deterministic automaton with n states and an $\alpha\beta$-acceptance condition over a set with s elements and of order m. Then there exists*

1. *an equivalent Muller automaton with $n3^s$ states,*
2. *an equivalent Rabin automaton with $n3^s$ states and s Rabin pairs, and*
3. *an equivalent parity automaton with at most $n(s+1)!$ states and $2m+1$ priorities.*

The proof of the last part of the lemma follows Piterman's construction [15], which itself uses the latest appearance record with hit. The first and second part simply add a function $M \to \{0,1,2\}$ to each state.

We can finally state:

Theorem 2. *Let $\mathscr{A}$ be a Büchi automaton. The deterministic automaton $\mathscr{A}^{\mathrm{ctr}}$ augmented by the above acceptance condition can be converted into a deterministic Muller, Rabin, or parity automaton equivalent to $\mathscr{A}$ with $2^{O(n \log n)}$ states (and n Rabin pairs or $2n+1$ priorities).*

We conclude with two open problems:

1. Give a lower bound for disambiguating Büchi automata.
2. Generalize our determinization construction to Streett automata.

References

1. Schulte-Althoff, C., Thomas, W., Wallmeier, N.: Observations on determinization of Büchi automata. Theor. Comput. Sci. 363(2), 224–233 (2006)
2. Arnold, A.: Rational omega-languages are non-ambiguous. Theor. Comput. Sci. 26, 221–223 (1983)
3. Büchi, J.R.: On a decision method in restricted second-order arithmetic. In: Nagel, E., Suppes, P., Tarski, A. (eds.) Logic, Methodology, and Philosophy of Science: Proc. of the 1960 International Congress, pp. 1–11. Stanford University Press, Stanford (1962)
4. Emerson, E.A., Sistla, A.P.: Deciding full branching time logic. Information and Control 61(3), 175–201 (1984)
5. Friedgut, E., Kupferman, O., Vardi, M.Y.: Büchi complementation made tighter. Int. J. Found. Comput. Sci. 17(4), 851–868 (2006)
6. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: 14th ACM Symposium on the Theory of Computing, San Francisco, Calif, pp. 60–65. ACM, New York (1982)
7. Gurumurthy, S., Kupferman, O., Somenzi, F., Vardi, M.Y.: On complementing nondeterministic Büchi automata. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 96–110. Springer, Heidelberg (2003)

8. Klarlund, N.: Progress measures for complementation of ω-automata with applications to temporal logic. In: 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, pp. 358–367. IEEE, Los Alamitos (1991)
9. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Logic 2(3), 408–429 (2001)
10. Kähler, D.: Determinisierung von ω-Automaten. Diploma thesis, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel (2001)
11. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Information and Control 9, 521–530 (1966)
12. Michel, M.: Complementation is more difficult with automata on infinite words (unpublished notes) (1988)
13. Muller, D.E.: Infinite sequences and finite machines. In: Proceedings of the 4th Annual IEEE Symposium on Switching Circuit Theory and Logical Design, pp. 3–16 (1963)
14. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. Theor. Comput. Sci. 141(1&2), 69–107 (1995)
15. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic Parity automata. In: 21th IEEE Symposium on Logic in Computer Science, Seattle, WA, USA, Proceedings, pp. 255–264. IEEE, Los Alamitos (2006)
16. Ozer Rabin, M.: Decidability of second-order theories and finite automata on infinite trees. Trans. Amer. Math. Soc. 141, 1–35 (1969)
17. Safra, S.: On the complexity of ω-automata. In: 29th Annual Symposium on Foundations of Computer Science, White Plains, New York, pp. 319–327. IEEE, Los Alamitos (1988)
18. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with appplications to temporal logic. Theor. Comput. Sci. 49, 217–237 (1987)
19. Thomas, W.: Complementation of Büchi automata revised. In: Karhumäki, J., Maurer, H.A., Paun, G., Rozenberg, G. (eds.) Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, pp. 109–120. Springer, Heidelberg (1999)
20. Vardi, M.Y., Wilke, Th.: Automata: from logics to algorithms. In: Flum, J., Grädel, E., Wilke, Th. (eds.) Logic and Automata: History and Perspectives. Texts in Logic and Games, vol. 2, pp. 629–736. Amsterdam University Press, Amsterdam (2007)
21. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37
22. Yan, Q.: Lower bounds for complementation of ω-automata via the full automata technique. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 589–600. Springer, Heidelberg (2006)

Tree Projections: Hypergraph Games and Minimality*

Gianluigi Greco[1] and Francesco Scarcello[2]

Dept. of Mathematics[1] and DEIS[2], UNICAL, Via P. Bucci 30B, 87036, Rende, Italy
{ggreco}@mat.unical.it, {scarcello}@deis.unical.it

Abstract. A hypergraph-game characterization is provided for hypergraph tree projections (TPs) and, hence, for the special cases of generalized and fractional hypertree decompositions, where such a characterization was missing and asked for. In this game, as for the Robber and Cops game characterizing tree decompositions, the existence of winning strategies implies the existence of monotone ones, which are yet somehow preferable, because they correspond to minimal tree projections. In fact, it is shown that minimal TPs enjoy a number of nice properties, such as the same kind of connection property as (minimal) tree decompositions of graphs. Finally, it is shown that this property is somehow tight, by giving a negative answer to an open question about a slightly stronger notion of connection property, defined to speed-up the computation of hypertree decompositions.

1 Introduction

Many NP-hard problems in different application areas, ranging, e.g., from AI [13] and Database Theory [4] to Game theory [12], are known to be efficiently solvable when restricted to instances whose underlying structures can be modeled via acyclic graphs or hypergraphs. Indeed, on these kinds of instances, solutions can usually be computed via dynamic programming, by incrementally processing the acyclic (hyper)graph, according to some of its topological orderings. Actually, structures arising from real applications are hardly precisely acyclic. Yet, they are often not very intricate and, in fact, tend to exhibit some limited degree of cyclicity, which suffices to retain most of the nice properties of acyclic ones. Therefore, several efforts have been spent to investigate invariants that are best suited to identify nearly-acyclic graph/hypergraphs, leading to the definition of a number of so-called *structural decomposition methods*. These methods aim at transforming a given cyclic (hyper)graph into an acyclic one, by organizing its edges or its nodes into a polynomial number of clusters, and by suitably arranging these clusters as a tree, called decomposition tree. The original problem instance can then be evaluated over the decomposition tree, with a cost that is usually exponential in the cardinality of the largest cluster, also called *width* of the decomposition.

As far as problems with binary structures are considered, it is known that the notion of *treewidth* [14] is the most general tractable graph decomposition method (see, e.g., [10]). In particular, deciding whether a given graph has treewidth bounded by a fixed natural number k is known to be feasible in linear time—in fact, in time $O(2^{ck^2} \times n)$ for graphs on n vertices [5].

* This work was partially supported by M.I.U.R. under project TOCAI.IT, and by the Institute for High Performance Computing and Networks (ICAR-CNR) under project ICT.P09.001.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 736–747, 2008.

Moreover, a nice game-theoretic characterization exists for tree decompositions in terms of the *Robber and Cops* game [15]: a graph has treewidth bounded by k if and only if $k+1$ cops can capture a Robber that can run at great speed along the edges of graphs, while being not permitted to run trough a vertex that is controlled by a cop. An important property of this game is that there is no restriction on the strategy employed by cops to capture the Robber, while in other (hyper)graph games they are constrained to play *monotonic strategies*, that is, to shrink the Robber's escape space in a monotonically decreasing way. More precisely, it was shown that playing non-monotonic strategies gives no more power to the cops. In many results about treewidth (e.g., [2]), this property turns out to be very useful, because good strategies for the Robber may be easily characterized as those strategies that allow the Robber to run forever.

A further desirable feature of structural decompositions is that adjacent vertices in a decomposition tree enjoy some *connection property*, in order to restrict the search space of the possible decompositions and thus to speed-up their computation. For tree decompositions, it was shown that we may focus on *connected* decompositions, as formalized in [6], that is, that the treewidth of a graph does not change if one looks only at such connected decompositions.

From what we have seen, a clear picture of structural decompositions emerges for binary (graph) instances. The situation pertaining decompositions methods for arbitrary (hypergraphs) instances is much more muddled instead. In fact, a number of powerful decomposition methods have been proposed for hypergraphs, such as the *hypertree decomposition* [7], the *generalized hypertree decomposition* [8], and the *fractional hypertree decomposition* [11]. However, whether or not they enjoy some of the above mentioned desirable features is still unknown. In particular:

(1) Game theoretic characterizations for hypergraph decompositions are missing or not completely satisfactory. For instance, the *Robber and Marshals* game [8], which characterizes the hypertree width, requires monotonic strategies for Marshals, because non-monotonic strategies give some extra-power that does not correspond to valid decompositions [1]. Moreover, a game that characterizes the generalized hypertree width (which is the notion that most resembles treewidth) is still missing and asked for. In [1] it is raised the question whether there is a (natural) game theoretic characterization for generalized hypertree width, where non-monotonicity does not represent a source of additional power. Such a characterization is missing for fractional hypertree decompositions, too.

(2) No systematic study on connection properties of generalized hypertree decompositions (or related notions) similar to those defined for tree decompositions appeared in the literature. Actually, some algorithms have been implemented limiting the search space to a kind of connected decompositions [16], but it is open whether the resulting method is a heuristic one or it does give an exact solution.

In this paper, we give a solution to the above issues and go beyond, by proving our results within the more general and unifying framework of *hypergraphs tree projections*. Roughly, given a pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$, a tree projection of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$ is an acyclic hypergraph $\mathcal{H}_a$ such that each hyperedge of $\mathcal{H}_1$ is contained in some hyperedge of $\mathcal{H}_a$, that is in its turn contained in a hyperedge of $\mathcal{H}_2$, which is called the *resource hypergraph*—see, Figure 1 for an illustration.

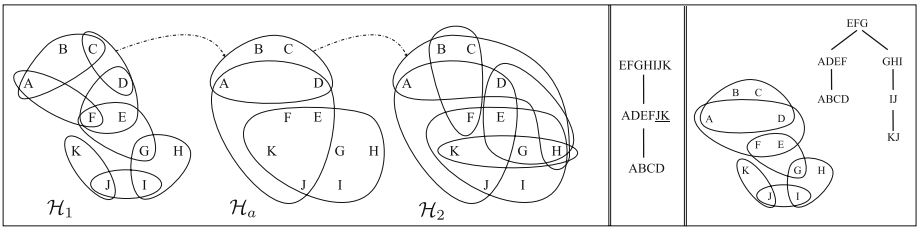


Fig. 1. A Tree Projection $\mathcal{H}_a$ of $\mathcal{H}_1$ with respect $\mathcal{H}_2$; e.g., $\{C, D\} \subseteq \{A, B, C, D\} \subseteq \{A, B, C, D, H\}$. On the center: A Join Tree JT_a for $\mathcal{H}_a$. On the right: A Minimal Tree Projection and a Join Tree for it.

Note that all the (known) structural decomposition methods can be recast as special cases of tree projections, where specific kinds of resource hypergraphs are used. For instance, let k be a fixed natural number, and consider any (hyper)graph $\mathcal{H}_1$ over a set $\mathcal{V}$ of nodes. Then, the width-k tree decompositions of $\mathcal{H}_1$ correspond to the tree projections of $\mathcal{H}_1$ with respect to a hypergraph $\mathcal{H}_2$, whose hyperedges are all possible clusters $B \subseteq \mathcal{V}$ of nodes such that $|B| \leq k + 1$ (or, equivalently, $|B| = k + 1$, if $|\mathcal{V}| \geq k + 1$). In fact, within this framework:

- In Section 3, we define the *Robber and Captain* game, to be played on pairs of hypergraphs, and in the subsequent Section 4 we show that in this game the Captain has a winning strategy if and only if she has a monotone winning strategy.
- In Section 5, we define and investigate a rather natural notion of *minimal* tree projections, where the minimal possible subsets of any resource edge are employed. Intuitively, such projections typically correspond to more efficient decompositions. And, we show that some properties required for normal form decompositions in various notions of structural decomposition methods (see, e.g., [7]) are in fact a consequence of minimality.
- In Section 5.2, we show that tree projections, as well as notable subcases such as the generalized hypertree decompositions, enjoy the same kind of connection property as tree decompositions. Indeed, we show that all minimal tree projections have this nice property, which may help in devising faster algorithms for their computation.
- As bad news, we give a negative answer the question raised in [16]. We observe that the notion of connected decomposition proposed therein differs from the one defined for tree decompositions and mentioned above. In particular, we focus on the case of (generalized) hypertree decompositions as in [16], and we show a hypergraph where enforcing this restriction leads to worse tree projections, i.e., to (generalized) hypertree decompositions with a higher width. It follows that the algorithm proposed in [16] for connected hypertree decompositions is not exact, as far as unrestricted hypertree decompositions are considered.
- Eventually, in Section 6, by exploiting the above notion of minimality, we show that tree projections (as well as generalized hypertree decompositions, fractional hypertree decompositions, and so on) may be characterized in terms of the Robber and Captain game. Thus, all these notions have now a quite natural game characterization where monotone and non-monotone strategies have the same power.

2 Preliminaries

Hypergraphs and Acyclicity. A *hypergraph* $\mathcal{H}$ is a pair (V, H), where V is a finite set of nodes and H is a set of hyperedges such that, for each $h \in H$, $h \subseteq V$. For the sake of simplicity, we always denote V and H by $\mathcal{N}(\mathcal{H})$ and $\mathcal{E}(\mathcal{H})$, respectively. For any set of nodes $X \subseteq V$, the sub-hypergraph of $\mathcal{H}$ induced by X is the hypergraph (X, H') where $H' = \{h \cap X \mid h \in H\}$.

A hypergraph $\mathcal{H}$ is *acyclic* iff it has a join tree [4]. A *join tree* $\mathcal{JT}(\mathcal{H})$ for a hypergraph $\mathcal{H}$ is a tree whose vertices are the edges of $\mathcal{H}$ such that, whenever the same node $X \in V$ occurs in two edges h_1 and h_2 of $\mathcal{H}$, then h_1 and h_2 are connected in $\mathcal{JT}(\mathcal{H})$, and X occurs in each vertex on the unique path linking h_1 and h_2 in $\mathcal{JT}(\mathcal{H})$.

Generalized Hypertree Decomposition. A *hypertree for a hypergraph* $\mathcal{H}$ is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, E)$ is a rooted tree, and χ and λ are labeling functions which associate each vertex $p \in N$ with two sets $\chi(p) \subseteq \mathcal{N}(\mathcal{H})$ and $\lambda(p) \subseteq \mathcal{E}(\mathcal{H})$. If $T' = (N', E')$ is a subtree of T, we define $\chi(T') = \bigcup_{v \in N'} \chi(v)$. In the following, for any rooted tree T, we denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p.

A *generalized hypertree decomposition* of a hypergraph $\mathcal{H}$ is a hypertree $HD = \langle T, \chi, \lambda \rangle$ for $\mathcal{H}$ such that: (1) for each edge $h \in \mathcal{E}(\mathcal{H})$, there exists $p \in vertices(T)$ such that $h \subseteq \chi(p)$; (2) for each node $Y \in \mathcal{N}(\mathcal{H})$, the set $\{p \in vertices(T) \mid Y \in \chi(p)\}$ induces a (connected) subtree of T; and (3) for each $p \in vertices(T)$, $\chi(p) \subseteq \mathcal{N}(\lambda(p))$. The *width* of a generalized hypertree decomposition $\langle T, \chi, \lambda \rangle$ is $max_{p \in vertices(T)} |\lambda(p)|$. The *generalized hypertree width* $ghw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its generalized hypertree decompositions.

A *hypertree decomposition* [7] of $\mathcal{H}$ is a generalized hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ where: (4) for each $p \in vertices(T)$, $\mathcal{N}(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$. The *hypertree width* $hw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its hypertree decompositions. Note that, for any hypergraph $\mathcal{H}$, it is the case that $ghw(\mathcal{H}) \leq hw(\mathcal{H}) \leq 3 \times ghw(\mathcal{H}) + 1$ [3]. Moreover, for any fixed natural number $k > 0$, deciding whether $hw(\mathcal{H}) \leq k$ is feasible in polynomial time (and, actually, is highly-parallelizable) [7], while deciding whether $ghw(\mathcal{H}) \leq k$ is NP-complete [9].

Tree Projections. For two hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ over the same set of nodes, we write $\mathcal{H}_1 \leq \mathcal{H}_2$ iff each hyperedge of $\mathcal{H}_1$ is contained in at least one hyperedge of $\mathcal{H}_2$. Let $\mathcal{H}_1 \leq \mathcal{H}_2$; then, a *tree projection* (short: TP) of $\mathcal{H}_1$ with respect to $\mathcal{H}_2$ is an acyclic hypergraph $\mathcal{H}_a$ such that $\mathcal{H}_1 \leq \mathcal{H}_a \leq \mathcal{H}_2$. Whenever such a hypergraph $\mathcal{H}_a$ exists, we say that the pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$ has a TP. The problem of deciding whether a pair of hypergraphs has a TP is called the *tree projection problem*, and it has recently been proven to be NP-complete [9].

Note that the notion of tree projection is more general than every structural decomposition method. For instance, consider the generalized hypertree decomposition approach: given a hypergraph $\mathcal{H}$ and a natural number $k > 0$, let $\mathcal{H}^k$ denote the hypergraph over the same set of nodes as $\mathcal{H}$, and whose set of edges is given by all possible unions of k edges in $\mathcal{H}$, i.e., $\mathcal{E}(\mathcal{H}^k) = \{h_1 \cup h_2 \cup \cdots \cup h_k \mid \{h_1, h_2, \ldots, h_k\} \subseteq \mathcal{E}(\mathcal{H})\}$. Then, it is well-known and easy to see that $\mathcal{H}$ has generalized hypertree width at most k if and only if there is a TP of $\mathcal{H}$ with respect to $\mathcal{H}^k$. Similarly, for fractional

hypertree decomposition (the most general known decomposition approach) [11], we may define a hypergraph $\mathcal{H}^{fk}$ over the same set of nodes as $\mathcal{H}$, and whose set of edges is given by all possible unions of edges in $\mathcal{H}$ having fractional width at most k, i.e., $\mathcal{E}(\mathcal{H}^{fk}) = \{\bigcup_{h\in H} h \mid H \subseteq \mathcal{E}(\mathcal{H}) \text{ and } fw(H) \leq k\}$. Again, it is easy to see that $\mathcal{H}$ has fractional hypertree width at most k iff there is a TP of $\mathcal{H}$ with respect to $\mathcal{H}^{fk}$.

3 The Robber and Captain Game

The *Robber and Captain* game is played on a pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$ over the same set of vertices $\mathcal{N}(\mathcal{H}_1) = \mathcal{N}(\mathcal{H}_2) = \mathcal{V}$, by a Robber and a Captain controlling some squads of cops, in charge of the surveillance of a number of strategic targets. The Robber stands on a vertex and can run at great speed along the edges of $\mathcal{H}_1$; however, she is not permitted to run trough a vertex that is controlled by a cop. Each move of the Captain involves one squad of cops, which is encoded as an edge $h \in \mathcal{E}(\mathcal{H}_2)$. The Captain may ask any cops in the squad h to run in action, as long as they occupy vertices that are currently reachable by the Robber, thereby blocking an escape path for the Robber. Thus, "second-lines" cops cannot be activated by the Captain. Note that the Robber is fast and may see cops that are entering in action. Therefore, while cops move, the Robber may run trough those positions that are left by cops or not yet occupied. The goal of the Captain is to place a cop on the vertex occupied by the Robber, while the Robber tries to avoid her capture.

For comparison, observe that this game is somehow in the middle between the Robber and Marshals game of [8], where the marshals occupy a full hyperedge at each move, and the Robber and Cops game of [15], where each cop stands on a vertex and thus, if there are enough cops, any subset of any edge can be blocked at each move. Instead, the Captain cannot employ "second-lines" cops, but only cops in charge of positions under possible Robber attacks.

Let V and W two sets of nodes, and $X, Y \in \mathcal{V}$. Then, X is said $[V]$-adjacent to Y if there exists an edge $h_1 \in \mathcal{E}(\mathcal{H}_1)$ such that $\{X, Y\} \subseteq (h_1 - V)$. A $[V]$-path from X to Y is a sequence $X = X_0, \ldots, X_\ell = Y$ of nodes such that X_i is $[V]$-adjacent to X_{i+1}, for each $i \in [0...\ell\text{-}1]$. We say that X $[V]$-touches Y if X is $[\emptyset]$-adjacent to $Z \in \mathcal{V}$, and there is a $[V]$-path from Z to Y; similarly, X $[V]$-touches the set W if X $[V]$-touches some node $Y \in W$. We say that W is $[V]$-connected if $\forall X, Y \in W$ there is a $[V]$-path from X to Y. A $[V]$-component of $\mathcal{H}_1$ is a maximal $[V]$-connected non-empty set of nodes $W \subseteq (\mathcal{V} - V)$. For any $[V]$-component C, let $\mathcal{E}(C) = \{h \in \mathcal{E}(\mathcal{H}_1) \mid h \cap C \neq \emptyset\}$, and for a set of edges $H \subseteq \mathcal{E}(\mathcal{H}_1)$, let $\mathcal{N}(H)$ denote the set of nodes occurring in H, that is $\mathcal{N}(H) = \bigcup_{h\in H} h$.

Definition 1 (R&C **Game).** The Robber and Captain game on $(\mathcal{H}_1, \mathcal{H}_2)$ (short: R&C$(\mathcal{H}_1, \mathcal{H}_2)$ game) is formalized as follows. A *position* for the Captain is a set M of vertices where the cops stand such that $M \subseteq h_2$, for some edge (squad) $h_2 \in \mathcal{E}(\mathcal{H}_2)$. A *configuration* is a pair (M, v), where M is a position for the Captain, and $v \in \mathcal{V}$ is the vertex where the Robber stands. The initial configuration is $(\emptyset, v_0)$, where v_0 is a vertex arbitrarily picked by the Robber.

Let (M_i, v_i) be the configuration at step i. This is a capture configuration, where the Captain wins, if $v_i \in M_i$. Otherwise, the Captain activates the cops in a novel position

M_{i+1} such that: $\forall X \in M_{i+1}$, X $[M_i]$-touches v_i; then, the Robber selects a node v_{i+1} such that there is a $[M_i \cap M_{i+1}]$-path from v_i to v_{i+1}. If the game continues forever, the Robber wins. □

Note that it does not make sense for the Captain to assume that the Robber is on a particular vertex, given the ability of the Robber of changing her positions before the cops land. Thus, given a configuration (M_i, v_i), we may assume w.l.o.g. that the next Captain's move is only determined by the $[M_i]$-component that contains v_i, rather than by v_i itself. And, accordingly, positions can equivalently be written as (M_i, C_i), where C_i is an $[M_i]$-component. In this case, capture configurations have the form $(M, \emptyset)$, and the initial configuration has the form $(\emptyset, \mathcal{V})$.

Definition 2 (Strategies). A *strategy* σ (for R&C$(\mathcal{H}_1, \mathcal{H}_2)$) is a function that encodes the *moves* of the Captain, i.e., given a configuration (M_i, C_i), with $C_i \neq \emptyset$, σ returns a position M_{i+1} such that: $\forall X \in M_{i+1}$, X $[M_i]$-touches C_i.

A *game-tree* for σ is a rooted tree $T(\sigma)$ defined over configurations as follows. Its root is the configuration $(\emptyset, \mathcal{V})$. Let (M_i, C_i) be a node in $T(\sigma)$ and let $M_{i+1} = \sigma(M_i, C_i)$. Then, (M_i, C_i) contains exactly one child (M_{i+1}, C_{i+1}), for each $[M_{i+1}]$-component C_{i+1} such that $C_i \cup C_{i+1}$ is $[M_i \cap M_{i+1}]$-connected; we call such a C_{i+1} an $[(M_i, C_i), M_{i+1}]$-option for the Robber. No further edge or node is in $T(\sigma)$.

Then, σ is said a *winning* strategy if $T(\sigma)$ is a finite tree. Moreover, define a position M_{i+1} to be a *monotone* move of the Captain in (M_i, C_i), if for each $[(M_i, C_i), M_{i+1}]$-option C_{i+1}, $C_{i+1} \subseteq C_i$. We say that σ is a *monotone* strategy if, for each edge from (M_i, C_i) to (M_{i+1}, C_{i+1}), it holds that M_{i+1} is a monotone move in (M_i, C_i). □

4 Monotone vs Non-monotone Strategies

In this section, we show that there is no incentive for the Captain to play a strategy σ that is not monotone, since it is always possible for her to construct and play a monotone strategy σ' that is equivalent to σ, i.e., such that σ' is winning if and only if σ is winning. This crucial property conceptually relates our game with the *Robber and Cops game* characterizing the *treewidth* [15], and differentiates it from most of the hypergraph-based games in the literature, in particular, from the *Robber and Marshals game*, whose monotone strategies characterize hypertree decompositions [8], while non-monotone strategies do not correspond with valid decompositions [1].

For any component C, denote by $\mathrm{Fr}(C)$ the *frontier* of C, i.e., the set $\mathrm{Fr}(C) = \mathcal{N}(\mathcal{E}(C)) = C \cup \{Z \mid \exists X \in C, h \in \mathcal{E}(\mathcal{H}_1)$ s.t. $\{X, Z\} \subseteq h\}$. Let $M_{i+1} = \sigma(M_i, C_i)$ and let C_{i+1} be an $[M_{i+1}]$-component (that is, $C_{i+1} \neq \emptyset$ and M_{i+1} is not a capture position). And, let the *escape-door* of the Robber in $v_i = (M_i, C_i)$ when attacked with M_{i+1} be defined as: $\mathrm{ED}(v_i, M_{i+1}) = M_i \cap \mathrm{Fr}(C_i) - M_{i+1}$.

Theorem 1. *On the* R&C$(\mathcal{H}_1, \mathcal{H}_2)$ *game, the existence of a winning strategy implies the existences of a monotone winning strategy.*

Proof (Idea).[1] Let σ be a winning strategy that is not monotone, and let $p = (M_p, C_p)$ be a configuration reached in $T(\sigma)$ from $(\emptyset, \mathcal{V})$ by a succession of monotone moves.

[1] For space reasons, details of all proofs are omitted and will be available in the full version.

Assume that M_r is the move of the Captain in p and that this is monotone, i.e., for each $[p, M_r]$-option C, $C \subseteq C_p$ (note that any move in the initial configuration is monotone). In particular, let $r = (M_r, C_r)$ be a child of p in $T(\sigma)$, and let $s = (M_s, C_s)$ be a child of r such that $C_s \not\subseteq C_r$. Our aim is to show that σ cannot be a strategy with the minimum total number of cops over all the vertices, precisely because of the non-monotonic step from r to s—thus, such a "minimum" winning strategy, which always exists, is necessarily monotone. Indeed, we can show that it is possible to build a winning strategy σ' that basically coincides with σ, except for the substitution of (M_r, C_r) with a configuration (M'_r, C'_r) such that $M'_r \subset M_r$ and $C_s \subseteq C'_r$. Of course, this will give a larger component C'_r available for Robber's movements. Yet, this component will be monotonically shrinkable by the Captain's squads.

The first ingredient of the proof is to observe that the non-monotonicity from r to s is due to some vertices through which the Robber may escape outside C_r, which were occupied in M_r and which are no longer controlled by the Captain in M_s. Let $\mathrm{ED}(r, M_s)$ be this escape-door set, which depends on r and on the move M_s, and let $C_r, C_{r_1}, ..., C_{r_n}, C'_{r_1}, ..., C'_{r_m}$ be the $[M_r]$-components where the Robber can run when the Captain occupies the position M_r in σ (formally, her $[p, M_r]$-options), with $C_r, C_{r_1}, ..., C_{r_n}$ being those components from which an access to $\mathrm{ED}(r, M_s)$ occurs.

The second ingredient of the proof is, then, to let the Captain play in σ' the position $M'_r = M_r - \mathrm{ED}(r, M_s)$ in place of M_r. Intuitively, we are removing from r the source of non-monotonicity that was suddenly evidenced while moving to s. We shall show that this modification "merged" $C_r, C_{r_1}, ..., C_{r_n}$ into a single $[M'_r]$-component C'_r, without affecting any other part of the strategy outside C'_r.

Finally, the last ingredient of the proof consists in observing that even though C'_r is strictly larger than C_r, the strategy M_s of σ is still applicable to $r' = (M'_r, C'_r)$ and winning. So, the whole subtree of $T(\sigma')$ rooted at (M'_r, C'_r) precisely coincides with the subtree of $T(\sigma)$ rooted at (M_r, C_r), but for the respective roots. □

5 Minimal Tree Projections

In order to establish a link of this hypergraph game with tree projections, we need a closer look at their properties. In particular, in this section, we shall focus on tree projections satisfying an additional minimality condition.

Let $\mathcal{H}$ and $\mathcal{H}'$ be two hypergraphs. Then, $\mathcal{H}$ is *contained* in $\mathcal{H}'$, denoted by $\mathcal{H} \subseteq \mathcal{H}'$, if for each edge $h \in \mathcal{E}(\mathcal{H}) - \mathcal{E}(\mathcal{H}')$, there is an edge $h' \in \mathcal{E}(\mathcal{H}') - \mathcal{E}(\mathcal{H})$ with $h \subseteq h'$ (which in fact entails $h \subset h'$). Note that $\mathcal{H} \subseteq \mathcal{H}'$ does not entail $\mathcal{H} \leq \mathcal{H}'$. Moreover, $\mathcal{H}$ is said *properly contained* in $\mathcal{H}'$, denoted by $\mathcal{H} \subset \mathcal{H}'$, if $\mathcal{H} \subseteq \mathcal{H}'$ and $\mathcal{H} \neq \mathcal{H}'$.

Definition 3 (Minimal TPs). Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two hypergraphs, and let $\mathcal{H}_a$ be a tree projection of $\mathcal{H}_1$ with respect to $\mathcal{H}_2$, i.e., $\mathcal{H}_a$ is an acyclic hypergraph such that $\mathcal{H}_1 \leq \mathcal{H}_a \leq \mathcal{H}_2$. Then, $\mathcal{H}_a$ is *minimal* if there is no tree projection $\mathcal{H}'_a \subset \mathcal{H}_a$ of $\mathcal{H}_1$ with respect to $\mathcal{H}_2$. □

5.1 On the Nice Properties of Minimal Tree Projections

Minimal tree projections enjoy several interesting properties, which will be investigated in the rest of the section and that are crucial to show the mapping with C&R games. In

particular, we shall investigate on the *(i)* existence of minimal tree projections, on the *(ii)* structures of their join trees, and on the *(iii)* preservations of components.

Existence of minimal tree projections. We start by observing that minimal tree projections always exist (as long as a tree projection exists).

Theorem 2. *A pair of hypergraphs* $(\mathcal{H}_1, \mathcal{H}_2)$ *has a* TP *iff* $(\mathcal{H}_1, \mathcal{H}_2)$ *has a minimal* TP.

Join trees for minimal tree projections. An important feature of minimal tree projections is that they admit join trees with a peculiar and rather useful structure. Indeed, we next show that some properties required for normal form decompositions in various notions of structural decomposition methods (see, e.g., [7]) are in fact a consequence of minimality. Observe that a join tree $\mathcal{JT}$ of an acyclic hypergraph $\mathcal{H}$ can be rooted at any of its vertices. In the following, we find convenient to think at join trees as rooted trees; then, for any edge $h \in \mathcal{E}(\mathcal{H})$, we shall denote by $\mathcal{JT}[h]$ the rooted tree obtained by rooting $\mathcal{JT}$ at the vertex h (recall that there is a one-to-one correspondence between the vertices of $\mathcal{JT}$ and $\mathcal{E}(\mathcal{H})$). In addition, given an edge $h' \in \mathcal{E}(\mathcal{H})$, we denote by $\mathcal{JT}[h]_{h'}$ the subtree of $\mathcal{JT}[h]$ rooted at h', and by $\mathcal{N}(\mathcal{JT}[h]_{h'})$ the set of nodes occurring in the vertices of $\mathcal{JT}[h]_{h'}$.

Lemma 3 (Normal Form). *Let* $\mathcal{H}$ *be a minimal* TP *for* $(\mathcal{H}_1, \mathcal{H}_2)$ *and let* h *be any of its edges. Then, there is a join tree* $\mathcal{JT}$ *for* $\mathcal{H}$ *that, rooted at* h*, has the following properties: for each pair* $h_r, h_s \in \mathcal{E}(\mathcal{H})$ *with* h_s *child of* h_r *in* $\mathcal{JT}[h]$*,*
(1) there is exactly one $[h_r]$*-component* C_r *such that* $\mathcal{N}(\mathcal{JT}[h]_{h_s}) = C_r \cup (h_s \cap h_r)$*;*
(2) $h_s \cap C_r \neq \emptyset$*, where* C_r *is the* $[h_r]$*-component in* $\mathcal{H}$ *satisfying Condition 1;*
(3) $h_s \subseteq \mathrm{Fr}(C_r)$*, where* C_r *is the* $[h_r]$*-component in* $\mathcal{H}$ *satisfying Condition 1.*

Therefore, a join tree $\mathcal{JT}[h]$ can be built for a minimal TP $\mathcal{H}$, where each subtree is in charge of decomposing exactly one component. In fact, we shall show that $\mathcal{JT}[h]$ decomposes all the "relevant" components. Assume a join tree $\mathcal{JT}[h]$ is given. Let $comp^{\uparrow}(h) = \mathcal{N}(\mathcal{H})$, and let $comp^{\uparrow}(h_s)$ be the unique $[h_r]$-component with $\mathcal{N}(\mathcal{JT}[h]_{h_s}) = C_r \cup (h_s \cap h_r)$, where h_s is a child of h_r in $\mathcal{JT}[h]$; also, let $edge^{\uparrow}(h) = \emptyset$, and let $edge^{\uparrow}(h_s)$ be h_r.

Lemma 4. *Let* $\mathcal{H}$ *be a minimal* TP *for* $(\mathcal{H}_1, \mathcal{H}_2)$*, and let* $h \in \mathcal{E}(\mathcal{H})$ *and* $\mathcal{JT}[h]$ *as in Lemma 3. Then, for each vertex* h_r *in* $\mathcal{JT}[h]$ *and for each* $[h_r]$*-component* C_r *in* $\mathcal{H}$ *such that* $C_r \subseteq comp^{\uparrow}(h_r)$*, there is exactly one child* h_s *of* h_r *such that: (1)* $\mathcal{N}(\mathcal{JT}[h]_{h_s}) = C_r \cup (h_s \cap h_r)$*; (2)* $h_s \cap C_r \neq \emptyset$*; and, (3)* $h_s \subseteq \mathrm{Fr}(C_r)$*.*

According to the above results, there is a one-to-one correspondence between components and subtrees of the join tree. In particular, for a node h_r and for each child h_s of h_r in $\mathcal{JT}[h]$, we shall denote by $\mathcal{JT}[h]_{h_r,C_r}$ the subtree that is univocally determined by the $[h_r]$-component C_r with $C_r \subseteq comp^{\uparrow}(h_r)$. If $h_r = h$, we simply write $\mathcal{JT}[h]_C$.
Components Preservation. The connectivity of an arbitrary tree projection $\mathcal{H}_a$ for $\mathcal{H}_1$ with respect to $\mathcal{H}_2$ can be characterized not only in terms of its components, but also in terms of the components of the original hypergraph $\mathcal{H}_1$.

Lemma 5. *Let* $\mathcal{H}_a$ *be a TP for* $(\mathcal{H}_1, \mathcal{H}_2)$*. For each* $h \in \mathcal{E}(\mathcal{H}_a)$ *and* $[h]$*-component* C_1 *in* $\mathcal{H}_1$*, there is an* $[h]$*-component* C_a *in* $\mathcal{H}_a$ *such that* $C_1 \subseteq C_a$*.*

Lemma 6. *Let $\mathcal{H}_a$ be a TP for $(\mathcal{H}_1, \mathcal{H}_2)$. For each $h \in \mathcal{E}(\mathcal{H}_a)$ and $[h]$-component C_a in $\mathcal{H}_a$, there are $C_1^1, ..., C_1^n$ $[h]$-components in $\mathcal{H}_1$ with $C_a = \bigcup_{i=1}^n C_1^i$.*

At a first sight, since each edge in $\mathcal{H}_1$ is contained in an edge of $\mathcal{H}_a$, one may naturally be inclined at thinking that such a bigger $\mathcal{H}_a$ adds connectivity, because some nodes that are not (directly) connected by any edge in $\mathcal{H}_1$ may be included together in some edge of $\mathcal{H}_a$. Indeed, in general, for any given set of nodes X, evaluating $[X]$-components in $\mathcal{H}_1$ gives proper subsets of the analogous components evaluated in $\mathcal{H}_a$. We show that this is not the case if minimal tree projections are considered. Intuitively, extra-connections due to the use of bigger edges coming from $\mathcal{H}_2$ are indeed entailed by the connections in the hypergraph $\mathcal{H}_1$ to be projected.

Theorem 7. *Let $\mathcal{H}_a$ be a minimal TP for $(\mathcal{H}_1, \mathcal{H}_2)$. Then, for each edge $h \in \mathcal{E}(\mathcal{H}_a)$, C is an $[h]$-component in $\mathcal{H}_a \Leftrightarrow C$ is an $[h]$-component in $\mathcal{H}_1$.*

Proof (Sketch). Let $\mathcal{H}_a$ be a minimal TP for $(\mathcal{H}_1, \mathcal{H}_2)$. Let $h \in \mathcal{E}(\mathcal{H}_a)$, and assume, by contradiction, that: C is an $[h]$-component in $\mathcal{H}_a \not\Leftrightarrow C$ is an $[h]$-component in $\mathcal{H}_1$. Because of Lemma 5 and Lemma 6, this entails that there is an $[h]$-component C_a in $\mathcal{H}_a$, and $n > 1$ $[h]$-components $C_1^1, ..., C_1^n$ in $\mathcal{H}_1$ such that $C_a = \bigcup_{i=1}^n C_1^i$.

The line of the proof is to show that, based on $\mathcal{H}_a$, it is possible to build a tree projection $\mathcal{H}'_a$ with $\mathcal{H}'_a \subset \mathcal{H}_a$, thereby contradicting the fact that $\mathcal{H}_a$ is minimal. To this end, let H be the set of hyperedges of $\mathcal{H}_a$ that intersects C_a, i.e., $H = \{h_a \mid h_a \in \mathcal{E}(\mathcal{H}_a) \wedge h_a \cap C_a \neq \emptyset\}$. Consider the hypergraph $\mathcal{H}'_a$ defined over the same set of nodes of $\mathcal{H}_a$ and such that: $\mathcal{E}(\mathcal{H}'_a) = \mathcal{E}(\mathcal{H}_a) - H \cup \bigcup_{i=1}^n \{h_a \cap (C_1^i \cup h) \mid h_a \in H\}$.

Eventually, we can show that $\mathcal{H}'_a$ is of the form required, by exploiting the characterizations of minimal tree projections in Lemma 3 and Lemma 4. In particular, to show that $\mathcal{H}'_a$ is acyclic we shall discuss how to build a join tree for it, say $\mathcal{JT}'[h]$, based on a join tree $\mathcal{JT}[h]$ for $\mathcal{H}_a$ in the form required in Lemma 3. The idea is, in a nutshell, to apply a normalization procedure over the subtree $\mathcal{JT}[h]_{C_a}$ which is in charge of decomposing C_a, in order to build the subtrees $\mathcal{JT}'[h]_{C_1^1}$,...,$\mathcal{JT}'[h]_{C_1^n}$, each one being in charge of decomposing an $[h]$-component in $\mathcal{H}_1$. The proof follows from: *Claim E:* $\mathcal{H}'_a \leq \mathcal{H}_2$. *Claim F:* $\mathcal{H}_1 \leq \mathcal{H}'_a$. *Claim G:* $\mathcal{H}'_a$ *is acyclic.* *Claim H:* $\mathcal{H}'_a \subset \mathcal{H}_a$. □

5.2 Connected Tree Projections

A well-known notion of connected decomposition has been introduced for treewidth (see, e.g., [6]). Roughly, in connected tree decompositions, for every bag X in the decomposition tree T of a graph G, every node in X is connected (in the graph G) either to another node in X or to some nodes in (at least two of its) adjacent vertices in T.

It is known that a graph G has a tree decomposition of width at most k iff it has a connected tree decomposition of width at most k. In particular, in [6], an $O(nk^3)$ algorithm, called *make-it-connected*, has been described that builds a width-k' connected tree decomposition of G from any of its width-k tree decompositions, with $k' \leq k$. This notion is next naturally extended the more general framework of tree projections.

Definition 4. A TP $\mathcal{H}_a$ for $(\mathcal{H}_1, \mathcal{H}_2)$ is *connected* if it has a *connected* join tree, i.e., a join tree $\mathcal{JT}$ with the following property: for each pair of adjacent vertices h_r, h_s of $\mathcal{JT}$, the sub-hypergraph of $\mathcal{H}_1$ induced by the nodes in $\mathcal{N}(\mathcal{JT}[h_r]_{h_s})$ is $[\emptyset]$-connected. □

Note that this notion is in fact a generalization of the one defined in [6]. Indeed, they coincide when width-k tree decompositions are considered, that is, whenever we look for TPs of pairs $(\mathcal{H}_1, \mathcal{H}_2)$ where $\mathcal{E}(\mathcal{H}_2) = \{h \subseteq \mathcal{N}(\mathcal{H}_1) \mid |h| \leq k+1\}$. As an example, the TP $\mathcal{H}_a$ reported in the center of Figure 1 is not connected, because it has no connected join trees. E.g. consider the join tree $\mathcal{JT}_a$ and let $h_r = \{E, F, G, H, I, J, K\}$ and $h_s = \{A, D, E, F, J, K\}$. The sub-hypergraph of $\mathcal{H}_1$ induced by $\mathcal{N}(\mathcal{JT}_a[h_r]_{h_s})$ contains only edges $\{D, F, E\}$, $\{K, J\}$, $\{A, B, C\}$, $\{C, D\}$, and $\{A, F\}$, and thus $\{K, J\}$ is clearly disconnected from the others. On the other hand, the join tree reported on the right is connected; in fact, observe that the tree projection from which it derives is a minimal one. We next show this is not by chance.

Theorem 8. *If $\mathcal{H}_a$ is a minimal* TP *for $(\mathcal{H}_1, \mathcal{H}_2)$, then $\mathcal{H}_a$ is a connected* TP.

Proof (Sketch). Assume that $\mathcal{H}_a$ is a minimal TP and let $\mathcal{JT}$ be a join tree for $\mathcal{H}_a$. From the connectedness property of join-trees and the fact that $\mathcal{H}_1 \leq \mathcal{H}_a$, it immediately follows that $\mathcal{JT}$ represents in fact a tree decomposition of $\mathcal{H}_1$, where the edges of $\mathcal{H}_a$ are the bags of the tree decomposition. By contradiction, if $\mathcal{JT}$ is not connected, then apply the above mentioned Algorithm *make-it-connected* [6]. Let $\langle T, \chi \rangle$ be the connected tree decomposition of $\mathcal{H}_1$ obtained by applying this algorithm with $\mathcal{JT}$ as its input. An important property of such a resulting tree decomposition, is that every bag is a subset of some bag of $\mathcal{JT}$, and some of them, say $B_1, \ldots, B_m$ are proper subsets of some B of $\mathcal{JT}$, which instead does not occur in any bag of the new decomposition.

Let $\mathcal{H}'_a$ be the acyclic hypergraph whose set of hyperedges are the bags of T, that is, $\mathcal{E}(\mathcal{H}'_a) = \{\chi(v) \mid v \in \mathcal{N}(T)\}$. Observe that $\mathcal{H}_1 \leq \mathcal{H}'_a \leq \mathcal{H}_a \leq \mathcal{H}_2$, and thus $\mathcal{H}'_a$ is a TP of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$. Moreover, for any B_i $(1 \leq i \leq m)$, $B_i \in \mathcal{E}(\mathcal{H}'_a) - \mathcal{E}(\mathcal{H}_a)$, $B \in \mathcal{E}(\mathcal{H}_a) - \mathcal{E}(\mathcal{H}'_a)$, and $B_i \subset B$. It follows that $\mathcal{H}'_a \subset \mathcal{H}_a$. Contradiction. □

Eventually, from Theorem 8 and Theorem 2, we get the following result.

Corollary 9 *$(\mathcal{H}_1, \mathcal{H}_2)$ has a* TP *iff $(\mathcal{H}_1, \mathcal{H}_2)$ has a connected* TP.

Another notion of connected decomposition has recently been introduced in [16]. In this setting, a (generalized) hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ is connected if the root r of T is such that $|\lambda(r)| = 1$, and for each pair of nodes p and s, with s child of p in T, and for each $h \in \lambda(s)$, $h \cap \chi(s) \cap \chi(p) \neq \emptyset$. The connected (generalized) hypertree width $c(g)hw$ is the minimum width over all the possible connected (generalized) hypertree decompositions. Whether or not $chw(\mathcal{H}) = hw(\mathcal{H})$ for every hypergraph $\mathcal{H}$ was an open question [16]. Next, we give a negative answer to this question.

Theorem 10. *There is a graph G such that $cghw(G) > hw(G)$.*

Proof (Sketch). Consider the graph G_{hex} in Figure 2 and the hypertree decomposition $HD_{hex} = \langle T, \chi, \lambda \rangle$ of G_{hex} having width 3 ($hw(G_{hex}) \leq 3$). For each vertex p of T, $\chi(p) = \mathcal{N}(\lambda(p))$ holds, and thus Figure 2 shows only the λ labels. Moreover, only the left branch is detailed, showing how to deal with the upper cluster of hexagons. The other subtrees are of the same form, and thus are just depicted as triangles, for the sake of simplicity. This decomposition is not connected (because of the root), and in fact it turns out that the only way to attack such hexagons is in a non-connected way. Indeed,

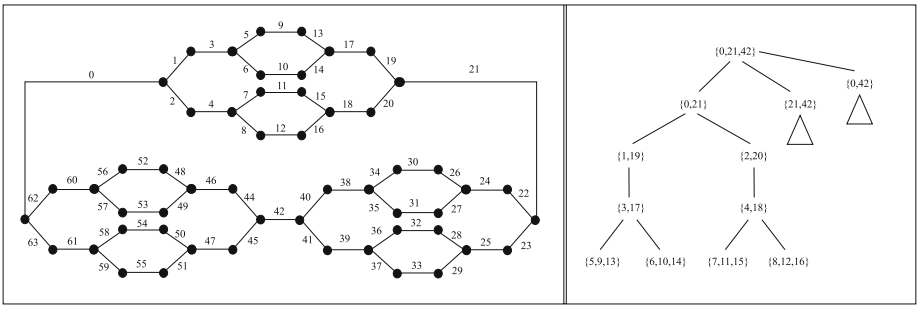


Fig. 2. A graph G_{hex} with $cghw(G_{hex}) = 4 > 3 = hw(G_{hex})$

there is neither a hypertree decomposition nor a generalized hypertree decomposition of G_{hex} that is connected and has width 3, as it can be checked by hands or, more comfortably, by using the algorithm implemented for [16]. □

6 Tree Projections and the R&C Game

In this section, we prove that the Robber and Captain game precisely characterizes the tree projection problem, in the sense that a winning strategy for R&C$(\mathcal{H}_1, \mathcal{H}_2)$ exists if and only if $(\mathcal{H}_1, \mathcal{H}_2)$ has a TP. Hence, any decomposition technique that can be restated in terms of tree projections is in turn characterized by R&C games.

Theorem 11. *If there is a winning strategy in* R&C$(\mathcal{H}_1, \mathcal{H}_2)$*, then* $(\mathcal{H}_1, \mathcal{H}_2)$ *has a* TP.

Proof. Let σ be a winning strategy. W.l.o.g., σ can be assumed to be monotone (cf. Theorem 1). Based on σ we build a hypergraph $\mathcal{H}_a(\sigma)$, where for each vertex (M, C) in $T(\sigma)$, $\mathcal{E}(\mathcal{H}_a(\sigma))$ contains the hyperedge M; and, no further hyperedge is in $\mathcal{E}(\mathcal{H}_a(\sigma))$. Note that, by construction, $\mathcal{H}_a(\sigma) \leq \mathcal{H}_2$, since each position M is such that $M \subseteq h_2$ for some edge $h_2 \in \mathcal{E}(\mathcal{H}_2)$. Let h_1 be an edge in $\mathcal{E}(\mathcal{H}_1)$. Since σ is a winning strategy, we trivially conclude that the Captain has necessarily covered in a complete form h_1 in some position. Thus, $\mathcal{H}_1 \leq \mathcal{H}_a(\sigma)$. Eventually, the fact that $\mathcal{H}_a(\sigma)$ is a tree projection follows from: *Claim I:* $\mathcal{H}_a(\sigma)$ *is acyclic.* □

Theorem 12. *If* $(\mathcal{H}_1, \mathcal{H}_2)$ *has a* TP*, then there is a winning strategy in* R&C$(\mathcal{H}_1, \mathcal{H}_2)$.

Proof. W.l.o.g., let $\mathcal{H}_a$ be a minimal tree projection (cf. Theorem 2). From Theorem 7, we already known that for each edge $h \in \mathcal{E}(\mathcal{H}_a)$, C is a $[h]$-*component* in $\mathcal{H}_a \Leftrightarrow$ C is a $[h]$-*component* in $\mathcal{H}_1$. Let $\mathcal{JT}[h]$ be a join tree for $\mathcal{H}_a$ rooted at an arbitrary edge h, satisfying conditions in Lemma 4. Then, let us build a strategy σ as follows. The Captain initially selects the edge h that is the root of $\mathcal{JT}[h]$, and at any times she moves on the hyperedges in $\mathcal{H}_a$. Let $h_0 = \emptyset$ and $C_0 = \mathcal{N}(\mathcal{H}_1)$. Then, given the current position h_i, for each $[(h_{i-1}, C_{i-1}), h_i]$-option C_i in $\mathcal{H}_1$, $\sigma(h_i, C_i)$ coincides with the root of $\mathcal{JT}[h_i]_{C_i}$. We now state: *Claim J: Let* C_r *be a* $[h_r]$*-component. Then,* $C_r \subseteq comp^{\uparrow}(h_r)$ *iff* $C_r \cup comp^{\uparrow}(h_r)$ *is* $[edge^{\uparrow}(h_r) \cap h_r]$*-connected.*

Thus, σ is well-defined. Indeed, each $[(h_{i-1}, C_{i-1}), h_i]$-option C_i is also such that $C_i \subseteq comp^{\uparrow}(h_i)$ and, hence, $\mathcal{JT}[h_i]_{C_i}$ occurs in $\mathcal{JT}[h]$ because of Lemma 4. Moreover, since at each step $C_i \subseteq C_{i-1}$, σ encodes a strategy that is monotone. □

Acknowledgments. The authors thank the anonymous referees for their useful comments and suggestions, which helped improving the quality of the paper.

References

1. Adler, I.: Marshals, monotone marshals, and hypertree-width. Journal of Graph Theory 47(4), 275–296 (2004)
2. Atserias, A., Bulatov, A., Dalmau, V.: On the Power of k -Consistency. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 279–290. Springer, Heidelberg (2007)
3. Adler, I., Gottlob, G., Grohe, M.: Hypertree-Width and Related Hypergraph Invariants. European Journal of Combinatorics 28, 2167–2181 (2007)
4. Bernstein, P.A., Goodman, N.: The power of natural semijoins. SIAM Journal on Computing 10(4), 751–771 (1981)
5. Bodlaender, H.L., Fomin, F.V.: A Linear-Time Algorithm for Finding Tree Decompositions of Small Treewidth. SIAM Journal on Computing 25(6), 1305–1317 (1996)
6. Fraigniaud, P., Nisse, N.: Connected Treewidth and Connected Graph Searching. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 479–490. Springer, Heidelberg (2006)
7. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. J. of Computer and System Sciences 64(3), 579–627 (2002)
8. Gottlob, G., Leone, N., Scarcello, F.: Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. J. of Computer and System Sciences 66(4), 775–808 (2003)
9. Gottlob, G., Miklós, Z., Schwentick, T.: Generalized hypertree decompositions: np-hardness and tractable variants. In: Proc. of PODS 2007, pp. 13–22 (2007)
10. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. Journal of the ACM 54(1) (2007)
11. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: Proc. of SODA 2006, pp. 289–298 (2006)
12. Daskalakis, C., Papadimitriou, C.H.: Computing pure nash equilibria in graphical games via markov random fields. In: Proc. of ACM EC 2006, pp. 91–99 (2006)
13. Pearson, J., Jeavons, P.G.: A Survey of Tractable Constraint Satisfaction Problems, CSD-TR-97-15, Royal Holloway, Univ. of London (1997)
14. Robertson, N., Seymour, P.D.: Graph minors III: Planar tree-width. Journal of Combinatorial Theory, Series B 36, 49–64 (1984)
15. Seymour, P.D., Thomas, R.: Graph searching and a min-max theorem for tree-width. Journal of Combinatorial Theory, Series B 58, 22–33 (1993)
16. Subbarayan, S., Reif Andersen, H.: Backtracking Procedures for Hypertree, HyperSpread and Connected Hypertree Decomposition of CSPs. In: Proc. of IJCAI 2007, pp. 180–185 (2007)

Explicit Non-adaptive Combinatorial Group Testing Schemes

Ely Porat and Amir Rothschild

Bar-Ilan University, Dept. of Computer Science, 52900 Ramat-Gan, Israel
{porately,amirrot}@gmail.com

Abstract. Group testing is a long studied problem in combinatorics: A small set of r ill people should be identified out of the whole (n people) by using only queries (tests) of the form "Does set X contain an ill human?". In this paper we provide an explicit construction of a testing scheme which is better (smaller) than any known explicit construction. This scheme has $\Theta\left(\min[r^2 \ln n, n]\right)$ tests which is as many as the best non-explicit schemes have. In our construction we use a fact that may have a value by its own right: Linear error-correction codes with parameters $[m, k, \delta m]_q$ meeting the Gilbert-Varshamov bound may be constructed quite efficiently, in $\Theta\left(q^k m\right)$ time.

1 Introduction

Group testing is an important and well known tool in combinatorics. Due to its basic nature, it has been found to be applied in a vast variety of situations. In 2006 DIMACS has dedicated a special workshop solely for the problem of group testing [1]. A representative instance of group testing considers a set of items, each of which can be either defective or non-defective, and the task is to identify the defective items using the minimum number of tests. Each test works on a group of items simultaneously and returns whether or not that group contains at least one defective item. A group testing algorithm is said to be *nonadaptive* if all the tests to be performed are specified in advance. A formal definition is given in Section 2.

Group testing has a long history dating back to at least 1943 [2]. In this early work the problem of detecting syphilitic men for induction into the United States military using the minimum number of laboratory tests was considered. While this idea is still relevant today for testing viruses such as HIV, it is only one of the many applications found for group testing: In the effort of mapping genomes, for example, we have a huge library of DNA sequences, and test whether each of them contains a probe from a given set of DNA pieces [3,4,5]. Somewhat less conventional uses for group testing were introduced lately in pattern matching algorithms [6,7] and in streaming algorithms [8]: For instance, [6] solves the problem of searching for a pattern in a text with a bounded number of mismatches. A recent paper about pattern matching in a streaming model even utilizes group testing twice in the same algorithm [9]. Additional applications

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 748–759, 2008.

of group testing include: compressed sensing [10,11,12,13,14], quality control in product testing [15], searching files in storage systems [16], sequential screening of experimental variables [17], efficient contention resolution algorithms for multiple-access communication [16,18], data compression [19], software testing [20,21], DNA sequencing [22] and other applications in computational molecular biology [23,24,25,26]. In most of the algorithms and applications presented here, our group testing algorithm generates improvements to the results.

Consider the situation where there are n items out of which at most r are defective. It has been shown that in this situation any nonadaptive combinatorial group testing ((n,r)-GT) procedure must use $\Omega(\min[r^2 \log_r n, n])$ tests [27]. The best known schemes use $\Theta\left(\min[r^2 \ln n, n]\right)$ tests [16], and the best known explicit (polynomial time constructable) schemes need as much as $\Theta\left(\min[r^2 \log^2_{r \ln n} n, n]\right)$ tests [16]. In this paper, we present an explicit GT scheme which contains merely $t = \Theta\left(\min[r^2 \ln n, n]\right)$ tests (the same as the best known non-explicit schemes), and takes $\Theta(rn \ln n)$ time to build, which is linear in its representation ($\mathcal{O}\left(t\frac{n}{r}\right)$). Hence, this paper closes the gap between the explicit and non-explicit group testing schemes.

1.1 Error Correction Codes

An error-correcting code (ECC) is a method for encoding data in a redundant way, such that any errors which are introduced can be detected and corrected (within certain limitations). Suppose Alice wants to send Bob a string of k letters from an alphabet of size q using some noisy channel. An $(m,k,d)_q$ error-correction code enables Alice to encode her string to an $m > k$ letters string, such that Bob will be able to detect whether the received message has up to d errors, and even decode the message if it has less than $\frac{d}{2}$ errors. A linear code (LC) is an important type of error-correction code which allows more efficient encoding and decoding algorithms than other codes. Error-correction codes are used in a vast variety of fields including information transmission, data preservation, data-structures, algorithms, complexity theory, and more.

One of the most important goals of coding theory is finding codes that can detect many errors, while having little redundancy. The Gilbert-Varshamov (GV) bound shows this can be done to some extent: We define the *rate* of a code, $R = \frac{k}{m}$ and the *relative distance* of a code, $\delta = \frac{d}{m}$. The GV bound asserts that there are codes with $R \geq 1 - H_q(\delta) - o(1)$ where $H_q(p)$ is the q-ary entropy function $H_q(p) = p \log_q \frac{q-1}{p} + (1-p) \log_q \frac{1}{1-p}$ and $o(1) \xrightarrow[m\to\infty]{} 0$ [28,29]. Though the GV bound is half a century old, no explicit construction of codes meeting it has yet been found. The best known construction takes polynomial time in q^{m-k} [30].

We present a more efficient deterministic construction for linear codes meeting the GV bound. Our construction takes $\Theta\left(q^k m\right)$ time. The importance of this result is apparent when constructing codes with low rates; First, for small rates the GV bound is the best known lower bound on the rate and relative distance of a code. Second, the lower the rate, the slower the previously known best construction, and the faster our construction.

1.2 Previous Results

Since the problem of group testing was first introduced in 1943, many problems related to it and generalizations of it were considered including: fully-adaptive group testing , two staged group testing and selectors [31,32,33,34,35,36], group testing with inhibitors [37,38,39,33], group testing in a random case where a distribution is given on the searched set [32,40,41,42,33], group testing in the presence of errors [43] and more. Regarding the original problem of group testing, Kautz and Singleton [16] proved the existence of GT schemes of size $\Theta\left(r^2 \ln n\right)$, and showed how to explicitly construct schemes of size $\Theta\left(\min[r^2 \log^2_{r \ln n} n, n]\right)$. They also managed to give an explicit construction of schemes of size $\Theta\left(\ln n\right)$ for the special case $r = 2$. Since their work, no asymptotic improvements to the size of the GT scheme were found. One paper succeeded, however, in improving the size of the *explicit* schemes (but only for *constant* values of r): [44] showed how to construct an explicit construction of schemes of size $\Theta\left(r^2 \ln n\right)$ in time polynomial in n^r. From the probabilistic perspective, there is no known Las-Vegas algorithm (though one easily stems from our methods) constructing a scheme of size $\Theta\left(r^2 \ln n\right)$. The only known probabilistic constructions are Monte-Carlo algorithms.

Regarding error-correction codes the picture is more complex. The GV bound was first presented by Gilbert in 1952 [28]. He provided a $\Theta\left(q^m\right)$ time greedy construction for codes meeting his bound. A few years later Varshamov [29] showed linear codes share this bound and Wozencraft [45] offered a $\Theta\left(q^m\right)$ time deterministic construction of such codes. In 1977 Goppa [46] initiated the fruitful study of algebraic geometric codes. Codes eventually found by this study surpass the GV bound for various alphabet sizes and rates [47]. Recently, an explicit, $\Theta\left(m^3 \text{polylog} m\right)$ time construction was given for algebraic geometric codes [48,49]. The best deterministic construction for alphabet sizes and rates where the GV bound is superior to the algebraic geometric bound, was provided in 1993 by Brualdi and Pless [30]. They presented a $\text{poly}(q^{m-k})$ construction of binary linear codes meeting the GV bound. Their construction can be easily generalized to deal with larger alphabets. Even under hardness assumptions, no explicit construction of codes meeting the GV bound has yet been found, though an effort presenting some worthy results is given in [50].

1.3 Our Results

We present the first explicit (n, r)-GT scheme which contains $t = \Theta\left(\min[r^2 \ln n, n]\right)$ tests, thus closing the gap between explicit and non-explicit group testing schemes. Our construction takes $\Theta\left(rn \ln n\right)$ time to build, meaning linear time in its representation $\left(\mathcal{O}\left(t\frac{n}{r}\right)\right)$.

Theorem 1. *Let n and r be positive integers. It is possible to construct a (n, r)-GT containing $\Theta\left(\min[r^2 \ln n, n]\right)$ tests in $\Theta\left(rn \ln n\right)$ time.*

We also present the most efficient deterministic construction for linear codes meeting the GV bound. Our construction builds an $[m, k, \delta m]_q$-LC in $\Theta\left(q^k m\right)$ time.

Theorem 2. *Let q be a prime power, m and k positive integers and $\delta \in [0,1]$. If $k \leq (1 - H_q(\delta))\, m$. It is possible to construct an $[m, k, \delta m]_q$-LC in time $\Theta\left(mq^k\right)$.*

1.4 The Paper Outline

We start this paper with formal definitions in Section 2, and continue by showing a connection between error-correction codes and group testing schemes in Section 3. Then we immediately move to the main result of the paper in Section 4, showing how to efficiently construct small group testing schemes. This construction for group testing schemes uses our construction of a linear code which is given in Section 5. Due to space constraints, most of the proofs have been omitted from this version. All proofs can be found in the full version of the paper [51].

2 Problems Definitions

Definition 1. *Consider a universe U. A family of tests (subsets) $\mathcal{F} \subset \mathcal{P}(U)$ is a group testing scheme of strength r ((n,r)-GT) if for any subset $A \subset U$ of size at most r, and for any element $x \notin A$, there exist a test $B \in \mathcal{F}$ that distinguishes x from A, meaning $x \in B$ while $A \cap B = \varnothing$.*

In order to ease reading, we present short notations of an error-correction code and a linear code.

Definition 2. *An ECC, $\mathcal{C}$, is said to have parameters $(m, k, d)_q$ if it consists of q^k words of length m over alphabet Σ of q elements, and has Hamming distance d. Such an ECC is denoted as $(m, k, d)_q$-ECC.*

Definition 3. *An $[m, k, d]_q$-LC is a special case of an $(m, k, d)_q$-ECC which is over alphabet $\Sigma = \mathbb{F}_q$ when the codewords form a linear subspace over $\mathbb{F}_q^m$. Such a linear code is said to have parameters $[m, k, d]_q$. A linear code has a* generator matrix *$\mathcal{G} \in \mathcal{M}_{m \times k}$ which generates it, meaning $\mathcal{C} = \{\mathcal{G}y \mid y \in \mathbb{F}_q^k\}$.*

3 Background

Our results concerning GT are more natural and straightforward using the combinatorial concepts *selection by intersection* and *strongly-selective family* (SSF) [52]. Selection by intersection means distinguishing an element from a set of elements by intersecting it with another set. More precisely,

Definition 4. *Given a subset $A \subset U$ of a universe U, element $x \in A$ is* selected *by subset $B \subset U$ if $A \cap B = \{x\}$. An element is* selected *by a family of subsets $\mathcal{F} \subset \mathcal{P}(U)$ if one of the subsets in $\mathcal{F}$ selects it.*

An SSF is a family of subsets that selects any element out of a small enough subset of the universe. More precisely,

Definition 5. *A family $\mathcal{F} \subset \mathcal{P}(U)$ is said to be* (n, r)-strongly-selective *if, for every subset $A \subset U$ of size $|A| = r$, all elements of A are selected by $\mathcal{F}$. We call such a family an (n, r)-SSF.*

SSFs and GT schemes are strongly connected: On the one hand, an $(n, r+1)$-SSF is a GT scheme of strength r, and on the other hand, a GT scheme of strength r in a universe of size n is an (n, r)-SSF. For a detailed proof see [16].

In what follows we will focus on SSF constructions. It is important to note that explicit constructions for SSFs give explicit constructions for GT schemes with the same asymptotic behavior. Next we show how to construct an SSF from an ECC, and how good this construction is. The foundations of the idea we present was developed in an earlier work by Kautz and Singleton on superimposed codes [16]. The context and formalisms that were employed are quite distinct from those we require, the idea is quite simple and though, we are not aware of this aspect of their work being developed subsequently. Thus, the following subsection will provide full and complete explanations and proofs of the construction (the full version [51] contains all the proofs).

3.1 Reducing ECCs to SSFs

As it turns out, one can build small strongly-selective families from good error-correction codes having large distance. Both the construction and the proof are given in this Subsection. In a few words, the idea behind the construction is that taking a small set of codewords from the ECC and another codeword w, there must be positions in which w differs from all the words in this set. This is because w differs from any other word in the code in many positions, and so, in a small set of codewords, there must be some shared positions in which all codewords differ from w. Therefore we'll get an SSF if we first translate elements of $[n]$ to codewords, and second, find tests which isolate a codeword w from a set of codewords if it differs from this set in a certain position. We construct such tests by assembling a test for each possible letter in each possible position in the word. A detailed construction follows.

Suppose $\mathcal{C} = \{w_1, ..., w_n\}$ is an $(m, \log_q n, \delta m)_q$-ECC. The constructed SSF, $\mathcal{F}(\mathcal{C})$, will be assembled from all the sets of indexes of codewords that have a certain letter in a certain position. More accurately, for any $p \in [m]$ and $v \in [q]$, define $s_{p,v} = \{i \in [n] \mid w_i[p] = v\}$. Define $\mathcal{F}(\mathcal{C})$ as the set of all such $s_{p,v}$-s: $\mathcal{F}(\mathcal{C}) = \{s_{p,v} \mid p \in [m] \text{ and } v \in [q]\}$.

The size of $\mathcal{F}(\mathcal{C})$ is at most mq. Notice that this construction may be performed in time $\Theta(nm)$ (linear in the size of the representation of $\mathcal{F}$) using Algorithm 1.

```
foreach i ∈ [n] do
    foreach p ∈ [m] do
        insert i into s_{p,w_i[p]} ;
```

Algorithm 1. Constructing an SSF from an ECC

The following Lemma shows that this construction really does result in a small SSF, and more specifically, that $\mathcal{F}(\mathcal{C})$ is an $(n, \lceil \frac{1}{1-\delta} \rceil)$-SSF.

Lemma 1. *Let $\mathcal{C}$ be an $(m, \log_q n, \delta m)_q$-ECC. Then $\mathcal{F}(\mathcal{C})$ is an $(n, \lceil \frac{1}{1-\delta} \rceil)$-SSF.*

For illustration, we consider the following example: If we test our algorithm on the Reed-Solomon $[3, 2, 2]_3$-LC we get a $(9, 3)$-SSF:

$$\mathcal{C} = \{\texttt{000, 111, 222, 012, 120, 201, 021, 102, 210}\}$$

$$\begin{aligned}\mathcal{F}(\mathcal{C}) = \{ \quad & \texttt{\{1,4,7\}, \{2,5,8\}, \{3,6,9\}, \{1,6,8\}, \{2,4,9\},} \\ & \texttt{\{3,5,7\}, \{1,5,9\}, \{2,6,7\}, \{3,4,8\} \quad \}}\end{aligned}$$

4 Main Theorem

Theorem 1. *Let n and r be positive integers. It is possible to construct an (n, r)-SSF of size $\Theta\left(\min[r^2 \ln n, n]\right)$ in $\Theta\left(rn \ln n\right)$ time.*

Proof. If $r^2 \ln n \geq n$, simply return the n tests $\{i\}_{i=1}^{n}$. We continue the proof assuming that $r^2 \ln n < n$. Set $\delta = \frac{r-1}{r}$ (which is equivalent to $r = \frac{1}{1-\delta}$), $q \in [2r, 4r)$ a prime power, $k = \log_q n$ and $m = \frac{k}{1-H_q(\delta)} = \Theta\left(kr \ln r\right) = \Theta\left(r \ln n\right)$.

Use Theorem 3 to construct an $[m, k, \delta m]_q$-LC in time $\Theta\left(nm\right)$. This is possible since $k \leq \left(1 - H_q(\delta)\right) m$.

According to Lemma 1, we can now construct an (n, r)-SSF of size $mq = \Theta\left(r^2 \ln n\right)$. The time this construction will take is $\Theta\left(nm\right) = \Theta\left(rn \ln n\right)$. □

5 Meeting the Gilbert-Varshamov Bound More Efficiently

In this Section we demonstrate a deterministic construction of LCs which meets the GV bound. We developed this deterministic algorithm by taking a randomized algorithm and derandomizing it using the method of conditional probabilities (a full discussion concerning this method is given in [53]). Using this method requires the randomized algorithm to have several non-trivial attributes. First, there need to be a goal function $goal : LinearCodes \rightarrow \mathbb{R}$ which returns a large result whenever the randomized algorithm fails. Second, this function has to have low expectation - lower than the minimum value returned by it when the algorithm fails. Third, the random selections of the algorithm have to be divided into stages with a small number of options to choose from in each. Finally, there should be an efficient algorithm for calculating in each stage of the algorithm the option minimizing the expectation of *goal* given all the selections done until that point. In Subsection 5.1 we'll show the randomized algorithm, present the goal function *goal*, show that the algorithm fails iff $goal(\mathcal{G}) \geq 1$ (where $\mathcal{G}$ is the generator matrix returned by the algorithm), and show that $E(goal) < 1$. In Subsection 5.2 we'll present the derandomized algorithm more accurately, showing

how to divide it to the small stages. We'll also prove it should work, and show how to calculate the option minimizing the expectation of *goal* in each stage. We'll finish this Subsection having an algorithm taking time polynomial in the complexity we desire, we improve it in Subsection 5.3 to acquire the desired complexity.

5.1 The Probabilistic Algorithm

Algorithm 2 is a standard probabilistic algorithm for building linear codes with rate meeting the GV bound.

Input: $m, k \in \mathbb{N}$, $\delta \in [0,1]$ s.t. $k \leq (1 - H_q(\delta))m$
Pick entries of the $m \times k$ generator matrix $\mathcal{G}$ uniformly and independently at random from $\mathbb{F}_q$;
Output: $\mathcal{G}$

Algorithm 2. Probabilistic Construction of a Linear Code

Definition 6. *Given a codeword x of length m, and a distance parameter $\delta \in [0,1]$, we define $\mathcal{B}_\delta(x)$ as the bad event that the weight of x is less than δm, $\omega(x) < \delta m$. By abuse of notation we refer to $\mathcal{B}_\delta(x)$ also as the indicator of the same event.*

If we manage to choose a code with no bad event (not considering the 0 codeword, of course), then the weight of the generated code is larger than δm. As the weight and distance of a linear code are equal, the algorithm succeeds. Therefore, our goal function will be $goal(\mathcal{G}) = \sum_{0 \neq y \in \mathbb{F}_q^k} \mathcal{B}_\delta(\mathcal{G}y)$. The algorithm succeeds iff $goal(\mathcal{G}) = 0$. We now need to show that $E(goal)$ is small. Therefore, we are interested in proving that the probability of a bad event is sufficiently small. In order to do so, we use the following version of the Chernoff bound:

Theorem 2 (Chernoff bound [54])
Assume random variables $X_1, ..., X_m$ are i.i.d. and $X_i \in [0,1]$. Let $\mu = E(X_i)$, and $\epsilon > 0$. Then

$$Pr\left(\frac{1}{m}\sum X_i \geq \mu + \epsilon\right) \leq \left(\left(\frac{\mu}{\mu+\epsilon}\right)^{\mu+\epsilon}\left(\frac{1-\mu}{1-\mu-\epsilon}\right)^{1-\mu-\epsilon}\right)^m$$

Lemma 2. *Let y be a nonzero vector in $\mathbb{F}_q^k$. Let $\mathcal{G}$ be a random generator matrix chosen according to algorithm 2. Then $\log_q\left(Pr\left(\mathcal{B}_\delta(\mathcal{G}y)\right)\right) \leq -m\left(1 - H_q\left(\delta\right)\right)$.*

We will now show that for an appropriate choice of parameters, the expected number of bad events, $E(goal)$, is smaller than 1.

Lemma 3. *Suppose $\mathcal{G}$ is a random generator matrix chosen according to algorithm 2. Suppose that $k \leq (1 - H_q(\delta))m$. Then $E(\text{goal}) < 1$.*

5.2 Derandomizing the Algorithm

Next we will show how to derandomize the algorithm. Algorithm 3 will determine the entries of the generator matrix one by one, while trying to minimize the expectation of the number of bad events, *goal*.

Input: $m, k \in \mathbb{N}$, $\delta \in [0,1]$ s.t. $k \leq (1 - H_q(\delta))m$
Initialize $\mathcal{G}$ to be an $m \times k$ matrix;
foreach $i \in [m]$ **do**
  foreach $j \in [k]$ **do**
    Set $\mathcal{G}[i,j]$ so as to minimize the expected value of *goal*$(\mathcal{G})$ given all the values of $\mathcal{G}$ chosen so far;
Output: $\mathcal{G}$

Algorithm 3. Finding a code having no Bad Events

In what follows, We'll need the following notations:

Definition 7. *We assert that the algorithm is in* step-(i,j) *when it is about to choose the entry* (i,j) *in* $\mathcal{G}$*. We denote the step following* (i,j) *by* $(i,j)+1$*.*

Definition 8. $ST_{(i,j)}$ *will denote the state of the matrix* $\mathcal{G}$ *at step* (i,j) *– i.e. which entries have been fixed to which values.*

Two questions arise from the above description of the algorithm: First, will this algorithm find a code with no bad events? Second, how can we find the value of $\mathcal{G}[i,j]$ in each step of the algorithm?

The answer to the first question is, of course, positive. The presented algorithm works according to the derandomization scheme of conditional probabilities, and so, the number of bad events in the returned solution will be no more than the expectation of this number before fixing any of the letters.

Lemma 4. *The above algorithm will find a code with no bad events,* $\text{goal}(\mathcal{G}) = 0$.

The answer to the second question, regarding how to find what the value of $\mathcal{G}[i,j]$ should be, requires additional work. It would be convenient to order the vectors $y \in \mathbb{F}_q^k$ according to the lexicographic order, setting y_ℓ to be the ℓ-th vector according to the lexicographic order.

We need to know for any codeword the number of positions in which it vanishes, at each step of the algorithm. For this purpose maintain an array A of q^k entries throughout the algorithm. Entry $A[\ell]$ in this array will hold the number of positions in which the code-word $\mathcal{G}y_\ell$ vanished so far. Maintaining this array will require overall $\Theta\left(mq^k\right)$ time. This is due to the fact that in each step (i,j) we only need to consider changing the values $A[y_\ell]$ for $q^{j-1} \leq \ell < q^j$ since the only letters we fixed during this step belong to these words. We claim that the number of position where the word $\mathcal{G}y_\ell$ vanishes determines the conditioned probability of $\mathcal{B}_\delta(y_\ell)$.

Lemma 5. *Consider a codeword $\mathcal{G}y_\ell$ for which all entries up to i were fixed (by the entries selected in $\mathcal{G}$), and entries i to m were not fixed yet. In other words, there exists a word $f \in \mathbb{F}_q^i$ of length i, such that for each possible $\mathcal{G}$, and $\forall t \leq i : (\mathcal{G}y_\ell)[t] = f[t]$, and the same is not true for $i+1$. Also suppose that until now, $\mathcal{G}y_\ell$ doesn't vanish on exactly c positions ($c = |\{t \leq i \mid f[t] \neq 0\}|$). Then $\omega(\mathcal{G}y_\ell) - c \sim B(m-i, 1-\frac{1}{q})$, and $Pr(\mathcal{B}_\delta(\mathcal{G}y_\ell) \mid \forall t \leq i : (\mathcal{G}y_\ell)[t] = f[t])$ is the probability that such a binomial variable will be smaller than $\delta m - c$.*

Now, in step (i,j), For any codeword $\mathcal{G}y_\ell$ s.t. $q^{j-1} \leq \ell < q^j$, we can calculate the probabilities $Pr(\mathcal{B}_\delta(y_\ell) \mid ST_{(i,j)}\ ,\ \mathcal{G}[i,j] = v)$ for all $v \in [q]$ in $\text{poly}(q^k, m)$ time using Lemma 5. Consequently, we can calculate all the expectations $E(\sum_{q^{i-1} \leq \ell < q^i} \mathcal{B}_\delta(y_\ell) \mid ST_{(i,j)}\ ,\ G[i,j] = v)$ for all $v \in [q]$ in $\text{poly}(q^k, m)$ time and find the value of v which minimizes this expectation. Hence, we can complete Algorithm 3 in $\text{poly}(q^k, m)$ time. In the following Subsection we give improvements to this algorithm, showing how to achieve the desired complexity.

5.3 Improving the Deterministic Algorithm

In order to find the letter v which minimizes $E_{i,j,v} = E(\sum_{q^{j-1} \leq \ell < q^j} \mathcal{B}_\delta(y_\ell) \mid ST_{(i,j)}\ ,\ \mathcal{G}[i,j] = v)$, we do not actually have to calculate the q expectations $E_{i,j,v}$. It is enough to calculate the differences of those expectations and a constant value. We will use the constant value which is the expected number of bad events given $ST_{(i,j)}$ and that $(\mathcal{G}y_\ell)[i] \neq 0$ for all $q^{j-1} \leq \ell < q^j$ (Of course, it's improbable that no letters would vanish in step (i,j), as the purpose of this assumption is only to help us with the proof). We denote this constant value $E_{i,j}$.

According to Lemma 5, for any vector y the following holds:

$$Pr(\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{i,j}\ ,\ (\mathcal{G}y)[i] = 0) - Pr(\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{i,j}\ ,\ (\mathcal{G}y)[i] \neq 0) = \binom{m-i}{\delta m - c} \left(1 - \frac{1}{q}\right)^{\delta m - c} \left(\frac{1}{q}\right)^{(m-i)-(\delta m - c)}$$

Denote the above expression $\text{Dif}_{i,j}(y)$. Let T be the time it takes to calculate this expression. Now, we can calculate all q differences $E_{i,j,v} - E_{i,j}$ quite efficiently in the following manner: Initialize a size q array W. Then, run over the vectors y_ℓ for $q^{j-1} \leq \ell < q^j$, and for each subtract the difference $\text{Dif}_{i,j}(y_\ell)$ from cell $v = -y_\ell[j]^{-1} \sum_{t=0}^{j-1} \mathcal{G}[i,t] y_\ell[t]$ in W (since this cell means setting $(\mathcal{G}y_\ell)[i] = \sum_{t=0}^{j-1} \mathcal{G}[i,t] y_\ell[t] + \mathcal{G}[i,j] y_\ell[j] = 0$). After considering all values of y_ℓ , the position with the maximal value in W is the letter we should set for $\mathcal{G}[i,j]$. Each entry number v can be calculated in constant time for all $q^{j-1} \leq \ell < q^j$ if we traverse over the ℓ-s in each step according to Gray code. Overall, the program will calculate mq^k entries, and so, it will take $mq^k T$ time.

Finally, we can drop the T factor and achieve a $\Theta\left(mq^k\right)$ running time by using some standard techniques which appear in the full version of the paper [51]. We conclude the discussion with the following Theorem:

Theorem 3. *Let q be a prime power, m and k positive integers and $\delta \in [0,1]$. If $k \leq (1 - H_q(\delta))\, m$, then it's possible to construct an $[m, k, \delta m]_q$-LC in time $\Theta\left(mq^k\right)$.*

6 Conclusion and Open Problems

We have presented a simple and intuitive construction of linear codes meeting the GV bound. Our construction is the most efficient known construction of such linear codes. We used our codes construction to construct explicitly, in $\Theta(rn \ln n)$ time, very good GT schemes of $\Theta\left(r^2 \ln n\right)$ tests. It would be interesting to study whether our linear codes construction can be made more efficient, or whether it can be improved to construct better codes. While we managed to close the gap between the sizes of explicit and non-explicit group testing schemes, the gap in the important generalization of selectors is still open; closing it is an interesting and important problem. We believe that other important special cases of group testing worth studying include the problem of minimizing the sets accumulative size rather than their number, and also, solely for algorithmic purposes – the case where the tests answers tell not only if there exists an element in the intersection or not, but rather, how many elements are there in it.

References

1. The Center for Discrete Mathematics and Theoretical Computer Science (DIMACS): DIMACS Workshop on Combinatorial Group Testing, The Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) (May 2006)
2. Dorfman, R.: The detection of defective members of large populations. The Annals of Mathematical Statistics 14(4), 436–440 (1943)
3. Barillot, E., Lacroix, B., Cohen, D.: Theoretical analysis of library screening using an n- dimensional pooling strategy. Nucleic Acids Research, 6241–6247 (1991)
4. Bruno, W., Balding, D., Knill, E., Bruce, D., Whittaker, C., Dogget, N., Stalling, R., Torney, D.: Design of efficient pooling experiments. Genomics 26, 21–30 (1995)
5. T., B., J.W., M., P., S.: Maximally efficient two-stage screening. Biometrics 56(8), 833–840 (2000)
6. Clifford, R., Efremenko, K., Porat, E., Rothschild, A.: k-mismatch with don't cares. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 151–162. Springer, Heidelberg (2007)
7. Amir, A., Kapah, O., Porat, E.: Deterministic length reduction: Fast convolution in sparse data and applications. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 183–194. Springer, Heidelberg (2007)
8. Cormode, G., Muthukrishnan, S.: What's hot and what's not: tracking most frequent items dynamically. ACM Trans. Database Syst. 30(1), 249–278 (2005)
9. Porat, B., Porat, E., Rothschild, A.: Pattern matching in a streaming model
10. Muthukrishnan, S.: Some algorithmic problems and results in compressed sensing. In: 44th Allerton Conference on Communication, Control and Computing (2006)
11. Indyk, P.: Explicit constructions for compressed sensing of sparse signals. In: SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms) (2008)
12. Gilbert, A.C., Strauss, M.J., Tropp, J.A., Vershynin, R.: One sketch for all: fast algorithms for compressed sensing. In: STOC 2007: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, pp. 237–246. ACM, New York (2007)

13. Cormode, G., Muthukrishnan, S.: Combinatorial algorithms for compressed sensing. In: Flocchini, P., Gasieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 280–294. Springer, Heidelberg (2006)
14. Iwen, M.A.: A deterministic sub-linear time sparse fourier algorithm via non-adaptive compressed sensing methods. CoRR abs/0708.1211 (2007)
15. Sobel, M., Groll, P.: Group testing to eliminate efficiently all defectives in a binomial sample. Bell Syst. Tech. J. 38, 1179–1252 (1959)
16. Kautz, W., Singleton, R.: Nonrandom binary superimposed codes. IEEE Transaction of InformationTheory 10, 363–377 (1964)
17. Li, C.: A sequential method for screening experimental variables. J. Amer. Sta. Assoc. 57, 455–477 (1962)
18. Wolf, J.: Born again group testing: Multiaccess communications. IEEE Transactions on Information Theory 31(2), 185–191 (1985)
19. Hong, E., Ladner, R.: Group testing for image compression. In: Data Compression Conference, pp. 3–12 (2000)
20. Blass, A., Gurevich, Y.: Pairwise testing. Bulletin of the EATCS 78, 100–132 (2002)
21. Cohen, D., Dalal, S., Fredman, M., Patton, G.: The AETG system: An approach to testing based on combinatiorial design. Software Engineering 23(7), 437–444 (1997)
22. Pevzner, P.A., Lipshutz, R.J.: Towards dna sequencing chips. In: MFCS 1994: Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science 1994, London, UK. LNCS, pp. 143–158. Springer, Heidelberg (1994)
23. Du, D., Hwang, F.: Combinatorial Group Testing and its Applications, 2nd edn. Series on Applied Mathematics, vol. 12. World Scientific, Singapore (2000)
24. Farach, M., Kannan, S., Knill, E., Muthukrishnan, S.: Group testing problems with sequences in experimental molecular biology. In: The Compression and Complexity of Sequences 1997, p. 357 (1997)
25. Ngo, H., Du, D.: A survey on combinatorial group testing algorithms with applications to DNA library screening. In: DIMACS Series Discrete Math. and Theor. Computer Science vol. 55, pp. 171–182. AMS (2000)
26. Balding, D.J., Bruno, W.J., Knill, E., Torney, D.C.: A comparative survey of non-adaptive pooling designs. Institute for Mathematics and Its Applications 81, 133 (1996)
27. Chaudhuri, S., Radhakrishnan, J.: Deterministic restrictions in circuit complexity. In: ACM Symposium on Theory of Computing (STOC), pp. 30–36 (1996)
28. Gilbert, E.: A comparison of signalling alphabets. Bell System Technical Journal 31, 504–522 (1952)
29. Varshamov, R.: Estimateof the number of signals in error correcting codes. Doklady Akadamii Nauk 117, 739–741 (1957)
30. Brualdi, R.A., Pless, V.: Greedy codes. J. Comb. Theory, Ser. A 64(1), 10–30 (1993)
31. Chrobak, M., Gasieniec, L., Rytter, W.: Fast broadcasting and gossiping in radio networks. In: IEEE Symposium on Foundations of Computer Science, pp. 575–581 (2000)
32. Knill: Lower bounds for identifying subset members with subset queries. In: SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms) (1995)
33. Bonis, A.D., Gasieniec, L., Vaccaro, U.: Generalized framework for selectors with applications in optimal group testing. In: ICALP, pp. 81–96 (2003)
34. Clementi, A.E.F., Monti, A., Silvestri, R.: Selective families, superimposed codes, and broadcasting on unknown radio networks. In: SODA, pp. 709–718 (2001)

35. Bonis, A.D., Vaccaro, U.: Constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels. Theor. Comput. Sci. 306(1-3), 223–243 (2003)
36. Bonis, A.D., Gasieniec, L., Vaccaro, U.: Optimal two-stage algorithms for group testing problems. SIAM J. Comput. 34(5), 1253–1270 (2005)
37. Farach, M., Kannan, S., Knill, E., Muthukrishnan, S.: Group testing problems with sequences in experimental molecular biology. In: SEQUENCES 1997: Proceedings of the Compression and Complexity of Sequences 1997, Washington, DC, USA, p. 357. IEEE Computer Society, Los Alamitos (1997)
38. Damaschke, P.: Randomized group testing for mutually obscuring defectives. Inf. Process. Lett. 67(3), 131–135 (1998)
39. Bonis, A.D., Vaccaro, U.: Improved algorithms for group testing with inhibitors. Inf. Process. Lett. 67(2), 57–64 (1998)
40. Berger, T., Levenshtein, V.I.: Asymptotic efficiency of two-stage disjunctive testing. IEEE Transactions on Information Theory 48(7), 1741–1749 (2002)
41. Berger, T., Levenshtein, V.I.: Application of cover-free codes and combinatorial designs to two-stage testing. Discrete Appl. Math. 128(1), 11–26 (2003)
42. A.J., M.: Probabilistic nonadaptive and two-stage group testing with relatively small pools and Dna library screening. Journal of Combinatorial Optimization 2, 385–397 (1998)
43. Knill, E., Bruno, W.J., Torney, D.C.: Non-adaptive group testing in the presence of errors. Discrete Applied Mathematics 88(1-3), 261–290 (1998)
44. Alon, N., Moshkovitz, D., Safra, S.: Algorithmic construction of sets for -restrictions. ACM Transactions on Algorithms 2(2), 153–177 (2006)
45. Wozencraft, J.: Threshold decoding. Personal communication in [55] section 2.5 (1963)
46. Goppa, V.: Codes associated with divisors. Problems of Information Transmission 13(1), 22–26 (1977)
47. Tsfasman, M., Vladut, S., Zink, T.: Modular curves, Shimura curves, and codes better then the Varshamov-Gilbert bound. Math. Nachrichten 109, 21–28 (1982)
48. Shum, K.: A Low-Complexity Construction of Algebric Geometry Codes Better Then the Gilbert-Varshamov Bound. PhD thesis, University of Southern California (December 2000)
49. Shum, K., Aleshnikov, I., Kumar, P., Stichtenoth, H., Deolalikar, V.: A low-complexity algorithm for the construction of algebraic geometric codes better then the Gilbert-Varshamov bound. IEEE Transaction on Information Theory 47(6), 2225–2241 (2001)
50. Cheraghchi, M., Shokrollahi, A., Wigderson, A.: Computational Hardness and Explicit Constructions of Error Correcting Codes. In: 44th Allerton Conference on Communication, Control and Computing (2006)
51. Porat, E., Rothschild, A.: Explicit non-adaptive combinatorial group testing schemes. CoRR abs/0712.3876 (2007)
52. Clementi, A., Monti, A., Silvestri, R.: Distributed broadcast in radio networds of unknown topology. Theoretical Computer Science 302(1–3), 337–364 (2003)
53. Alon, N., Spencer, J.: The Probabilistic Method, 2nd edn. John Wiley and Sons Inc., Chichester (2001)
54. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58(301), 13–30 (1963)
55. Massey, J.L.: Threshold decoding. MIT Press, Cambridge (1963)

Tight Lower Bounds for Multi-pass Stream Computation Via Pass Elimination

Sudipto Guha[1,⋆] and Andrew McGregor[2]

[1] Department of Computer and Information Science, University of Pennsylvania
[2] Information Theory & Applications Center, University of California, San Diego

Abstract. There is a natural relationship between lower bounds in the multi-pass stream model and lower bounds in multi-round communication. However, this connection is less understood than the connection between single-pass stream computation and one-way communication. In this paper, we consider data-stream problems for which reductions from natural multi-round communication problems do not yield tight bounds or do not apply. While lower bounds are known for some of these data-stream problems, many of these only apply to deterministic or comparison-based algorithms, whereas the lower bounds we present apply to any (possibly randomized) algorithms. Our results are particularly relevant to evaluating functions that are dependent on the ordering of the stream, such as the longest increasing subsequence and a variant of tree pointer jumping in which pointers are revealed according to a post-order traversal.

Our approach is based on establishing "pass-elimination" type results that are analogous to the round-elimination results of Miltersen et al. [23] and Sen [29]. We demonstrate our approach by proving tight bounds for a range of data-stream problems including finding the longest increasing sequences (a problem that has recently become very popular [22,16,30,15,12] and we resolve an open question of [30]), constructing convex hulls and fixed-dimensional linear programming (generalizing results of [8] to randomized algorithms), and the "greater-than" problem (improving results of [9]). These results will also clarify one of the main messages of our work: sometimes it is necessary to prove lower bounds directly for stream computation rather than proving a lower bound for a communication problem and then constructing a reduction to a data-stream problem.

1 Introduction

A natural connection between communication complexity and small-space computation in the data-stream model has been widely exploited since the early papers on the data-stream model [2, 19]. For example, a stream computation can be conceptualized as a communication problem between players who know different segments of the input stream. Consequently, a lower-bound on the total amount of communication necessary yields a lower-bound on the amount of memory required by the streaming algorithm. This approach has been very successful and many lower bounds have been proved in this manner (e.g., [2,3,6,18].) However, interesting issues arise when trying to prove lower

⋆ This research was supported by in part by an Alfred P. Sloan Research Fellowship and by NSF Awards CCF-0430376, and CCF-0644119.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 760–772, 2008.

bounds for multi-pass algorithms[1] and these have not been explored as thoroughly as in the single-pass case. Many of the lower bounds that do exist only apply to deterministic algorithms, e.g., recent results on approximating the length of the longest increasing subsequence [15, 12] and solving geometric problems [8], or only apply to restricted classes of algorithm that may only use memory in a certain way [24, 16, 10]. Given that randomization is necessary for many classic data-stream problems (e.g., ℓ_p norms and frequency moments [2, 20]) we consider it natural to seek lower-bounds for fully general randomized algorithms. We are particularly interested in space/pass trade-offs.

The goal of this paper is to construct a framework for proving multi-pass lower bounds for certain types of data-stream problems. Our approach is based on establishing pass-elimination results which tailor ideas from the round-elimination technique in communication complexity [29, 23] specifically to data-stream computation. This allows us to prove results on tree pointer jumping when the data is revealed by a *post-order traversal* rather than level-by-level. This will play a crucial role in proving lower bounds for order-dependent functions in the data-stream model and will exemplify a distinction we wish to make between multi-party communication and data-stream computation. With this technology we prove results for a range of problems including finding the longest increasing subsequence, determining the larger of two values, constructing convex hulls, and fixed dimensional linear programming. The longest increasing subsequence (LIS) problem is to find the increasing sub-stream of maximum length. This problem, and the related problem of estimating the length of the longest increasing subsequence has become very popular in recent years [22, 16, 30, 15, 12]. We prove a tight multi-pass lower bound for LIS, thereby resolving an open question of Sun and Woodruff [30]. We note that this problem exhibits an interesting doubly-exponential space/pass trade-off. The greater-than (GT) problem is to determine which of two integers is larger given the bits of their binary expansion. It is often used as an example of the utility of round-elimination in the communication setting. A lower bound for a data-stream version was given by Chang and Kannan [9] by appealing to the communication result. We show that this lower bound can be substantially improved using the pass-elimination framework and this will further highlight the round-elimination/pass-elimination and data-stream/communication distinctions that are important in this paper. Finally, we generalize bounds for deterministic algorithms for geometric problems including constructing convex hulls and fixed dimensional linear programming [8] to apply to randomized algorithms.

Previous Approaches and Limitations: There do exist multi-pass lower bounds for randomized algorithms for some problems in the data-stream model. However, almost all of these bounds are based on reductions from a relatively small set of communication problems. Many bounds are based on the multi-round communication of multi-player set-disjointness (e.g., [28, 3, 13, 22, 16]) and these bounds achieve lower-bounds on the product of the number of passes and the amount of space. Hence, lower bounds from set-disjointness only exhibit space bounds that scale with the reciprocal of the number of passes. An approach that yields more interesting space/pass trade-offs is to consider pointer jumping problems (e.g., [25, 14, 18]) and related problems such

[1] We consider input to be on a read-only tape rather than a read-write tape as in, e.g., [4, 11].

as the greater-than problem (e.g., [23, 29, 9]). There are two issues with reductions from pointer-jumping and greater-than: the difficulty in achieving tight results in the data-stream model (rather than a communication model) and applying these techniques to order-dependent functions in the data-stream model. To elaborate upon these issues, it is necessary to review the round-elimination technique that underlies many communication lower bounds [7, 1, 5, 27, 26]. In round-elimination we wish to evaluate a function $f(x, y)$ where x is known to Alice and y to Bob. A "meta-problem" $P_f(x_1, \ldots, x_t, i, y) = f(x_i, y)$ is defined where $x_1, \ldots, x_t$ is known to Alice and i, y is known to Bob. It was shown that the existence of a "good" k round protocol for P_f with Alice starting, implied that there exists a "good" $k-1$ round protocol for f where Bob communicates first. The result holds true even if public coins are allowed and Bob is given $x_1, \ldots, x_{i-1}$. Essentially, the good protocol for P_f, stripped of its "not-so-useful" first round, gives a good protocol for f. If P_f is a "self-reducible" problem, i.e., P_f and f can be thought of as the same problem on different size inputs, this gives a beautiful inductive way of bounding the communication of k-round protocols for f given a lower bound for 1 round protocols.

For example, the approach implies that if Alice and Bob have n-bit binary strings x and y respectively and are permitted to communicate a total of r messages, then to determine if $x < y$ requires $\tilde{\Omega}(n^{1/r})$ bits of communication. However, if we consider the same problem in the stream setting we deduce that any p-pass algorithm requires $\tilde{\Omega}(n^{1/(2p-1)})$ space because p passes over the data corresponds to $2p-1$ messages. We show that this is not tight and that a $\tilde{\Omega}(n^{1/p})$ lower-bound exists. Note that if $p = 1$, $2p - 1 = p$ and hence tight one-pass lower bounds are sometimes achieved.

The second issue relates to proving lower bounds for evaluating functions, such as finding the longest increasing subsequence, which are dependent on the ordering of the stream. Here the round elimination lemma has a fundamental problem: the fact that x_i and y must interact (because f encodes order), implies that x_i, x_j interacts (through y). But the round elimination framework requires independence of x_i, x_j ($i \neq j$) and the framework does not apply as is; while we cannot describe all failed attempts, the inherent difficulty can be seen by trying to prove even a two pass result.

Our Approach and Results: We prove "Pass Elimination" lemmas that allow us to prove results related to round elimination *directly* for data streams rather than the usual two-step process of proving a communication lower bound and then using this to imply a data-stream lower bound. One intuitive (but technically inaccurate) way of viewing this result is to consider round-elimination in which Bob has an index i and no other information. Hence, after the first pass Bob has nothing else to say if he reveals i. This will solve the problem of doubling the rounds and avoiding the issue of the interactions. However, in the communication complexity framework, evaluating $f(x_i)$ is trivial once i is known since one player knows x_i! However, in the data-stream setting, evaluating $f(x_i)$ remains a meaningful problem. We prove the pass-elimination results using an information theoretic approach similar to that used by Sen [29]. However, we emphasize that the main contribution of our work is understanding and quantifying the exact problem which allows us to a prove a variety of tight lower bounds. While in most cases the stronger model of communication complexity makes it simpler to prove lower bounds, our tighter lower bound are achievable by exploiting the weakness of the data stream

model. We believe that these results provide a natural, direct, and intuitive framework for proving multi-pass lower bounds, which is likely to encourage its use.

We demonstrate the utility of the pass-elimination lemmas by applying them to variety of problems. We consider the resulting bounds interesting in their own right as they are either the first known lower bounds for the specific problem, or they significantly improve previous known bounds (either by establishing better bounds or by being more general, e.g., applying to all algorithms rather than just deterministic algorithms.)

1. *Post-Order Traversal Tree Pointer Jumping:* In Section 2, we consider the data-stream problem of tree pointer jumping in which the pointer values for each node are revealed according to a post-order traversal of the tree. We prove that any p-pass algorithm for the $(p+1)$-level, t-ary problem requires $\Omega(t/2^p)$ space. Note that this really is a data-stream bound in the sense that the only natural way to conceptualize the problem as a communication problem is if there is a player for each of the $O(t^p)$ nodes in the tree. Consequently, the usual approach of bounding maximum message length (or space in the data-stream setting) by averaging the total communication over each message sent fails. A main achievement in [15] was circumventing such an argument but this was only achieved in the deterministic setting.
2. *Longest Increasing Subsequence:* In Section 3, we prove that any p-pass (randomized) algorithm that finds the elements of the longest increasing subsequence requires $\tilde{\Omega}(k^{1+1/(2^p-1)})$ space where k is the length of the longest increasing subsequence. This matches the upper bound of [22] and resolves an open question of [30]. This doubly-exponential space/pass trade-off is unusual in the data-stream literature but we expect it may arise for other problem for which the natural way to attack the problem is by dynamic programming, e.g., for time-series histograms [17].
3. *Promise Minimum Missing Element and Greater-Than:* The greater-than (GT) problem is to determine which of two integers, $x, y \in 2^n$, is larger given a length $2n$ stream whose elements are the bits of the binary expansion of x and y (in some order.) In Section 4, we show that any p-pass (randomized) algorithm for this problem requires $\tilde{\Omega}(n^{1/p})$ space. Our result also applies to the related problem of finding the minimum missing element (MME) of a stream where all elements smaller than this element occur with multiplicity 1.[2] The best previous bound was $\tilde{\Omega}(n^{1/(2p-1)})$ [9].
4. *Convex Hulls and Fixed-Dimensional Linear Programming:* In Section 5, we show that any p-pass algorithm for fixed-dimensional linear programming requires $\tilde{\Omega}(n^{1/p})$ space where the stream consists of n constraints and the objective function is known. In the full version, we also show bounds for p-pass algorithms for constructing the convex hull of a stream of n points in $\mathbb{R}^2$. If the points are sorted on their x-coordinates we show that $\tilde{\Omega}(\sqrt{n})$ space is required for $p = O(1)$. These bounds generalize previous bounds for a restricted class of deterministic algorithms [8].

Important Note on the Model: It will be convenient to consider a more powerful variant of the usual space-bounded data-stream model. Specifically, we allow algorithms to do an unbounded amount of work, using an unbounded amount of space between the arrival of each data element. We only insist that the amount of space in use when the

[2] Without the promise, MME is related to 2-party set disjointness and stronger bounds exist.

next element arrives satisfies the appropriate bound. Of course, lower bounds for this more powerful model also apply to the usual model. Also, throughout we will assume that the number of passes is constant although many of the results generalize.

2 Pass Elimination (When the First Pass Is Passé...)

In this section we present two new general lower-bounds for multi-pass algorithms in the data-stream model. These results are established by taking ideas from the round-elimination results in communication complexity and applying them directly to data streams. We start by defining the following "meta problems."

Definition 1. *For a problem f defined on a set of streams $\mathcal{X}$, we define:*

1. *$P_{t,f,g}$: Given a stream $\langle x_1, \ldots, x_t, i, g(x_1, x_2, \ldots, x_{i-1})\rangle \in \mathcal{X}^t \times [t] \times \mathcal{X}$, solve $f(x_i)$ where g is an arbitrary function $g : \mathcal{X}^{i-1} \to \mathcal{X}$.*
2. *$Q_{t,f}$: Given a stream $\langle x_1, \ldots, x_t\rangle \in \mathcal{X}^t$, output $\{\langle i, f(x_i)\rangle : i \in [t]\}$ in any order. We do not require the output to be stored in working memory.*[3]

We will prove the following lemma in Section 2.1.

Lemma 1 (Pass Elimination for $P_{t,f,g}$). *Assume $t \geq 50000s$. If there exists a k-pass, s-space algorithm for $P_{t,f,g}$ with prob. at least $1-\delta$, then there exists a $(k-1)$-pass, $2s\log(\delta^{-1})$-space algorithm for f with prob. at least $1-\delta$.*

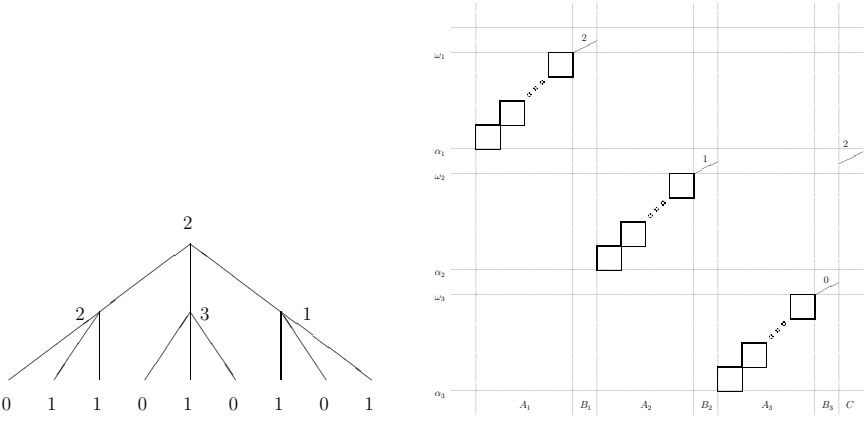
It is reasonable to ask if it is possible to prove a version of the above lemma whether the space requirement remains the same while the error probability only increases additively. This was the case in the communication setting [29]. Unfortunately in the data-stream setting the state must be encoded in the current memory whereas, in the communication setting, the state is determined by all messages that have been sent (note that 2-player communication can be trivially assumed to be in the blackboard model.) The proof of Lemma 2 is similar to that of Lemma 1 and can be found in the full version.

Lemma 2 (Pass Elimination for $Q_{t,f}$). *Assume $t \geq 2s\delta^{-1}$. If there exists a k-pass, s-space algorithm for $Q_{t,f}$ with prob. at least $1-\delta$, then there exists a $(k-1)$-pass, $2s$-space algorithm for $Q_{t/2,f}$ with prob. at least $1-10\delta$.*

Note that the algorithm for f whose existence is proven in Lemma 2 does not succeed with the same probability as $Q_{t,f}$. Unfortunately this can not be fixed by parallel repetition because the algorithm for $Q_{t,f}$ may write the solution to a write-only tape.

Post-Order-Traversal: We now define a data-stream problem related to pointer-chasing. The significant difference in our work is the order in which we encode a pointer-chasing instance in the data-stream. This will be essential in proving lower-bounds for order-dependent functions such as finding the elements of the longest increasing subsequence.

[3] Note that $Q_{t,f}$ can be solved in one pass if there is sufficient memory to solve $f(x)$ in one pass as the $f(x_i)$ can be computed sequentially. However, if $f(x)$ can be solved in much less space given multiple passes, it is not clear if $Q_{t,f}$ can be solved in less space.



(a) 3-level, 3-ary tree traversal. (b) LIS construction.

Fig. 1. Part (a) shows an instance of $(p+1)$-level, t-ary tree traversal with $t = 3$, $p = 2$, and $g^3(v_{\text{root}}) = 0$. The POT stream is $\langle 0,1,1,2,0,1,0,3,1,0,1,1,2\rangle$. The SPOT stream is $\langle 0,1,1,2,0,0,1,0,3,0,1,1,0,1,1,2,2,0,1,1\rangle$. Part (b) is referenced in Section 3.

Definition 2. *Consider a $(p+1)$-level, t-ary tree T rooted at v_{root} and a function $f : V(T) \to [t]$ where $f(v) \in \{0,1\}$ if v is a leaf of T. Define $g(v)$ to be the $f(v)$-th child of v if v is an internal node and $f(v)$ if v is a leaf. The* POT *problem to evaluate* $\text{POT}(f) = g^{p+1}(v_{root})$ *given the stream $\langle f(v_{\sigma(1)}), f(v_{\sigma(2)}), \ldots, f(v_{\sigma(|T|)})\rangle$, where σ is a post-order-traversal of $V(T)$.*

The next theorem follows by induction from Lemma 1.

Theorem 1. *Any p-pass algorithm that solves the $(p+1)$-level, t-ary* POT *problem (w/p. $9/10$) requires $\Omega(t/2^p)$ space.*

Strong-Post-Order-Traversal: A strong-post-order-traversal of a tree is a variant of the post-order-traversal in which a partial pre-order traversal is made between the visit of each node on the post-order-traversal. While this may seem like a very artificial construction, it will be very important to some of our other lower bounds.

Definition 3. *Consider a $(p+1)$-level, t-ary tree T rooted at v_{root} and a function $f : V(T) \to [t]$ where $f(v) \in \{0,1\}$ if v is a leaf of T. Define $g(v)$ to be the $f(v)$-th child of v. The* SPOT *problem is to evaluate* $\text{SPOT}(f) = g^{p+1}(v_{root})$ *given the stream $\langle S(v_{\sigma(1)}), S(v_{\sigma(2)}), \ldots, S(v_{\sigma(|T|)})\rangle$, where σ is a post-order-traversal of $V(T)$ and $S(v)$ is defined as follows:*

$$S(v) = \begin{cases} \langle f(v)\rangle & \text{if } v \text{ is a leaf} \\ \langle f(v), R(u_1), \ldots, R(u_{f(v)-1})\rangle & \text{if } v \text{ has children } u_1, \ldots, u_t \end{cases}$$

where $R(u)$ is a pre-order traversal of the nodes in the sub-tree rooted at u.

The next theorem follows by induction from Lemma 1.

Theorem 2. *Solving the $(p+1)$-level t-ary* SPOT *problem (w/p. 9/10) in p passes requires $\Omega(t/2^p)$ space.*

The next theorem follows from Lemma 2 by induction.

Theorem 3. *If solving $f(x)$ in one pass (w/p. $9/10$) requires s space then any p-pass algorithm for $Q_{t,f}$ (w/p. $1-1/10^p$) requires $\Omega(s)$-space if $t > s(20)^{(p-1)p/2}$.*

2.1 Proof of Lemma 1

Consider an arbitrary distribution D over $\mathcal{X}$. By assumption and Yao's lemma there exists a deterministic k-pass s-space algorithm $\mathcal{A}$ that, with probability at least $1-\delta$, solves $P_{t,f,g}$ correctly on the stream $\langle X_1, \ldots, X_t, i, g(X_1, \ldots, X_{i-1})\rangle$ where each $X_j \sim D$ are independent and $i \in_R [t]$. We will show that such an algorithm can be used to construct a deterministic $(k-1)$-pass $2s$-space algorithm that solves f on $\langle X_i\rangle$ with probability at least $9/10$. Since D was arbitrary, by Yao's lemma, this shows that there is a randomized algorithm that solves $P_{t,f}$ with probability at least $4/5$ over an arbitrary input. Running $O(\log \delta^{-1})$ copies in parallel and taking the median value results in an algorithm that is successful with probability at least $1-\delta$.

First we introduce some further notation. Let the memory states of $\mathcal{A}$ be $\mathcal{M} = \{m_1, \ldots, m_{2^s}\}$ where m_1 is the initial memory state. Let $\mathcal{A}(S; m_u)$ denote the memory state of $\mathcal{A}$ after being initialized with memory state m_u and reading stream S. If $m_u = m_1$ we omit it. Let $\mathcal{C}$ be the set of inputs upon which $\mathcal{A}$ is successful. By assumption, $\Pr[\langle X_1, \ldots, X_t, r, g(X_1, \ldots, X_{r-1})\rangle \in \mathcal{C}] \geq 1-\delta$.

Lemma 3. *There exists $r \in [t]$ and $\langle x_1, x_2, \ldots, x_{r-1}\rangle \in \mathcal{X}^{r-1}$ such that,*

$$I(M^r; M^t) \leq \frac{9s}{t} \text{ and } \Pr[\langle x_1, \cdots x_{r-1}, X_r, \ldots, X_t, r, g(x_1, \ldots, x_{r-1})\rangle \in \mathcal{C}] \geq 1-9\delta$$

where $M^r = \mathcal{A}(x_1, \ldots, x_{r-1}, X_r)$, $M^t = \mathcal{A}(x_1, \ldots, x_{r-1}, X_r, \ldots, X_t)$, and $I(\cdot;\cdot)$ is the mutual information.

See the full version for the proof of the above lemma and the rest of the lemmas in this section. Next we consider the distribution of the memory states of the algorithm after processing various prefixes of the stream. Define the distributions, $p_{u,v} = \Pr[M^r = m_u, M^t = m_v]$, $p_u = \Pr[M^r = m_u]$, $q_v = \Pr[M^t = m_v]$, and $p^v_u = p_{u,v}/q_v$. Note that p^v is the distribution M^r conditioned on the event that $\{M^t = v\}$. Intuitively, if the mutual information between M^r and M^t is low then p^v is similar to p. The next lemma, a variant of the Average Encoding Theorem [21], substantiates this.

Lemma 4. *If v is chosen with probability q_v then the variational distance between the distribution of M^r and M^r conditioned on $\{M^t = v\}$ is small if $I(M^r; M^t)$ is small. Specifically,* $\Pr_{v\sim q_v}\left[\sum_u |p_u - p^v_u| \leq 2\sqrt{3I(M^r; M^t)}\right] \geq 2/3$.

The next lemma shows that there exists a state m_v such that we may choose a continuation of the stream $\langle x_1, \ldots, x_{r-1}, X_r\rangle$ that a) is only a function of M^r, b) yields a stream upon which the algorithm is likely to succeed, and c) guarantees $M^t = m_v$.

Lemma 5. *There exists* $m_v \in \mathcal{M}$ *and* $Z : \mathcal{M} \to \mathcal{X}^{t-r}$ *such that* $\mathcal{A}(Z(m_u); m_u) = m_v$ *and* $\Pr[\langle x_1, \ldots, x_{r-1}, X_r, Z(M^r), r, g(x_1, \ldots, x_{r-1})\rangle \in \mathcal{C}] \geq 9/10$, *if* $\delta < 1/1000$ *and* $s < 50000t$.

Let $m_w = \mathcal{A}(r, g(x_1, \ldots, x_{r-1}); m_v)$. We construct a $(k-1)$-pass algorithm $\mathcal{A}'$ by running $\mathcal{A}$ on the stream $\langle x_1, \ldots, x_{r-1}, X_r, Z(M^r), r, g(x_1, \ldots, x_{r-1})\rangle$ and essentially collapsing the first two passes of $\mathcal{A}$ into a single pass. We compute $m_u = \mathcal{A}(x_1, x_2, \ldots, x_{r-1}, X_r)$ and $m_x = \mathcal{A}(x_1, x_2, \ldots, x_{r-1}, X_r; m_w)$ in parallel using a single pass. We store M^r for the rest of the algorithm and use it to construct $Z(m_u)$ when necessary. Note that if $Z(\cdot)$ is not defined on m_u, we may output an arbitrary value. We then compute the first pass of $\mathcal{A}'$ by computing, $\mathcal{A}(Z(m_u), r, g(x_1, \ldots, x_{r-1}); m_x)$. The remaining $k-2$ passes of $\mathcal{A}'$ emulate the remaining $k-2$ passes of $\mathcal{A}$ where we generate $Z(m_u)$ from the stored value of m_u when necessary.

3 Longest Increasing Subsequence

An instance of the longest increasing subsequence (LIS) problem is, given a stream $S = \langle x_1, \ldots, x_m\rangle \in [n]^m$, find the elements of a sub-stream of increasing values $\langle x_{i_1}, \ldots, x_{i_j}\rangle$ of maximum length. In a promise variant, the PLIS problem, we are promised $|\textsc{lis}(S)| \leq k$. The best algorithmic result for this problem is due to Liben-Nowell et al. [22]: there exists a deterministic algorithm for PLIS that uses p passes and $\tilde{O}(k^{1+1/(2^p-1)})$ space. Sun and Woodruff [30] established that Liben-Nowell et al.'s algorithm was essentially optimal for one-pass algorithms. In particular they showed that any one-pass algorithm for PLIS (with prob. $9/10$) requires $\Omega(k^2)$ space. It was left as an open question whether the doubly exponential trade-off between space and passes in Liben-Nowell et al.'s result was the true trade-off. We show that this is the case and present a matching lower-bound for PLIS by a reduction to the POT problem.

Theorem 4. *Solving* PLIS *(w/p.* $9/10$*) in* p *passes requires* $\Omega(k^{1+\frac{1}{2^p-1}})$ *space.*

Proof. Let $t = m^{1/p}$. We show a reduction from the $(p+1)$-level t-ary POT problem to computing the elements of a PLIS problem over universe $[2^p mt]$ where the LIS is promised to have length $k = \Theta(t^{1-1/2^p})$. Let f be an instance of $(p+1)$-level t-ary POT. Let $h_1 = 1$ and then recursively define

$$r_j := \left\lceil \sqrt{t/h_{j-1}} \right\rceil, \; t_j := \lceil t/r_j \rceil, \text{ and } h_j := r_j h_{j-1} + t_j \; .$$

By induction it can be shown that $t^{1-1/2^j}/6 \leq h_{j+1} < 6t^{1-1/2^j}$.

With each node v we associate a sequence of numbers, $I(v)$. If v is leaf then $I(v) = (f(v))$. If v is an internal node we will define $I(v)$ inductively such that if v is at level j, $\textsc{lis}(I(v)) = h_j$. Let children$(v) = \{u_0, \ldots, u_{t-1}\}$. We group the children$(v)$ into t_j sets $S_i = \{u_{(i-1)r_j+\ell} : \ell = 0, \ldots, r_j - 1\}$. Note that $|S_1| = \ldots = |S_{t_j-1}| = r_j$ but $|S_{t_j}| = t - (t_j - 1)r$ may be smaller. The $I(v)$ we construct will have the following form: $I(v) = (A_1 : B_1 : \ldots : A_{t_j} : B_{t_j} : C)$ where ":" denotes concatenation. A_i will be constructed from $I(v)$ for $v \in S_i$ and B_i will be an increasing sequence. C will be an increasing sequence of length $a := \lceil f(v)/r_j \rceil$. Together A_i and B_i and C will satisfy:

1. $\text{LIS}(A_i : B_i) = r_j h_{j-1} + t_j - i$ and $\max(A_i : B_i) < \min(A_{i-1} : B_{i-1})$
2. $\text{LIS}(A_a : B_a : C) = h_j$
3. $\max(A_a : B_a) < \min(C) < \max(C) < \min(A_{a-1} : B_{a-1})$

It follows that the elements of the longest increasing subsequence of $I(v)$ include the elements of the longest increasing subsequence of A_a. See Figure 1(b).

Constructing A_i, B_i and C: We construct A_i and B_i as follows. First, let $R_{j-1} \geq \max(I(u_0), \ldots, I(u_{t-1}))$. Let $I_1(u_{ir_j+\ell})$ be the sequence formed by adding ℓR_{j-1} to each element of $I(u_{ir_j+\ell})$. Therefore for all i, $\max(I_1(u_{ir_j+\ell})) \leq |S_{i+1}|R_{j-1}$ and

$$\text{LIS}(I_1(u_{ir_j}) : \ldots : I_1(u_{ir_j+|S_{i+1}|-1})) = |S_{i+1}|h_{j-1} \, .$$

Let $I_2(u_{ir_j+\ell})$ be the sequence formed by adding $(t_j - i)(r_j R_{j-1} + t_j)$ to each element of $I_1(u_{ir_j+\ell})$. We now define $A_i = (I_2(u_{ir_j}) : \ldots : I_2(u_{ir_j+|S_i|-1}))$. For $i \in [t_j - 1]$,

$$\begin{aligned} \min(A_i) &\geq (t_j - i)(r_j R_{j-1} + t_j) =: \alpha_i \\ \max(A_{i+1}) &\leq r_j R_{j-1} + (t_j - 1 - i)(r_j R_{j-1} + t_j) =: \omega_{i+1} \, . \end{aligned}$$

and therefore $\alpha_i - \omega_{i+1} \geq t_j$. Hence, letting

$$B_i = (\omega_i + 1, \omega_i + 2, \ldots, \omega_i + (r_j h_{j-1} + t_j - i - |S_{i+1}|h_{j-1}))$$

ensures $\text{LIS}(A_i : B_i) = r_j h_{j-1} + t_j - i$ and $\max(A_i : B_i) < \min(A_{i-1} : B_{i-1})$. Letting C be the sequence $C = (\alpha_{a-1} - a, \ldots, \alpha_{a-1} - 1)$ ensures all the necessary properties. One issue remains: an element of the longest increasing subsequence encodes the answer to the POT problem but it is not clear which one because the subsequence only encodes $\lceil f(v)/r_j \rceil$ rather than $f(v)$. However, this can be fixed by encoding $f(v)$ in the lower order bits of C with a factor t increase in universe size.

4 Promise Min. Missing Element and Greater-Than

In this section, we consider the related problems of MME and GT. An instance of MME consists of n non-negative integers A and the goal to identify $\text{MME}(A) = \min\{i \in \mathbb{N}_0 : i \notin A\}$. There exists an $\Omega(n/p)$ space lower-bound for any p-pass algorithm that solves this problem that can be proved using a reduction from SET-DISJOINTNESS. We are interested in a promise version of the problem in which we assume that all elements less than $\text{MME}(A)$ occur only once in A. This changes the complexity of the problem considerably.

An instance of the GT consists of two n-bit strings $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_n$ and the goal is to determine if $x < y$, i.e., if there exists $i \in [n]$ such that $x_i < y_i$ and $x_j = y_j$ for all $j < i$. In the data-stream setting we assume that the stream consists of the elements $\{(x_i, i), (y_i, i) : i \in [n]\}$ in some arbitrary order. Note that MME and GT are related. For example, if $A = \{2(i-1) + x_i, 2(i-1) + 1 - y_i : i \in [n]\}$ then the parity of $\text{MME}(A)$ is odd iff $x < y$. Note that a stream of such elements can be transduced from a stream of elements from $\{(x_i, i), (y_i, i) : i \in [n]\}$ in constant space. Hence any lower-bound for MME also yields a lower-bound for GT.

Both problems were considered by Chang and Kannan [9] for the purposes of proving lower bounds on the "generalized histogram learning problem" (see [9] for details). Using the round-elimination lemma and results from [23,29] they proved a space lower bound of $\Omega(n^{1/(2p-1)})$ for any constant p-pass algorithm that solved either GT or MME. We improve this by a polynomial factor improvement when $p > 1$.

Theorem 5. *Solving* GT *or* MME *(w/p.* $9/10$*) in* p *passes requires* $\Omega(n^{1/p})$ *space.*

Proof. We reduce SPOT to MME. Let the function f be an instance of SPOT and let $\{u_1, \ldots, u_{t^p}\}$ be the leaves of a $(p+1)$-level, t-ary tree. With each leaf u_i we associate $c(u_i) = 2(i-1) + f(u_i)$ and $d(u_i) = 2(i-1) + 1 - f(u_i)$.

The idea behind the reduction is that if $g^p(v_{root}) = u_j$ then the smallest $2j-1$ elements of the stream will be $\{c(u_i), d(u_i) : i < j\} \cup \{c(u_j)\}$ and the stream will not contain $d(u_j)$. Consequently $d(u_j)$ will be the minimum missing element and hence, $1 - (d(u_j) \bmod 2) = g^{p+1}(v_{root})$.

For a leaf u, the reduction replaces $f(u)$ by $c(u)$ the first time the leaf is visited in the strong post-order traversal. If u is visited a second time then we replace $f(u)$ by $d(u)$ on this second visit. On all subsequent visits to u, $f(u)$ is ignored. For example, the SPOT instance in Figure 1(a) would become $\langle 0, 3, 5, 1, 6, 9, 10, 7, 8, 13, 14, 17, 2, 4\rangle$. Note that the minimum missing element is 11 and $f(g^p(v_{root})) = 1 - (11 \bmod 2) = 0$.

We need to show that this reduction can be performed in a streaming fashion. In particular, we need to establish that it is possible to determine if a node in the strong post-order traversal has already been visited once or at least twice. We can recognize the first time we visit a leaf because the first visit to u_i is before the first visit to u_j if $i < j$. Hence, it suffices to remember the smallest leaf that has not been visited.

Leaf u is only revisited during a pre-order traversal of a sub-tree containing u. Consider a pre-order traversal of the j-level, t-ary sub-tree that contains u. Let v_j be the root of this sub-tree and let $v_j, v_{j-1}, \ldots, v_1$ be the path from root to leaf $v_1 = u$. Let v_{i-1} be the a_i-th child of v_i. Then u is being revisited for the first time if $a_i < f(v_i)$ for all $i = 2, \ldots, j$. Note that the set of relevant values of f can be maintained in $O(p)$ space because the sub-tree is visited in pre-order.

5 Fixed-Dimensional Linear Programming

In this section, we consider the problem of linear programming in $\mathbb{R}^d$ where d is assumed constant. An instance of this problem consists of an objective function known to the algorithm and a set of n constraints each of which are specified by an element of the stream. In a recent paper, Chan and Chen [8] showed that any p-pass deterministic algorithm "of a certain type" for finding the lowest point in the intersection of n upper half planes in $2D$ requires $\Omega(n^{1/p})$ space. The algorithms considered were for the decision tree model where "the only allowable operations on the input half-planes are testing the sign of a function evaluated at the coefficients of a subset of half-planes currently in memory." While these test functions could be any continuous function, the algorithm is restricted to storing points of the input only.

We show that the $\Omega(n^{1/p})$ bound holds in $3D$ for any algorithm, e.g., for algorithms that may be randomized or may store information other than the points themselves. For $2D$, the same bound holds if we allow one non-linear constraint.

Theorem 6. *Solving* 3DLP *(w/p.* $9/10$*) in* $p = O(1)$ *passes requires* $\Omega(n^{1/p})$ *space.*

Proof. Let $t = n^{1/p}$. We first reduce $(p+1)$-level t-ary POT to (the feasibility of) 2D linear programming with the added non-linear constraint that $x^2 + y^2 = 1$. We then show how to remove this constraint using one extra dimension. The f be an instance of $(p+1)$-level t-ary POT. We consider constraints specified by chords on the bottom half of the unit circle. It will be convenient to use polar representation of points and represent every half-space by a chord through a pair of points. The reduction is as follows:

1. For each node $v \in V(T)$ define two "end-points" s_v and t_v. If v is the root of T let $s_v = (-1, 0)$ and $t_v = (1, 0)$. For any other node v, let v be the i-th child of node w. Consider partitioning the arc from s_w to t_w into t equi-length arcs and define s_v and t_v to be the end points of the i-th such arc. For each leaf, also define a "mid-point" m_v that is halfway along the arc from s_v to t_v.
2. For each internal node $v \in V(T)$: Add chords through s_v and $s_{u_{f(v)}}$ and through $t_{u_{f(v)}}$ and t_v where $\{u_i : i \in [t]\}$ are the children of v.
3. For each leaf $v \in V(T)$: If $f(v) = 0$, add the chord through s_v and t_v.
4. Lastly, add "dead-zone" constraints. Let $\{u_1, \ldots, u_{t^p}\}$ be the leaves of the tree. For $i \in [t^p]$, add the chord between m_{u_i} and $m_{u_{i+1}}$ where $m_{u_{m^p+1}} = m_{u_1}$

The constraint at an internal node v ensures that any feasible points lies on the arc (s_v, t_v). The constraints at a leaf node v determine if there exists a feasible point on the arc (s_v, t_v) given that the dead-zone constraints ensure that any feasible point is a mid-point. Consequently, there exists a feasible point on the unit circle iff $\text{POT}(f) = 1$. It is immediate that the number of constraints is $O(n)$ and the smallest angle (at the origin) we construct in this process is $\Theta(1/n)$. Thus in a polar format the input is polynomially bounded. To remove the constraint $x^2 + y^2 = 1$, consider lifting the problem to the cone $z = x^2 + y^2$, and add the constraint $z = 1$. While the constraint $z = x^2 + y^2$ is not linear we observe that we are only interested in $O(n)$ predetermined points on the cone. Thus we can approximate the cone by $O(n)$ planes.

Acknowledgments. Thanks to Kook Jin Ahn and Matei David for helpful comments.

References

1. Adler, M., Demaine, E.D., Harvey, N.J.A., Patrascu, M.: Lower bounds for asymmetric communication channels and distributed source coding. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 251–260 (2006)
2. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. Journal of Computer and System Sciences 58(1), 137–147 (1999)
3. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. In: IEEE Symposium on Foundations of Computer Science, pp. 209–218 (2002)
4. Beame, P., Jayram, T.S., Rudra, A.: Lower bounds for randomized read/write stream algorithms. In: ACM Symposium on Theory of Computing, pp. 689–698 (2007)
5. Chakrabarti, A.: Lower bounds for multi-player pointer jumping. In: IEEE Conference on Computational Complexity, pp. 33–45 (2007)

6. 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)
7. Chakrabarti, A., Regev, O.: An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In: IEEE Symposium on Foundations of Computer Science, pp. 473–482 (2004)
8. Chan, T.M., Chen, E.Y.: Multi-pass geometric algorithms. Discrete & Computational Geometry 37(1), 79–102 (2007)
9. Chang, K.L., Kannan, R.: The space complexity of pass-efficient algorithms for clustering. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 1157–1166 (2006)
10. Chu, M., Kannan, S., McGregor, A.: Checking and spot-checking of heaps. In: International Colloquium on Automata, Languages and Programming, pp. 728–739 (2007)
11. Demetrescu, C., Finocchi, I., Ribichini, A.: Trading off space for passes in graph streaming problems. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 714–723 (2006)
12. Ergun, F., Jowhari, H.: On the distance to monotonicity and longest increasing subsequence of a data stream. In: ACM-SIAM Symposium on Discrete Algorithms (2008)
13. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: Graph distances in the streaming model: the value of space. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 745–754 (2005)
14. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. Theoretical Computer Science 348(2-3), 207–216 (2005)
15. Gal, A., Gopalan, P.: Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In: IEEE Symposium on Foundations of Computer Science (2007)
16. Gopalan, P., Jayram, T., Krauthgamer, R., Kumar, R.: Estimating the sortedness of a data stream. In: ACM-SIAM Symposium on Discrete Algorithms (2007)
17. Guha, S., Koudas, N., Shim, K.: Approximation and streaming algorithms for histogram construction problems. ACM Trans. Database Syst. 31(1), 396–438 (2006)
18. Guha, S., McGregor, A.: Lower bounds for quantile estimation in random-order and multi-pass streaming. In: International Colloquium on Automata, Languages and Programming, pp. 704–715 (2007)
19. Henzinger, M.R., Raghavan, P., Rajagopalan, S.: Computing on data streams. In: External memory algorithms, pp. 107–118 (1999)
20. Indyk, P.: Stable distributions, pseudorandom generators, embeddings, and data stream computation. J. ACM 53(3), 307–323 (2006)
21. Klauck, H., Nayak, A., Ta-Shma, A., Zuckerman, D.: Interaction in quantum communication and the complexity of set disjointness. In: ACM Symposium on Theory of Computing, pp. 124–133 (2001)
22. Liben-Nowell, D., Vee, E., Zhu, A.: Finding longest increasing and common subsequences in streaming data. J. Comb. Optim. 11(2), 155–175 (2006)
23. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. J. Comput. Syst. Sci. 57(1), 37–49 (1998)
24. Munro, J.I., Paterson, M.: Selection and sorting with limited storage. Theor. Comput. Sci. 12, 315–323 (1980)
25. Nisan, N., Wigderson, A.: Rounds in communication complexity revisited. SIAM J. Comput. 22(1), 211–219 (1993)
26. Patrascu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: ACM Symposium on Theory of Computing, pp. 232–240 (2006)
27. Patrascu, M., Thorup, M.: Randomization does not help searching predecessors. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 555–564 (2007)

28. Razborov, A.A.: On the distributional complexity of disjointness. Theor. Comput. Sci. 106(2), 385–390 (1992)
29. Sen, P.: Lower bounds for predecessor searching in the cell probe model. In: IEEE Conference on Computational Complexity, pp. 73–83 (2003)
30. Sun, X., Woodruff, D.: The communication and streaming complexity of computing the longest common and increasing subsequences. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 336–345 (2007)

Impossibility of a Quantum Speed-Up with a Faulty Oracle

Oded Regev* and Liron Schiff**

School of Computer Science
Tel-Aviv University

Abstract. We consider Grover's unstructured search problem in the setting where each oracle call has some small probability of failing. We show that no quantum speed-up is possible in this case.

1 Introduction

Unstructured search problem: The unstructured search problem, also known as the unordered search problem or as Grover's search problem, is the most basic problem in the query model. The goal is to find a marked entry out of N possible entries. In this model the entries are accessible only through a black box (the oracle), and the complexity of the algorithm is measured in terms of the number of oracle queries. In the classical world, it is easy to see that solving this search problem requires $\Theta(N)$ queries, even if we allow randomization. In the quantum world, however, one can find a marked item with only $O(\sqrt{N})$ queries, as was shown in Grover's seminal paper [1]. Moreover, it is known that this is optimal (see, e.g., [2,3,4]). This remarkable quadratic improvement is considered one of the biggest successes of quantum computing, and has sparked a huge interest in the quantum query model (see [5] for a recent survey).

Searching with a faulty oracle: In this paper we consider the unstructured search problem in the *faulty oracle model*, a question originally presented to us by Harrow [6]. In this model, each oracle call succeeds with some probability $1-p$, and with the remaining probability p the state given to the oracle remains unchanged. More formally, each oracle call maps an input state ρ into $(1-p)\cdot O\rho O^\dagger + p\cdot\rho$ where O is the original (unitary) oracle operation. We note that this model can be seen to be equivalent to other, seemingly more realistic, models of faults, such as the model considered in Shenvi et al. [7] in which the oracle's operation is subject to small random phase fluctuations.

Our motivation for considering the faulty oracle model is twofold. First, we believe that since the unstructured search problem is such a basic question,

* School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Supported by the Binational Science Foundation, by the Israel Science Foundation, and by the European Commission under the Integrated Project QAP funded by the IST directorate as Contract Number 015848.

** School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 773–781, 2008.

it is theoretically interesting to consider it in different settings, as this might shed more light on the strengths and weaknesses of quantum query algorithms. A second motivation is related to implementation aspects of quantum query algorithms, as one can expect any future implementation of a Grover oracle to be imperfect (see [7] for a further discussion of the physical significance of the model).

To motivate our main result and to get some intuition for the model, let us consider the behavior of Grover's original algorithm in this setting. Recall that Grover's algorithm can be seen as a sequence of two alternating reflections, $OUOUOU \cdots OU$ where U is the reflection given by Grover's algorithm and O is the reflection representing the oracle call. In the analysis of Grover's algorithm, one observes that the state of the system is restricted to a two-dimensional subspace, inside which lie the initial state and the target state. The angle between these two states is essentially $\pi/2$. Furthermore, the combined operation OU of two consecutive reflections can be seen a rotation by an angle of essentially $1/\sqrt{N}$ inside this two dimensional subspace. Hence the total number of oracle calls required to get to the target state is $O(\sqrt{N})$.

In the faulty oracle model, each oracle call O has some constant probability of not doing anything. Hence, the sequence of reflections might look like $OUOUOUUOUOUOUO$. The effect of this is that after a sequence of rotations OU by $1/\sqrt{N}$, we instead obtain a sequence of rotations $UO = (OU)^\dagger$ by $-1/\sqrt{N}$ which cancel the previous ones. The cancellation can also be seen by noting that $U^2 = O^2 = I$. The end result is that instead of rotating towards the target, our rotation behaves like a random walk, alternating between steps of $1/\sqrt{N}$ and steps of $-1/\sqrt{N}$. Using known properties of random walks on a line, the number of steps required for this walk to reach the target is $\Theta(N)$, which shows that Grover's algorithm is no better than the naive classical search algorithm.

But can there be another, more sophisticated algorithm that copes better with the faults? Our main result shows that the answer is essentially 'no'.

Our result: Our main result shows that there is essentially no quantum advantage when searching with a faulty oracle.

Theorem 1. *Any algorithm that solves the p-faulty Grover problem must use* $T > \frac{p}{10(1-p)}N$ *queries.*

In particular, for any constant $p > 0$, this gives a lower bound of $\Omega(N)$.

Notice that the above statement holds for *any* quantum algorithm, and not just for Grover's algorithm. In particular, it shows that some natural approaches, like fault-tolerant quantum computation [8], cannot help in this setting. Note, however, that this impossibility result applies only in case that the oracle is truly a black-box oracle; if, instead, the oracle is given as a faulty circuit, then fault-tolerant schemes *can* be used to achieve a quantum speed-up by applying them to the circuit obtained by taking Grover's algorithm and replacing the oracle calls with their circuit implementation.

Related work: There has been a considerable amount of work dedicated to analyzing Grover's algorithm in all kinds of faulty settings (see, e.g., [9,7,10]). All these works concentrate on Grover's algorithm (or variants thereof) and none of them give a general statement that applies to all algorithms. In particular, Shenvi et al. [7] analyze the behavior of Grover's algorithm in a physically motivated model that is equivalent to ours. Our result answers the main open question presented in their paper.

There has also been a significant amount of work on searching with an imperfect, but still unitary, oracle (see, e.g., [11,12,13,14,15]). Such oracles are sometimes known as *noisy* oracles. The motivation for this model is algorithmic, and is related to what is known as amplitude amplification. Typically in this case, the quantum speed-up of $O(\sqrt{N})$ is still achievable. Very roughly speaking, this is because a unitary operation (even an imperfect one) is reversible and does not lead to decoherence. There has also been some recent work on analyzing the case of an imperfect unitary implementation of Grover's algorithm (as opposed to an imperfect *oracle*) [16], again showing that a speed-up of $O(\sqrt{N})$ is achievable.

Open problems: One interesting open question is to extend our result to other physically interesting fault models. We believe that our proof technique should be applicable in a more general setting. One natural fault model suggested to us by Nicolas Cerf is the one in which each oracle query has probability p of turning the state into the completely mixed state. Also, is there *any* reasonable fault model for which a quantum speed-up *is* achievable? We suspect that the answer is no.

Another open question is to extend our result to other search problems (see [5] for a recent survey). Is there *any* search problem for which a quantum speed-up is achievable with a faulty oracle? Can one extend our lower bound to a more general lower bound in the spirit of the adversary method (see [4,17])? It is also worth investigating whether the polynomial method [18] can be used to derive lower bounds in the faulty oracle case; our attempts to do so were unsuccessful. We should emphasize, however, that our faulty oracle model is not necessarily so natural for other search problems, and before approaching the above open questions, some thought should be given to the choice of the faulty oracle model.

2 Preliminaries

We assume familiarity with basic notions of quantum computation (see [19]).

Definition 1 (Grover oracle). *For each $k \in \{1, \ldots, N\}$ where N is an integer, the* perfect oracle $\hat{O}^k$ *is the unitary transformation acting on an N-dimensional register that maps $|k\rangle$ to $-|k\rangle$ and $|i\rangle$ to $|i\rangle$ for each $i \neq k$, i.e.,*

$$\hat{O}^k = -|k\rangle\langle k| + \sum_{i \neq k} |i\rangle\langle i|.$$

We also extend the definition to $k = 0$ by defining $\hat{O}^0$ to be the 'null' oracle, given by the identity matrix I.

Definition 2. *The* p-faulty oracle O_p^k *is defined as the operation that with probability* $1 - p$*, acts as the perfect oracle* $\hat{O}^k$ *and otherwise does nothing, i.e., for any density matrix* ρ*,*

$$O_p^k(\rho) = (1-p) \cdot \hat{O}^k \rho \hat{O}^{k\dagger} + p \cdot \rho.$$

We note that instead of our phase-flipping oracle, one could also consider a bit-flipping oracle. Since it is not difficult to construct the latter from the former (see, e.g., [20, Chapter 8]), our lower bound also applies to the bit-flipping case.

Definition 3. *Let* $0 < p < 1$ *be some constant. In the* p-faulty Grover problem*, we are given oracle access to the* p*-faulty oracle* O_p^k *for some unknown* $k \in \{0, \ldots, N\}$ *and our goal is to decide whether* $k = 0$ *or not with success probability at least* $\frac{9}{10}$*.*

Note that the choice of success probability is inconsequential, as one can easily increase it by repeating the algorithm a few times. Also note that we consider here the decision problem, as opposed to the search problem of recovering k from O_p^k. Since we are interested in lower bounds, this makes our result stronger.

3 Proof

We start by giving a brief outline of the proof. For simplicity, we consider the case $p = 1/2$. The proof starts with a simple, yet crucial, observation (Claim 1) which gives an alternative description of the faulty oracle. In the case $p = 1/2$, it says that the oracle O_p^k is essentially performing the two-outcome measurement given by $\{|k\rangle, |k\rangle^\perp\}$. Then, in Lemma 1, we 'approximate' the mixed states that arise during the algorithm with (unnormalized) pure states. This is done by assuming that the measurements done by the oracle all end up in the $|k\rangle^\perp$ subspace. The rest of the proof is similar in structure to previous lower bounds. Using the pure state description, we define a progress measure H_t^k, which is initially zero. We show that at the end of the algorithm it must be high (Lemma 2), and that it cannot increase by too much at each step (Lemma 3). This yields the desired lower bound on the number of queries T. We now proceed with the proof.

Let A be an algorithm for the p-faulty Grover problem on N elements that uses T queries. Assume the algorithm is described by the unitary operations $U_0, U_1, U_2, \ldots, U_T$ acting on an NM-dimensional system, composed of an N-dimensional query register used as oracle input, and an M-dimensional ancillary register. Let $\widetilde{\rho}_0$ denote the initial state of the system, which we assume without loss of generality to be a pure state $\widetilde{\rho}_0 = |\widetilde{\phi}_0\rangle\langle\widetilde{\phi}_0|$. For $k \in \{0, \ldots, N\}$, we let $\widetilde{\rho}_0^k = \widetilde{\rho}_0$, $\rho_0^k = U_0 \widetilde{\rho}_0^k U_0^\dagger$, $\widetilde{\rho}_1^k = O_p^k(\rho_0^k)$, $\rho_1^k = U_1 \widetilde{\rho}_1^k U_1^\dagger, \ldots, \rho_T^k = U_T \widetilde{\rho}_T^k U_T^\dagger$ be the intermediate states of the algorithm when run with oracle O_p^k (see Figure 1). In other words, ρ_t^k is the state of the system right after applying U_t, and $\widetilde{\rho}_{t+1}^k$ is the state of the system right after applying O_p^k on ρ_t^k.

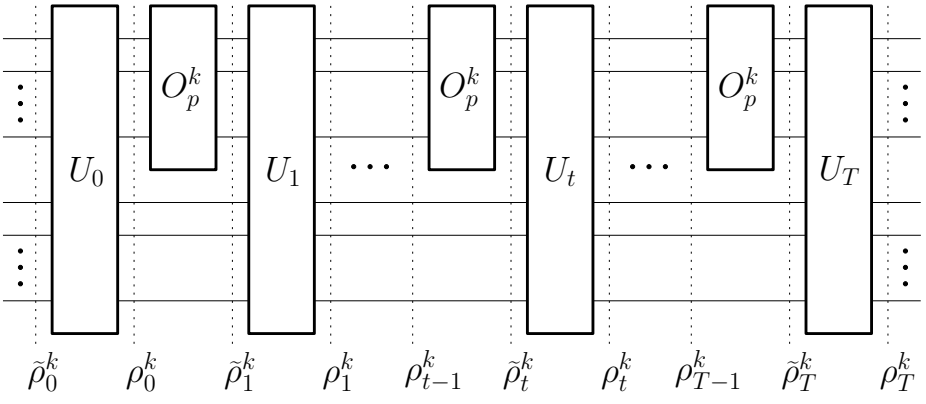


Fig. 1. State evolution

First we show a different way to decompose the outcome of O_p^k.

Claim 1. *Let $|\phi\rangle \in \mathbb{C}^{N \cdot M}$ be an arbitrary vector and let $|\beta_i\rangle \in \mathbb{C}^M$ be such that $|\phi\rangle = \sum_{i=1}^N |i, \beta_i\rangle$. Then*

$$O_p^k(|\phi\rangle\langle\phi|) = |\widetilde{\phi}\rangle\langle\widetilde{\phi}| + 4p(1-p)|k, \beta_k\rangle\langle k, \beta_k|$$

where

$$|\widetilde{\phi}\rangle := \sum_{i=1}^{N} |i, \beta_i\rangle - 2(1-p)|k, \beta_k\rangle.$$

Proof. By Definition 2 we have

$$O_p^k(|\phi\rangle\langle\phi|) = p|\phi\rangle\langle\phi| + (1-p)|\psi\rangle\langle\psi|$$

where $|\psi\rangle = \sum_{i \neq k} |i, \beta_i\rangle - |k, \beta_k\rangle$. Therefore

$$\begin{aligned} O_p^k(|\phi\rangle\langle\phi|) = &\sum_{i\neq k}\sum_{j\neq k} |i,\beta_i\rangle\langle j,\beta_j| - (1-2p)\sum_{j\neq k}|k,\beta_k\rangle\langle j,\beta_j| \\ &- (1-2p)\sum_{i\neq k}|i,\beta_i\rangle\langle k,\beta_k| + |k,\beta_k\rangle\langle k,\beta_k| \\ = &\left(\sum_{i\neq k}|i,\beta_i\rangle - (1-2p)|k,\beta_k\rangle\right)\left(\sum_{j\neq k}\langle j,\beta_j| - (1-2p)\langle k,\beta_k|\right) \\ &+ (1-(1-2p)^2)|k,\beta_k\rangle\langle k,\beta_k|. \end{aligned}$$

□

We will use the following vectors to track the progress of the algorithm.

Definition 4. *For $k \in \{0,\ldots,N\}$ and $t \in \{0,\ldots,T\}$ we define the vectors $|\phi_t^k\rangle, |\widetilde{\phi}_t^k\rangle \in \mathbb{C}^{N\cdot M}$ and $|\alpha_{t,i}^k\rangle \in \mathbb{C}^M$ as follows. First,*

$$|\widetilde{\phi}_0^k\rangle := |\widetilde{\phi}_0\rangle,$$
$$|\phi_t^k\rangle := U_t|\widetilde{\phi}_t^k\rangle$$

and $|\alpha_{t,i}^k\rangle$ are given by

$$|\phi_t^k\rangle = \sum_{i=1}^{N} |i,\alpha_{t,i}^k\rangle.$$

Finally, for $k \in \{1,\ldots,N\}$ and $t \in \{0,\ldots,T-1\}$ we define

$$|\widetilde{\phi}_{t+1}^k\rangle := |\phi_t^k\rangle - 2(1-p)|k,\alpha_{t,k}^k\rangle = \sum_{i=1}^{N} |i,\alpha_{t,i}^k\rangle - 2(1-p)|k,\alpha_{t,k}^k\rangle$$

and for $k=0$ we define $|\widetilde{\phi}_{t+1}^0\rangle := |\phi_t^0\rangle$.

Lemma 1. *For all $t \in \{0,\ldots,T\}$ and $k \in \{1,\ldots,N\}$, we can write*

$$\rho_t^k = |\phi_t^k\rangle\langle\phi_t^k| + \sigma_t^k$$

for some positive semidefinite matrix σ_t^k.

Proof. Fix some $k \in \{1,\ldots,N\}$. The lemma clearly holds for $t=0$ (with $\sigma_0^k = 0$). Suppose the lemma holds for t and let us prove it for $t+1$. By the induction hypothesis,

$$\widetilde{\rho}_{t+1}^k = O_p^k(\rho_t^k) = O_p^k(|\phi_t^k\rangle\langle\phi_t^k|) + O_p^k(\sigma_t^k). \tag{1}$$

By Claim 1 and the definition of $|\widetilde{\phi}_t^k\rangle\langle\widetilde{\phi}_t^k|$

$$O_p^k(|\phi_t^k\rangle\langle\phi_t^k|) = |\widetilde{\phi}_{t+1}^k\rangle\langle\widetilde{\phi}_{t+1}^k| + 4p(1-p)|k,\alpha_{t,k}^k\rangle\langle k,\alpha_{t,k}^k|.$$

By combining this with Eq. (1) we get

$$\widetilde{\rho}_{t+1}^k = |\widetilde{\phi}_{t+1}^k\rangle\langle\widetilde{\phi}_{t+1}^k| + 4p(1-p)|k,\alpha_{t,k}^k\rangle\langle k,\alpha_{t,k}^k| + O_p^k(\sigma_t^k).$$

We apply U_{t+1} and obtain

$$\begin{aligned}\rho_{t+1}^k &= U_{t+1}\widetilde{\rho}_{t+1}^k U_{t+1}^\dagger \\ &= U_{t+1}|\widetilde{\phi}_{t+1}^k\rangle\langle\widetilde{\phi}_{t+1}^k|U_{t+1}^\dagger + U_{t+1}\left(4p(1-p)|k,\alpha_{t,k}^k\rangle\langle k,\alpha_{t,k}^k| + O_p^k(\sigma_t^k)\right)U_{t+1}^\dagger \\ &= |\phi_{t+1}^k\rangle\langle\phi_{t+1}^k| + U_{t+1}\left(4p(1-p)|k,\alpha_{t,k}^k\rangle\langle k,\alpha_{t,k}^k| + O_p^k(\sigma_t^k)\right)U_{t+1}^\dagger.\end{aligned}$$

The second term is clearly positive semidefinite, as required. □

We now define our progress measure H_t^k.

Definition 5. *For $t \in \{0, \ldots, T\}$ and $k \in \{1, \ldots, N\}$ we define*

$$H_t^k := \left\| |\phi_t^0\rangle - |\phi_t^k\rangle \right\|^2.$$

Notice that $H_0^k = 0$. The following lemma shows that at the end of the algorithm, the progress measure must be not too small. Intuitively, this holds since if H_T^k is small, then $|\phi_T^k\rangle$ is close to $|\phi_T^0\rangle$ and since the latter is a unit vector, the former must be of norm close to 1. This, in turn, implies that ρ_T^k is close to $|\phi_T^k\rangle\langle\phi_T^k|$, which is close to $|\phi_T^0\rangle\langle\phi_T^0| = \rho_T^0$ and thus the algorithm cannot distinguish between ρ_T^k and ρ_T^0 in contrast to our assumption about the algorithm. We proceed with the formal proof.

Lemma 2. *For all $k \in \{1, \ldots, N\}$, $H_T^k > \frac{1}{10}$.*

Proof. By our assumption on the correctness of the algorithm,

$$\begin{aligned}
\frac{9}{10} \leq \left\| \rho_T^k - \rho_T^0 \right\|_{\mathrm{tr}} &= \left\| \rho_T^k - |\phi_T^0\rangle\langle\phi_T^0| \right\|_{\mathrm{tr}} \\
&\leq \sqrt{1 - \langle\phi_T^0|\rho_T^k|\phi_T^0\rangle} \\
&= \sqrt{1 - \langle\phi_T^0|(|\phi_T^k\rangle\langle\phi_T^k| + \sigma_T^k)|\phi_T^0\rangle} \\
&\leq \sqrt{1 - |\langle\phi_T^0|\phi_T^k\rangle|^2}
\end{aligned}$$

where our definition of trace norm is normalized to be in $[0, 1]$ and in the second inequality we used that for a (normalized) pure state $|\varphi\rangle$ and a mixed state ρ, we have $\|\rho - |\varphi\rangle\langle\varphi|\|_{\mathrm{tr}} \leq \sqrt{1 - \langle\varphi|\rho|\varphi\rangle}$ (see, e.g., [19, Chapter 9]). Therefore,

$$\begin{aligned}
H_T^k &= \left\| |\phi_T^0\rangle - |\phi_T^k\rangle \right\|^2 \\
&= \langle\phi_T^0|\phi_T^0\rangle + \langle\phi_T^k|\phi_T^k\rangle - 2\mathrm{Re}(\langle\phi_T^0|\phi_T^k\rangle) \\
&\geq 1 - 2|\langle\phi_T^0|\phi_T^k\rangle| > \frac{1}{10},
\end{aligned}$$

where the next to last inequality uses the fact that $\langle\phi_T^0|\phi_T^0\rangle = 1$ and $\langle\phi_T^k|\phi_T^k\rangle \geq 0$. □

The following lemma bounds the amount by which the progress measure H_t^k can increase in each step.

Lemma 3. *For all $k \in \{1, \ldots, N\}$ and any $0 \leq t < T$,*

$$H_{t+1}^k - H_t^k \leq \frac{1-p}{p} \cdot \|\alpha_{t,k}^0\|^2.$$

Proof. By the definition of the progress measure,

$$\begin{aligned}
H_{t+1}^k &= \left\| |\phi_{t+1}^k\rangle - |\phi_{t+1}^0\rangle \right\|^2 \\
&= \left\| U_{t+1}|\widetilde{\phi}_{t+1}^k\rangle - U_{t+1}|\phi_t^0\rangle \right\|^2 \\
&= \left\| |\widetilde{\phi}_{t+1}^k\rangle - |\phi_t^0\rangle \right\|^2 \\
&= \left(\langle\phi_t^k| - 2(1-p)\langle k, \alpha_{t,k}^k| - \langle\phi_t^0| \right) \left(|\phi_t^k\rangle - 2(1-p)|k, \alpha_{t,k}^k\rangle - |\phi_t^0\rangle \right) \\
&= H_t^k - 4(1-p)\|\alpha_{t,k}^k\|^2 + 2(1-p)\langle\alpha_{t,k}^k|\alpha_{t,k}^0\rangle \\
&\quad + 2(1-p)\langle\alpha_{t,k}^0|\alpha_{t,k}^k\rangle + 4(1-p)^2\|\alpha_{t,k}^k\|^2 \\
&\leq H_t^k - 4p(1-p)\|\alpha_{t,k}^k\|^2 + 4(1-p)\|\alpha_{t,k}^k\|\|\alpha_{t,k}^0\| \\
&\leq H_t^k + \frac{1-p}{p}\|\alpha_{t,k}^0\|^2
\end{aligned}$$

where the last inequality follows by maximizing the quadratic expression over $\|\alpha_{t,k}^k\|$. □

Theorem 1. *Any algorithm that solves the p-faulty Grover problem must use* $T > \frac{p}{10(1-p)}N$ *queries.*

Proof. By Lemma 3, for all $k \in \{1, \ldots, N\}$,

$$H_T^k \leq \frac{1-p}{p}\sum_{t=0}^{T-1}\|\alpha_{t,k}^0\|^2.$$

Since for any t, $|\phi_t^0\rangle$ is a unit vector,

$$\sum_{k=1}^{N} H_T^k \leq \frac{1-p}{p}\sum_{k=1}^{N}\sum_{t=0}^{T-1}\|\alpha_{t,k}^0\|^2 = \frac{1-p}{p}T.$$

To complete the proof, note that by Lemma 2, $\sum_{k=1}^{N} H_T^k > \frac{1}{10}N$. □

Acknowledgments. We thank Aram Harrow for presenting us with the faulty Grover problem and for useful discussions. We also thank Nicolas Cerf, Frédéric Magniez, and the anonymous referees for useful comments.

References

1. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the ACM Symposium on the Theory of Computing, pp. 212–219 (1996)
2. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM J. Comput. 26(5), 1510–1523 (1997)
3. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik 46, 493–505 (1998)

4. Ambainis, A.: Quantum lower bounds by quantum arguments. In: Proceedings of the ACM Symposium on Theory of Computing, New York, pp. 636–643 (2000)
5. Ambainis, A.: Quantum search algorithms. SIGACT News 35(2), 22–35 (2004)
6. Harrow, A.: Personal communication (2006)
7. Shenvi, N., Brown, K.R., Whaley, K.B.: Effects of a random noisy oracle on search algorithm complexity. Phys. Rev. A 68(5), 052313 (2003)
8. Knill, E., Laflamme, R., Zurek, W.H.: Resilient quantum computation. Science 279(5349), 342–345 (1998)
9. Long, G.L., Li, Y.S., Zhang, W.L., Tu, C.C.: Dominant gate imperfection in Grover's quantum search algorithm. Physical Review A 61, 042305 (2000)
10. Shapira, D., Mozes, S., Biham, O.: Effect of unitary noise on Grover's quantum search algorithm. Phys. Rev. A 67(4), 42301 (2003)
11. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum computation and information. Contemp. Math, vol. 305, pp. 53–74. Amer. Math. Soc., Providence (2002)
12. Høyer, P., Mosca, M., de Wolf, R.: Quantum search on bounded-error inputs. In: Proceedings of ICALP 2003. LNCS, vol. 2719, pp. 291–299. Springer, Berlin (2003)
13. Buhrman, H., Newman, I., Röhrig, H., de Wolf, R.: Robust polynomials and quantum algorithms. Theory Comput. Syst. 40(4), 379–395 (2007); Preliminary version in STACS 2005
14. Iwama, K., Raymond, R., Yamashita, S.: General bounds for quantum biased oracles. IPSJ Journal 46(10), 1234–1243 (2005)
15. Suzuki, T., Yamashita, S., Nakanishi, M., Watanabe, K.: Robust quantum algorithms with ϵ-biased oracles. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 116–125. Springer, Heidelberg (2006)
16. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. In: Proceedings of the ACM Symposium on the Theory of Computing, New York, pp. 575–584 (2007)
17. Høyer, P., Lee, T., Špalek, R.: Negative weights make adversaries stronger. In: Proceedings of the ACM Symposium on the Theory of Computing, pp. 526–535 (2007) quant-ph/0611054
18. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. J. ACM 48(4), 778–797 (2001)
19. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
20. Kaye, P., Laflamme, R., Mosca, M.: An introduction to quantum computing. Oxford University Press, Oxford (2007)

Superpolynomial Speedups Based on Almost Any Quantum Circuit

Sean Hallgren[1] and Aram W. Harrow[2]

[1] Department of Computer Science and Engineering, The Pennsylvania State University University Park, PA
[2] Department of Mathematics, University of Bristol, Bristol, U.K.
a.harrow@bris.ac.uk

Abstract. The first separation between quantum polynomial time and classical bounded-error polynomial time was due to Bernstein and Vazirani in 1993. They first showed a $O(1)$ vs. $\Omega(n)$ quantum-classical oracle separation based on the quantum Hadamard transform, and then showed how to amplify this into a $n^{O(1)}$ time quantum algorithm and a $n^{\Omega(\log n)}$ classical query lower bound.

We generalize both aspects of this speedup. We show that a wide class of unitary circuits (which we call *dispersing* circuits) can be used in place of Hadamards to obtain a $O(1)$ vs. $\Omega(n)$ separation. The class of dispersing circuits includes all quantum Fourier transforms (including over nonabelian groups) as well as nearly all sufficiently long random circuits. Second, we give a general method for amplifying quantum-classical separations that allows us to achieve a $n^{O(1)}$ vs. $n^{\Omega(\log n)}$ separation from any dispersing circuit.

1 Background

Understanding the power of quantum computation relative to classical computation is a fundamental question. When we look at which problems can be solved in quantum but not classical polynomial time, we get a wide range: quantum simulation, factoring, approximating the Jones polynomial, Pell's equation, estimating Gauss sums, period-finding, group order-finding and even detecting some mildly non-abelian symmetries [Sho97, Hal07, Wat01, FIM+03, vDHI03]. However, when we look at what algorithmic tools exist on a quantum computer, the situation is not nearly as diverse. Apart from the BQP-complete problems [AJL06], the main tool for solving most of these problems is a quantum Fourier transform (QFT) over some group. Moreover, the successes have been for cases where the group is abelian or close to abelian in some way. For sufficiently nonabelian groups, there has been no indication that the transforms are useful even though they can be computed exponentially faster than classically. For example, while an efficient QFT for the symmetric group has been intensively studied for over a decade because of its connection to graph isomorphism, it is still unknown whether it can be used to achieve any kind of speedup over classical computation [Bea97].

The first separation between quantum computation and randomized computation was the Recursive Fourier Sampling problem (RFS) [BV97]. This algorithm had two components, namely using a Fourier transform, and using recursion. Shortly after this,

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 782–795, 2008.

Simon's algorithm and then Shor's algorithm for factoring were discovered, and the techniques from these algorithms have been the focus of most quantum algorithmic research since [Sim97, Sho97]. These developed into the hidden subgroup framework. The hidden subgroup problem is an oracle problem, but solving certain cases of it would result in solutions for factoring, graph isomorphism, and certain shortest lattice vector problems. Indeed, it was hoped that an algorithm for graph isomorphism could be found, but recent evidence suggests that this approach may not lead to one [HMR+06]. As a way to understand new techniques, this oracle problem has been very important, and it is also one of the very few where super-polynomial speedups have been found [IMS01, BCvD05].

In comparison to factoring, the RFS problem has received much less attention. The problem is defined as a property of a tree with labeled nodes and it was proven to be solvable with a quantum algorithm super-polynomially faster than the best randomized algorithm. This tree was defined in terms of the Fourier coefficients over $\mathbb{Z}_2^n$. The definition was rather technical, and it seemed that the simplicity of the Fourier coefficients for this group was necessary for the construction to work. Even the variants introduced by Aaronson [Aar03] were still based on the same QFT over $\mathbb{Z}_2^n$, which seemed to indicate that this particular abelian QFT was a key part of the quantum advantage for RFS.

The main result of this paper is to show that the RFS structure can be generalized far more broadly. In particular, we show that an RFS-style super-polynomial speedup is achievable using almost any quantum circuit, and more specifically, it is also true for any Fourier transform (even nonabelian), not just over $\mathbb{Z}_2^n$. This illustrates a more general power that quantum computation has over classical computation when using recursion. The condition for a quantum circuit to be useful for an RFS-style speedup is that the circuit be *dispersing*, a concept we introduce to mean that it takes many different inputs to fairly even superpositions over most of the computational basis.

Our algorithm should be contrasted with the original RFS algorithm. One of the main differences between classical and quantum computing is so-called garbage that results from computing. It is important in certain cases, and crucial in recursion-based quantum algorithms because of quantum superpositions, that intermediate computations are uncomputed and that errors do not compound. The original RFS paper [BV97] avoided the error issue by using an oracle problem where every quantum state create from it had the exact property necessary with no errors. Their algorithm could have tolerated polynomially small errors, but in this paper we relax this significantly. We show that even if we can only create states with constant accuracy at each level of recursion, we can still carry through a recursive algorithm which introduces new constant-sized errors a polynomial number of times.

The main technical part of our paper shows that most quantum circuits can be used to construct separations relative to appropriate oracles. To understand the difficulty here, consider two problems that occur when one tries to define an oracle whose output is related to the amplitudes that result from running a circuit. First, it is not clear how to implement such an oracle since different amplitudes have different magnitudes, and only phases can be changed easily. Second, we need an oracle where we can prove that a classical algorithm requires many queries to solve the problem. If the oracle outputs many bits, this can be difficult or impossible to achieve. For example, the matrix

entries of nonabelian groups can quickly reveal which representation is being used. To overcome these two problems we show that there are binary-valued functions that can approximate the complex-valued output of quantum circuits in a certain way.

One by-product of our algorithm is related to the Fourier transform of the symmetric group. Despite some initial promise for solving graph isomorphism, the symmetric group QFT has still not found any application in quantum algorithms. One instance of our result is the first example of a problem (albeit a rather artificial one) where the QFT over the symmetric group is used to achieve a super-polynomial speedup.

2 Statement of Results

Our main contributions are to generalize the RFS algorithm of [BV97] in two stages. First, [BV97] described the problem of Fourier sampling over $\mathbb{Z}_2^n$, which has an $O(1)$ vs. $\Omega(n)$ separation between quantum and randomized complexities. We show that here the QFT over $\mathbb{Z}_2^n$ can be replaced with a QFT over any group, or for that matter with almost any quantum circuit. Next, [BV97] turned Fourier sampling into recursive Fourier sampling with a recursive technique. We will generalize this construction to cope with error and to amplify a larger class of quantum speedups. As a result, we can turn any of the linear speedups we have found into superpolynomial speedups.

Let us now explain each of these steps in more detail. We replace the $O(1)$ vs $\Omega(n)$ separation based on Fourier sampling with a similar separation based on a more general problem called *oracle identification*. In the oracle identification problem, we are given access to an oracle $\mathcal{O}_a : X \to \{0,1\}$ where $a \in A$, for some sets A and X with $\log|A|, \log|X| = \Theta(n)$. Our goal is to determine the identity of a. Further, assume that we have access to a testing oracle $T_a : A \to \{0,1\}$ defined by $T_a(a') = \delta_{a,a'}$, that will let us confirm that we have the right answer.[1]

A quantum algorithm for identifying a can be described as follows: first prepare a state $|\varphi_a\rangle$ using q queries to $\mathcal{O}_a$, then perform a POVM $\{\Pi_{a'}\}_{a' \in A}$ (with $\sum_{a'} \Pi_{a'} \leq I$ to allow for the possibility of a "failure" outcome), using no further queries to $\mathcal{O}_a$. The success probability is $\langle\varphi_a|\Pi_a|\varphi_a\rangle$. For our purposes, it will suffice to place a $\Omega(1)$ lower bound on this probability: say that for each a, $\langle\varphi_a|\Pi_a|\varphi_a\rangle \geq \delta$ for some constant $\delta > 0$. On the other hand, any classical algorithm trivially requires $\geq \log(|A|\delta) = \Omega(n)$ oracle calls to identify a with success probability $\geq \delta$. This is because each query returns only one bit of information. In Theorem 9 we will describe how a large class of quantum circuits can achieve this $O(1)$ vs. $\Omega(n)$ separation, and in Theorems 11 and 12 we will show specifically that QFTs and most random circuits fall within this class.

Now we describe the amplification step. This is a variant of the [BV97] procedure in which making an oracle call in the original problem requires solving a sub-problem from the same family as the original problem. Iterating this ℓ times turns query complexity q into $q^{\Theta(\ell)}$, so choosing $\ell = \Theta(\log n)$ will yield the desired polynomial vs.

[1] This will later allow us to turn two-sided into one-sided error; unfortunately it also means that a non-deterministic Turing machine can find a with a single query to T_a. Thus, while the oracle defined in BV is a candidate for placing BQP outside PH, ours will not be able to place BQP outside of NP. This limitation appears not to be fundamental, but we will leave the problem of circumventing it to future work.

super-polynomial separation. We will generalize this construction by defining an amplified version of oracle identification called *recursive oracle identification*. This is described in the next section, where we will see how it gives rise to superpolynomial speedups from a broad class of circuits.

We conclude that quantum speedups—even superpolynomial speedups—are much more common than the conventional wisdom would suggest. Moreover, as useful as the QFT has been to quantum algorithms, it is far from the only source of quantum algorithmic advantage.

3 Recursive Amplification

In this section we show that once we are given a constant versus linear separation (for quantum versus classical oracle identification), we are able to amplify this to a superpolynomial speedup. We require a much looser definition than in [BV97] because the constant case can have a large error.

Definition 1. *For sets A, X, let $f : A \times X \to \{0,1\}$ be a function. To set the scale of the problem, let $|X| = 2^n$ and $|A| = 2^{\Omega(n)}$. Define the set of oracles $\{\mathcal{O}_a : a \in A\}$ by $\mathcal{O}_a(x) = f(a,x)$, and the states $|\varphi_a\rangle = \frac{1}{\sqrt{|X|}} \sum_{x \in X} (-1)^{f(a,x)} |x\rangle$. The single-level oracle identification problem is defined to be the task of determining a given access to $\mathcal{O}_a$. Let U be a family of quantum circuits, implicitly depending on n. We say that U solves the single-level oracle identification problem if*

$$|\langle a|U|\varphi_a\rangle|^2 \geq \Omega(1)$$

for all sufficiently large n and all $a \in A$. In this case, we define the POVM $\{\Pi_a\}_{a \in A}$ by $\Pi_a = U^\dagger |a\rangle\langle a| U$.

When this occurs, it means that a can be identified from $\mathcal{O}_a$ with $\Omega(1)$ success probability and using a single query. In the next section, we will show how a broad class of unitaries U (the so-called *dispersing* unitaries) allow us to construct f for which U solves the single-level oracle identification problem. There are natural generalizations to oracle identification problems requiring many queries, but we will not explore them here.

Theorem 2. *Suppose we are given a single-level oracle problem with function f and unitary U running in time $\mathrm{poly}(n)$. Then we can construct a modified oracle problem from f which can be solved by a quantum computer in polynomial time (and queries), but requires $n^{\Omega(\log n)}$ queries for any classical algorithm that succeeds with probability $\frac{1}{2} + n^{-o(\log n)}$.*

We start by defining the modified version of the problem (Definition 3 below), and describing a quantum algorithm to solve it. Then in Theorem 4 we will show that the quantum algorithm solves the problem correctly in polynomial time, and in Theorem 6, we will show that randomized classical algorithms require superpolynomial time to have a nonnegligible probability of success.

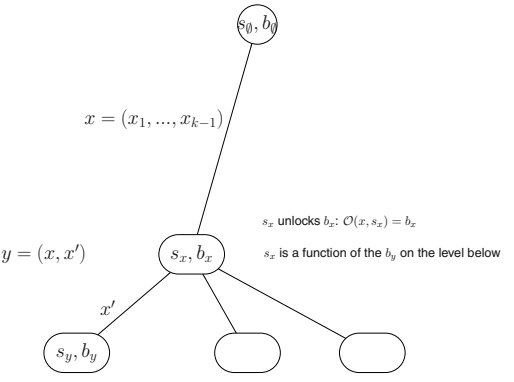


Fig. 1. A depth k node at location $x = (x_1, \ldots, x_k)$ is labeled by its secret s_x and a bit b_x. The secret s_x can be computed from the bits b_y of its children, and once it is known, the bit b_x is computed from the oracle $\mathcal{O}(x, s_x) = b_x$. If x is a leaf then it has no secret and we simply have $b_x = \mathcal{O}(x)$. The goal is to compute the secret bit $b_\emptyset$ at the root.

The recursive version of the problem simply requires that another instance of the problem be solved in order to access a value at a child. Figure 1 illustrates the structure of the problem.

Using the notation from Figure 1, the relation between a secret s_x, and the bits b_y of its children is given by $b_y = f(s_x, x')$, where f is the function from the single-level oracle identification problem. Thus by computing enough of the bits $b_{y_1}, b_{y_2}, \ldots$ corresponding to children $y_1, y_2, \ldots$, we can solve the single-level oracle identification problem to find s_x. Of course computing the b_y will require finding the secret strings s_y, which requires finding the bits of *their* children and so on, until we reach the bottom layer where queries return answer bits without the need to first produce secret strings.

Definition 3. *A level-ℓ recursive oracle identification problem is specified by X, A and f from a single-level oracle identification problem (Definition 1), any function $s : \emptyset \cup X \cup X \times X \cup \ldots \cup X^{\ell-1} \to A$, and any final answer $b_\emptyset \in \{0, 1\}$. Given these ingredients, an oracle $\mathcal{O}$ is defined which takes inputs in*

$$\bigcup_{k=0}^{\ell-1} \left[X^k \times A\right] \cup X^\ell$$

and to return outputs in $\{0, 1, \text{FAIL}\}$. On inputs $x_1, \ldots, x_k \in X, a \in A$ with $1 \leq k < \ell$, $\mathcal{O}$ returns

$$\mathcal{O}(x_1, \ldots, x_k, a) = f(s(x_1, \ldots, x_{k-1}), x_k) \qquad \text{when } a = s(x_1, \ldots, x_k) \tag{1}$$

$$\mathcal{O}(x_1, \ldots, x_k, a) = \text{FAIL} \qquad \text{when } a \neq s(x_1, \ldots, x_k). \tag{2}$$

If $k = 0$, then $\mathcal{O}(s(\emptyset)) = b_\emptyset$ and $\mathcal{O}(a) = \text{FAIL}$ if $a \neq s(\emptyset)$. When $k = \ell$,

$$\mathcal{O}(x_1, \ldots, x_\ell) = f(s(x_1, \ldots, x_{\ell-1}), x_\ell).$$

The recursive oracle identification problem is to determine $b_\emptyset$ given access to $\mathcal{O}$.

Note that the function s gives the values s_x in Figure 1. These values are actually defined in the oracle and can be chosen arbitrarily at each node. Note also that the oracle defined here effectively includes a testing oracle, which can determine whether $a = s(x_1, \ldots, x_k)$ for any $a \in A, x_1, \ldots, x_k \in X$ with one query. (When $x = (x_1, \ldots, x_k)$, we use $s(x_1, \ldots, x_k)$ and s_x interchangeably.) A significant difference between our construction and that of [BV97] is that the values of s at different nodes can be set completely independently in our construction, whereas [BV97] had a complicated consistency requirement.

The algorithm. Now we turn to a quantum algorithm for the recursive oracle identification problem. If a quantum computer can identify a with one-sided[2] error $1 - \delta$ using time T and q queries in the non-recursive problem, then we will show that the recursive version can be solved in time $O((q\frac{\log 1/\delta}{\delta})^\ell T)$. For concreteness, suppose that $|\varphi_a\rangle = \frac{1}{\sqrt{|X|}} \sum_{x \in X} (-1)^{f(a,x)} |x\rangle$, so that $q = 1$; the case when $q > 1$ is an easy, but tedious, generalization. Suppose that our identifying quantum circuit is U, so a can be identified by applying the POVM $\{\Pi_{a'}\}_{a' \in A}$ with $\Pi_{a'} = U^\dagger |a'\rangle\langle a'| U$ to the state $|\varphi_a\rangle$.

The intuitive idea behind our algorithm is as follows: At each level, we find $s(x_1, \ldots, x_k)$ by recursively computing $s(x_1, \ldots, x_{k+1})$ for each x_{k+1} (in superposition) and using this information to create many copies of $|\varphi_{s(x_1,\ldots,x_k)}\rangle$, from which we can extract our answer. However, we need to account for the errors carefully so that they do not blow up as we iterate the recursion. In what follows, we will adopt the convention that Latin letters in kets (e.g. $|a\rangle, |x\rangle, \ldots$) denote computational basis states, while Greek letters (e.g. $|\zeta\rangle, |\varphi\rangle, \ldots$) are general states that are possibly superpositions over many computational basis states. Also, we let the subscript $_{(k)}$ indicate a dependence on $(x_1, \ldots, x_k)$. The recursive oracle identification algorithm is as follows:

Algorithm: FIND
Input: $|x_1, \ldots, x_k\rangle|0\rangle$ for $k < \ell$
Output: $a_{(k)} = s(x_1, \ldots, x_k)$ up to error $\varepsilon = (\delta/8)^2$, where δ is the constant from the oracle. This means $|x_1, \ldots, x_k\rangle \left[\sqrt{1-\varepsilon_{(k)}}|0\rangle|a_{(k)}\rangle|\zeta_{(k)}\rangle + \sqrt{\varepsilon_{(k)}}|1\rangle|\zeta'_{(k)}\rangle\right]$, where $\varepsilon_{(k)} \le \varepsilon$ and $|\zeta_{(k)}\rangle$ and $|\zeta'_{(k)}\rangle$ are arbitrary. (We can assume this form without loss of generality by absorbing phases into $|\zeta_{(k)}\rangle$ and $|\zeta'_{(k)}\rangle$.)

1. Create the superposition $\frac{1}{\sqrt{|X|}} \sum_{x_{k+1} \in X} |x_{k+1}\rangle$.
2. If $k + 1 < \ell$ then let $a_{(k+1)} = \text{FIND}(x_1, \ldots, x_{k+1})$ (with error $\le \varepsilon$), otherwise $a_{(k+1)} = \emptyset$.
3. Call the oracle $\mathcal{O}(x_1, \ldots, x_{k+1}, a_{(k+1)})$ to apply the phase $(-1)^{f(s(x_1,\ldots,x_k),x_{k+1})}$ using the key $a_{(k+1)}$.
4. If $k + 1 < \ell$ then call FIND† to (approximately) uncompute $a_{(k+1)}$.
5. We are now left with $|\tilde{\varphi}_{(k)}\rangle$, which is close to $|\varphi_{s(x_1,\ldots,x_k)}\rangle$.
   Repeat steps 1–4 $m = \frac{4}{\delta} \ln \frac{8}{\delta}$ times to obtain $|\tilde{\varphi}_{(k)}\rangle^{\otimes m}$
6. Coherently measure $\{\Pi_a\}$ on each copy and test the results (i.e. apply U, test the result, and apply $U^\dagger$).
7. If any tests pass, copy the correct $a_{(k)}$ to an output register, along with $|0\rangle$ to indicate success.
   Otherwise put a $|1\rangle$ in the output to indicate failure.
8. Let everything else comprise the junk register $|\zeta_{(k)}\rangle$.

Theorem 4. *Calling* FIND *on* $|0\rangle$ *solves the recursive oracle problem in quantum polynomial time.*

[2] One-sided error is a reasonable demand given our access to a testing oracle. Most of these results go through with two-sided error as well, but for notational simplicity, we will not explore them here.

Proof. The proof is by backward induction on k; we assume that the algorithm returns with error $\leq \varepsilon$ for $k+1$ and prove it for k. The initial step when $k = \ell$ is trivial since there is no need to compute $a_{\ell+1}$, and thus no source of error. If $k < \ell$, then assume that correctness of the algorithm has already been proved for $k+1$. Therefore Step 2 leaves the state

$$\frac{1}{\sqrt{|X|}} \sum_{x_{k+1} \in X} |x_{k+1}\rangle \left[\sqrt{1-\varepsilon_{(k+1)}}|0\rangle|a_{(k+1)}\rangle|\zeta_{(k+1)}\rangle + \sqrt{\varepsilon_{(k+1)}}|1\rangle|\zeta'_{(k+1)}\rangle \right].$$

In Step 3, we assume for simplicity that the oracle was called conditional on the success of Step 2. This yields

$$|\psi'_{(k)}\rangle := \frac{1}{\sqrt{|X|}} \sum_{x_{k+1} \in X} |x_{k+1}\rangle \left[(-1)^{f(a_{(k)}, x_{k+1})} \sqrt{1-\varepsilon_{(k+1)}}|0\rangle|a_{(k+1)}\rangle|\zeta_{(k+1)}\rangle + \sqrt{\varepsilon_{(k+1)}}|1\rangle|\zeta'_{(k+1)}\rangle \right].$$

Now define the state $|\psi_{(k)}\rangle$ by

$$|\psi_{(k)}\rangle := \frac{1}{\sqrt{|X|}} \sum_{x_{k+1} \in X} (-1)^{f(a_{(k)}, x_{k+1})} |x_{k+1}\rangle \left[\sqrt{1-\varepsilon_{(k+1)}}|0\rangle|a_{(k+1)}\rangle|\zeta_{(k+1)}\rangle + \sqrt{\varepsilon_{(k+1)}}|1\rangle|\zeta'_{(k+1)}\rangle \right].$$

Note that

$$\langle \psi'_{(k)} | \psi_{(k)} \rangle = \frac{1}{|X|} \sum_{x_{k+1} \in X} \left(1 - \varepsilon_{(k+1)} + (-1)^{f(a_{(k)}, x_{k+1})} \varepsilon_{(k+1)} \right).$$

This quantity is real and always $\geq 1 - 2\varepsilon_{(k+1)} \geq \sqrt{1-4\varepsilon}$ by the induction hypothesis. Let

$$|\phi_{(k)}\rangle := \frac{1}{|X|} \sum_{x_{k+1} \in X} (-1)^{f(a_{(k)}, x_{k+1})} |x_{k+1}\rangle|0\rangle.$$

Note that $\mathrm{FIND}^\dagger |x_1, \ldots, x_k, \psi_{(k)}\rangle = |x_1, \ldots, x_k, \phi_{(k)}\rangle$. Thus there exists $\varepsilon_{(k)}$ such that applying $\mathrm{FIND}^\dagger$ to $|x_1, \ldots, x_k\rangle|\psi'_{(k)}\rangle$ yields

$$|x_1, \ldots, x_k\rangle \otimes \left[\sqrt{1-4\varepsilon_{(k)}}|\phi_{(k)}\rangle + \sqrt{4\varepsilon_{(k)}}|\phi'_{(k)}\rangle \right],$$

where $\langle \phi_{(k)} | \phi'_{(k)} \rangle = 0$ and $\varepsilon_{(k)} \leq \varepsilon$.

We now want to analyze the effects of measuring $\{\Pi_a\}$ when we are given the state

$$|\varphi_{(k)}\rangle := \sqrt{1-4\varepsilon_{(k)}}|\phi_{(k)}\rangle + \sqrt{4\varepsilon_{(k)}}|\phi'_{(k)}\rangle$$

instead of $|\phi_{(k)}\rangle$. If we define $\|M\|_1 = \mathrm{tr}\sqrt{M^\dagger M}$ for a matrix M, then $\| \, |\varphi_{(k)}\rangle\langle\varphi_{(k)}| - |\phi_{(k)}\rangle\langle\phi_{(k)}| \, \|_1 = 4\sqrt{\varepsilon_{(k)}}$ [FvdG99]. Thus

$$\langle \varphi_{(k)} | \Pi_{a_{(k)}} | \varphi_{(k)} \rangle \geq \langle \phi_{(k)} | \Pi_{a_{(k)}} | \phi_{(k)} \rangle - 4\sqrt{\varepsilon_{(k)}} \geq \delta - 4\sqrt{\varepsilon_{(k)}} \geq \delta/2.$$

In the last step we have chosen $\varepsilon = (\delta/8)^2$.

Finally, we need to guarantee that with probability $\geq 1-\varepsilon$ at least one of the tests in Step 6 passes. After applying U and the test oracle to $|\varphi_{(k)}\rangle$, we have $\geq \sqrt{\delta/2}$ overlap with a successful test and $\leq \sqrt{1-\delta/2}$ overlap with an unsuccessful test. When we repeat this m times, the amplitude in the subspace corresponding to all tests failing is $\leq (1-\delta/2)^{m/2} \leq e^{-m\delta/4}$. If we choose $m = (2/\delta)\ln(1/\varepsilon) = (4/\delta)\ln(8/\delta)$ then the failure amplitude will be $\leq \sqrt{\varepsilon}$, as desired.

To analyze the time complexity, first note that the run-time is $O(T)$ times the number of queries made by the algorithm, and we have assumed that T is polynomial in n. Suppose the algorithm at level k requires $Q(k)$ queries. Then steps 2 and 4 require $mQ(k+1)$ queries each, steps 3 and 6 require m queries each and together $Q(k) = 2mQ(k+1)+2m$. The base case is $k=\ell$, for which $Q(\ell)=0$, since there are no secret strings to calculate for the leaves. The total number of queries required for the algorithm is then $Q(0) \approx (2m)^{2\ell}$. If we choose $\ell = \log n$ the quantum query complexity will thus be $n^{2\log 2m} = n^{O(1)}$ and the quantum complexity will be polynomial in n compared with the $n^{\Omega(\log n)}$ lower bound.

This concludes the demonstration of the polynomial-time quantum algorithm. Now we turn to the classical $n^{\Omega(\log n)}$ lower bound. Our key technical result is the following lemma:

Lemma 5. *Define the recursive oracle identification problem as above, with a function $f : A \times X \to \{0,1\}$ and a secret $s : \emptyset \cup X \cup X \times X \cup \ldots \cup X^{\ell-1} \mapsto A$ encoded in an oracle $\mathcal{O}$. Fix a deterministic classical algorithm that makes $\leq Q$ queries to $\mathcal{O}$. Then if s and* ANS *are chosen uniformly at random, the probability that* ANS *is output by the algorithm is*

$$\leq \frac{1}{2} + \max\left(\frac{Q}{|A|^{1/3}-Q}, Q\left(\frac{\log|A|}{3}\right)^{-\ell}\right).$$

Using Yao's minimax principle and plugging in $|A| = 2^{\alpha n}$, $\ell = \log n$ and $Q = n^{o(\log n)}$ readily yields.

Theorem 6. *If $\log|A| = n^{\Omega(1)}$ and $\ell = \Omega(\log n)$, then any randomized classical algorithm using $Q = n^{o(\log n)}$ queries will have $\frac{1}{2} + n^{-\Omega(\log n)}$ probability of successfully outputting* ANS.

Proof (of Lemma 5). Let $T = \emptyset \cup X \cup \ldots \cup X^{\ell}$ denote the tree on which the oracle is defined. We say that a node $x \in T$ has been *hit* by the algorithm if position x has been queried by the oracle together with the correct secret, i.e. $\mathcal{O}(s(x), x)$ has been queried. The only way to find to obtain information about ANS is for the algorithm to query $\emptyset$ with the appropriate secret; in other words, to hit $\emptyset$.

For $x, y \in T$ we say that x is an *ancestor* of y, and that y is a *descendant* of x, if $y = x \times z$ for some $z \in T$. If $z \in X$ then we say that y is a *child* of x and that x is a *parent* of y. Now define $S \subset T$ to be the set of all $x \in T$ such that x has been hit but none of x's ancestors have been. Also define a function $d(x)$ to be the depth of a node x; i.e. for all $x \in X^k$, $d(x) = k$. We combine these definitions to declare an invariant

$$Z = \sum_{x \in S}\left(\frac{\log|A|}{3}\right)^{-d(x)}$$

The key properties of Z we need are that:

1. Initially $Z = 0$.
2. If the algorithm is successful then it terminates with $Z = 1$.
3. Only oracle queries change the value of Z.
4. Querying a leaf can add at most $(\log |A|/3)^{-\ell}$ to Z.
5. Querying an internal node (i.e. not a leaf) can add at most $2/(|A|^{1/3} - Q)$ to $\mathbf{E}\, Z$, where $\mathbf{E}$ indicates the expectation over random choices of s.

Combining these facts yields the desired bound.

Properties 1–4 follow directly from the definition (with the inequality in property 4 because it is possible to query a node that has already been hit). To establish property 5, suppose that the algorithm queries node $x \in T$ and that it has previously hit k of x's children. This gives us some partial information about $s(x)$. We can model this information as a partition of A into 2^k disjoint sets $A_1, \ldots, A_{2^k}$ (of which some could be empty). From the k bits returned by the oracle on the k children of x we have successfully queried, we know not only that $s(x) \in A$, but that $s(x) \in A_i$ for some $i \in \{1, \ldots, 2^k\}$.

We will now divide the analysis into two cases. Either $k \leq \frac{1}{3} \log |A|$ or $k > \frac{1}{3} \log |A|$. We will argue that in the former case, $|A_i|$ is likely to be large, and so we are unlikely so successfully guess $s(x)$, while in the latter case even a successful guess will not increase Z. The latter case ($k > \frac{1}{3} \log |A|$) is easier, so we consider it first. In this case, Z only changes if x is hit in this step and neither x nor any of its ancestors have been previously hit. Then even though hitting x will contribute $(\log |A|/3)^{-d(x)}$ to Z, it will also remove the k children from S (as well as any other descendants of x), which will decrease Z by at least $k(\log |A|/3)^{-d(x)-1} > (\log |A|/3)^{-d(x)}$, resulting in a net decrease of Z.

Now suppose that $k \leq \frac{1}{3} \log |A|$. Recall that our information about $s(x)$ can be expressed by the fact that $s(x) \in A_i$ for some $i \in \{1, \ldots, 2^k\}$. Since the values of s were chosen uniformly at random, we have $\Pr(A_i) = |A_i|/|A|$. Say that a set A_i is *bad* if $|A_i| \leq |A|^{2/3}/2^k$. Then for a particular bad set A_i, $\Pr(A_i) \leq |A|^{-1/3} 2^{-k}$. From the union bound, we see that the probability that *any* bad set is chosen is $\leq |A|^{-1/3}$.

Assume then that we have chosen a good set A_i, meaning that conditioned on the values of the children there are $|A_i| \geq |A|^{2/3}/2^k \geq |A|^{1/3}$ possible values of $s(x)$. However, previous failed queries at x may also have ruled out specific possible values of x. There have been at most Q queries at x, so there are $\geq |A|^{1/3} - Q$ possible values of $s(x)$ remaining. (Queries to any other nodes in the graph yield no information on $s(x)$.) Thus the probability of hitting x is $\leq 1/(|A|^{1/3} - Q)$ if we have chosen a good set. We also have a $\leq |A|^{-1/3}$ probability of choosing a bad set, so the total probability of hitting x (in the $k \leq \frac{1}{3} \log |A|$ case) is $\leq |A|^{-1/3} + 1/(|A|^{1/3} - Q) \leq 2/(|A|^{1/3} - Q)$. Finally, hitting x will increase Z by at most one, so the largest possible increase of $\mathbf{E}\, Z$ when querying a non-leaf node is $\leq 2/(|A|^{1/3} - Q)$. This completes the proof of property 5 and thus the Lemma.

4 Dispersing Circuits

In this section we define *dispersing* circuits and show how to construct an oracle problem with a constant versus linear separation from any such circuit. In the next sections

we will show how to find dispersing circuits. Our strategy for finding speedups will be to start with a unitary circuit U which acts on n qubits and has size polynomial in n. We will then try to find an oracle for which U efficiently solves the corresponding oracle identification problem. Next we need to define a state $|\varphi_a\rangle$ that can be prepared with $O(1)$ oracle calls and has $\Omega(1)$ overlap with $U^\dagger|a\rangle$. This is accomplished by letting $|\varphi_a\rangle$ be a state of the form $2^{-n/2}\sum_x \pm|x\rangle$. We can prepare $|\varphi_a\rangle$ with only two oracle calls (or one, depending on the model), but to guarantee that $|\langle a|U|\varphi_a\rangle|$ can be made large, we will need an additional condition on U. For any $a \in A$, $U^\dagger|a\rangle$ should have amplitude that is mostly spread out over the entire computational basis. When this is the case, we say that U is *dispersing*. The precise definition is as follows:

Definition 7. *Let U be a quantum circuit on n qubits. For $0 < \alpha, \beta \leq 1$, we say that U is (α, β)-dispersing if there exists a set $A \subseteq \{0,1\}^n$ with $|A| \geq 2^{\alpha n}$ and*

$$\sum_{x \in \{0,1\}^n} |\langle a|U|x\rangle| \geq \beta 2^{\frac{n}{2}}. \tag{3}$$

for all $a \in A$.

Note that the LHS of (3) can also be interpreted as the L_1 norm of $U^\dagger|a\rangle$.

The speedup in [BV97] uses $U = H^{\otimes n}$, which is (1,1)-dispersing since $\sum_x |\langle a|H^{\otimes n}|x\rangle| = 2^{n/2}$ for all a. Similarly the QFT over the cyclic group is (1,1)-dispersing.[3] Nonabelian QFTs do not necessarily have the same strong dispersing properties, but they satisfy a weaker definition that is still sufficient for a quantum speedup. Suppose that the measurement operator is instead defined as $\Pi_a = U(|a\rangle\langle a| \otimes I)U^\dagger$, where a is a string on m bits and I denotes the identity operator on $n-m$ bits. Then U still permits oracle identification, but our requirements that U be dispersing are now relaxed. Here, we give a definition that is loose enough for our purposes, although further weakening would still be possible.

Definition 8. *Let U be a quantum circuit on n qubits. For $0 < \alpha, \beta \leq 1$ and $0 < m \leq n$, we say that U is (α, β)-pseudo-dispersing if there exists a set $A \subseteq \{0,1\}^m$ with $|A| \geq 2^{\alpha n}$ such that for all $a \in A$ there exists a unit vector $|\psi\rangle \in \mathbb{C}^{2^{n-m}}$ such that*

$$\sum_{x \in \{0,1\}^n} |\langle a|\langle \psi|U|x\rangle| \geq \beta 2^{\frac{n}{2}}. \tag{4}$$

This is a weaker property than being dispersing, meaning that any (α, β)-dispersing circuit is also (α, β)-pseudo-dispersing.

We can now state our basic constant vs. linear query separation.

Theorem 9. *If U is (α, β)-pseudo-dispersing, then there exists an oracle problem which can be solved with one query, one use of U and success probability $(2\beta/\pi)^2$. However, any classical randomized algorithm that succeeds with probability $\geq \delta$ must use $\geq \alpha n + \log \delta$ queries.*

[3] Another possible way to generalize [BV97] is to consider other unitaries of the form $U = A^{\otimes n}$, for $A \in \mathcal{U}_2$. However, it is not hard to show that the only way for such a U to be $(\Omega(1), \Omega(1))$-dispersing is for A to be of the form $e^{i\phi_1\sigma_z}He^{i\phi_2\sigma_z}$.

Before we prove this Theorem, we state a Lemma about how well states of the form $2^{-n/2}\sum_x e^{i\phi_x}|x\rangle$ can be approximated by states of the form $2^{-n/2}\sum_x \pm|x\rangle$.

Lemma 10. *For any vector* $(x_1,\ldots,x_d)\in\mathbb{C}^d$ *there exists* $(\theta_1,\ldots,\theta_d)\in\{\pm1\}^d$ *such that*

$$\left|\sum_{k=1}^{d} x_k\theta_k\right| \geq \frac{2}{\pi}\sum_{k=1}^{d}|x_k|\,.$$

The proof is in the full version of the paper[HH08].

Proof of Theorem 9: Since U is (α,β)-pseudo-dispersing, there exists a set $A\subset\{0,1\}^m$ with $|A|\geq 2^{\alpha n}$ and satisfying (4) for each $a\in A$. The problem will be to determine a by querying an oracle $\mathcal{O}_a(x)$. No matter how we define the oracle, as long as it returns only one bit per call any classical randomized algorithm making q queries can have success probability no greater than $2^{q-\alpha n}$ (or else guessing could succeed with probability $>2^{-\alpha n}$ without making any queries). This implies the classical lower bound.

Given $a\in A$, to define the oracle $\mathcal{O}_a$, first use the definition to choose a state $|\psi\rangle$ satisfying (4). Then by Lemma 10 (below), choose a vector $\boldsymbol{\theta}$ that (when normalized to $|\theta\rangle$) will approximate the state $U^\dagger|a\rangle|\psi\rangle$. Define $\mathcal{O}_a(x)$ so that $(-1)^{\mathcal{O}_a(x)}=\theta_x=2^{n/2}\langle x|\theta\rangle$. By construction,

$$2^{-n/2}|\langle a|\langle\psi|U|\theta\rangle| \geq \frac{2}{\pi}\beta \tag{5}$$

which implies that creating $|\theta\rangle$, applying U, and measuring the first register has probability $\geq(2\beta/\pi)^2$ of yielding the correct answer a. □

5 Any Quantum Fourier Transform Is Pseudo-dispersing

In this section we start with some special cases of dispersing circuits by showing that any Fourier transform is dispersing. In the next section we show that most circuits are dispersing.

The original RFS paper [BV97] used the fact that $H^{\otimes n}$ is (1,1)-dispersing to obtain their starting $O(1)$ vs $\Omega(n)$ separation. The QFT on the cyclic group (or any abelian group, in fact) is also (1,1)-dispersing. In fact, if we will accept a pseudo-dispersing circuit, then any QFT will work:

Theorem 11. *Let* G *be a group with irreps* $\hat{G}$ *and* d_λ *denoting the dimension of irrep* λ. *Then the Fourier transform over* G *is* $(\alpha,1/\sqrt{2})$*-pseudo-dispersing, where* $\alpha=(\log\sum_\lambda d_\lambda)/\log|G|\geq 1/2$.

Via Theorem 9 and Theorem 2, this implies that any QFT can be used to obtain a superpolynomial quantum speedup. For most nonabelian QFTs, this is the first example of a problem which they can solve more quickly than a classical computer.

Proof (Proof of Theorem 11). Let $A=\{(\lambda,i):\lambda\in\hat{G},i\in\{1,\ldots,d_\lambda\}\}$.

Let V_λ denote the representation space corresponding to an irrep $\lambda \in \hat{G}$. The Fourier transform on G maps vectors in $\mathbb{C}[G]$ to superpositions of vectors of the form $|\lambda\rangle|v_1\rangle|v_2\rangle$ for $|v_1\rangle, |v_2\rangle \in V_\lambda$.

Fix a particular choice of λ and $|i\rangle \in V_\lambda$. If U denotes the QFT on G then let

$$\rho = U^\dagger \left(|\lambda\rangle\langle\lambda| \otimes |i\rangle\langle i| \otimes \frac{I_{V_\lambda}}{d_\lambda} \right) U.$$

Define $V := \operatorname{supp} \rho$, and let $\mathbf{E}_{|\psi\rangle \in V}$ denote an expectation over $|\psi\rangle$ chosen uniformly at random from unit vectors in V.[4] Finally, let Π be the projector onto V. Note that $\rho = \Pi/d_\lambda = \mathbf{E}\,|\psi\rangle\langle\psi|$.

Because of the invariance of ρ under right-multiplication by group elements (i.e. $\langle g_1|\rho|g_2\rangle = \langle g_1 h|\rho|g_2 h\rangle$ for all $g_1, g_2, h \in G$), we have for any g that

$$\langle g|\rho|g\rangle = \frac{1}{|G|} \sum_h \langle gh|\rho|gh\rangle = \frac{1}{|G|} \operatorname{tr}(\rho) = \frac{1}{|G|}. \tag{6}$$

Since $\mathbf{E}\,|\psi\rangle\langle\psi| = \rho$, (6) implies that

$$\mathop{\mathbf{E}}_{|\psi\rangle \in V} |\langle g|\psi\rangle|^2 = \langle g|\rho|g\rangle = \frac{1}{|G|}.$$

Next, we would like to analyze $\mathbf{E}\,|\langle g|\psi\rangle|^4$.

$$\mathop{\mathbf{E}}_{|\psi\rangle} |\langle g|\psi\rangle|^4 = \mathop{\mathbf{E}}_{|\psi\rangle} \operatorname{tr}\left(|g\rangle\langle g| \otimes |g\rangle\langle g|\right) \cdot \left(|\psi\rangle\langle\psi| \otimes |\psi\rangle\langle\psi|\right) \tag{7}$$

$$= \operatorname{tr}\left(|g\rangle\langle g| \otimes |g\rangle\langle g|\right) \frac{I + \text{SWAP}}{d_\lambda(d_\lambda + 1)} (\Pi \otimes \Pi) \tag{8}$$

$$\leq \operatorname{tr}\left(|g\rangle\langle g| \otimes |g\rangle\langle g|\right) \cdot (I + \text{SWAP})(\rho \otimes \rho) \tag{9}$$

$$= 2(\langle g|\rho|g\rangle)^2 = \frac{2}{|G|^2} \tag{10}$$

To prove the equality on the second line, we use a standard representation-theoretic trick (cf. section V.B of [PSW06]). First note that $|\psi\rangle^{\otimes 2}$ belongs to the symmetric subspace of $V \otimes V$, which is a $\frac{d_\lambda(d_\lambda+1)}{2}$-dimensional irrep of $\mathcal{U}_{d_\lambda}$. Since $\mathbf{E}_{|\psi\rangle} |\psi\rangle\langle\psi|^{\otimes 2}$ is invariant under conjugation by $u \otimes u$ for any $u \in \mathcal{U}_{d_\lambda}$, it follows that $\mathbf{E}_{|\psi\rangle} |\psi\rangle\langle\psi|^{\otimes 2}$ is proportional to a projector onto the symmetric subspace of $V^{\otimes 2}$. Finally, $\text{SWAP}\Pi^{\otimes 2}$ has eigenvalue 1 on the symmetric subspace of $V^{\otimes 2}$ and eigenvalue -1 on its orthogonal complement, the antisymmetric subspace of $V^{\otimes 2}$. Thus, $\frac{I+\text{SWAP}}{2}\Pi^{\otimes 2}$ projects onto the symmetric subspace and we conclude that

$$\mathop{\mathbf{E}}_{|\psi\rangle} |\psi\rangle\langle\psi|^{\otimes 2} = \frac{(I + \text{SWAP})(\Pi \otimes \Pi)}{d_\lambda(d_\lambda + 1)}.$$

[4] We can think of $|\psi\rangle$ either as the result of applying a Haar uniform unitary to a fixed unit vector, or by choosing $|\psi'\rangle$ from any rotationally invariant ensemble (e.g. choosing the real and imaginary part of each component to be an i.i.d. Gaussian with mean zero) and setting $|\psi\rangle = |\psi'\rangle/\sqrt{\langle\psi'|\psi'\rangle}$.

Now we note the inequality

$$\mathbf{E}\,|Y| \geq (\mathbf{E}\,Y^2)^{\frac{3}{2}}/(\mathbf{E}\,Y^4)^{\frac{1}{2}}, \tag{11}$$

which holds for any random variable Y and can be proved using Hölder's inequality [Ber97]. Setting $Y = |\langle g|\psi\rangle|$, we can bound $\mathbf{E}_{|\psi\rangle}\,|\langle g|\psi\rangle| \geq 1/\sqrt{2|G|}$. Summing over G, we find

$$\mathop{\mathbf{E}}_{|\psi\rangle} \sum_{g\in G} |\langle g|\psi\rangle| \geq \frac{1}{\sqrt{2}}\sqrt{|G|}.$$

Finally, because this last inequality holds in expectation, it must also hold for at least some choice of $|\psi\rangle$. Thus there exists $|\psi\rangle \in V$ such that

$$\sum_{g\in G} |\langle g|\psi\rangle| \geq \frac{1}{\sqrt{2}}\sqrt{|G|}.$$

Then U satisfies the pseudo-dispersing condition in (4) for the state $|\psi\rangle$ with $\beta = 1/\sqrt{2}$.

This construction works for each $\lambda \in \hat{G}$ and for $|v_1\rangle$ running over any choice of basis of V_λ. Together, this comprises $\sum_{\lambda\in\hat{G}} d_\lambda$ vectors in the set A.

6 Most Circuits Are Dispersing

Our final, and most general, method of constructing dispersing circuits is simply to choose a polynomial-size random circuit. We define a length-t random circuit to consist of performing the following steps t times.

1. Choose two distinct qubits i, j at random from $[n]$.
2. Choose a Haar-distributed random $U \in \mathcal{U}_4$.
3. Apply U to qubits i and j.

A similar model of random circuits was considered in [DOP07]. Our main result about these random circuits is the following Theorem.

Theorem 12. *For any $\alpha, \beta > 0$, there exists a constant C such that if U is a random circuit on n qubits of length $t = Cn^3$ then U is (α, β)-dispersing with probability*

$$\geq 1 - \frac{2\beta^2}{1 - 2^{-n(1-\alpha)}}.$$

Theorem 12 is proved in the extended version of this paper[HH08]. The idea of the proof is to reduce the evolution of the fourth moments of the random circuit (i.e. quantities of the form $\mathbf{E}_U \operatorname{tr} UM_1U^\dagger M_2 U M_3 U^\dagger M_4$) to a classical Markov chain, using the approach of [DOP07]. Then we show that this Markov chain has a gap of $\Omega(1/n^2)$, so that circuits of length $O(n^3)$ have fourth moments nearly identical to those of Haar-uniform unitaries from $\mathcal{U}_{2^n}$. Finally, we use (11), just as we did for quantum Fourier transforms, to show that a large fraction of inputs are likely to be mapped to states with large L_1-norm. This will prove Theorem 12 and show that superpolynomial quantum speedups can be built by plugging almost any circuit into the recursive framework we describe in Section 3.

References

[Aar03] Aaronson, S.: Quantum lower bound for recursive Fourier sampling. Quantum Information and Computation 3(2), 165–174 (2003)

[AJL06] Aharonov, D., Jones, V., Landau, Z.: A polynomial quantum algorithm for approximating the jones polynomial. In: STOC 2006: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 427–436. ACM Press, New York (2006)

[BCvD05] Bacon, D., Childs, A.M., van Dam, W.: From optimal measurement to efficient quantum algorithms for the hidden subgroup problem over semidirect product groups. In: FOCS 2005: 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 469–478 (2005)

[Bea97] Beals, R.: Quantum computation of Fourier transforms over symmetric groups. In: STOC 1997: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, El Paso, Texas, May 4–6, 1997, pp. 48–53. ACM Press, New York (1997)

[Ber97] Berger, B.: The fourth moment method. Siam J. Comp. 26(4), 1188–1207 (1997)

[BV97] Bernstein, E., Vazirani, U.: Quantum complexity theory. Siam J. Comp. 26(5), 1411–1473 (1997)

[DOP07] Dahlsten, O.C.O., Oliveira, R., Plenio, M.B.: Emergence of typical entanglement in two-party random processes. J. Phys. A 40, 8081–8108 (2007)

[FIM+03] Friedl, K., Ivanyos, G., Magniez, F., Santha, M., Sen, P.: Hidden translation and orbit coset in quantum computing. In: STOC 2003: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, San Diego, CA, pp. 1–9. ACM Press, New York (2003)

[FvdG99] Fuchs, C.A., van de Graaf, J.: Cryptographic distinguishability measures for quantum mechanical states. IEEE Trans. Inf. Th. 45(4), 1216–1227 (1999)

[Hal07] Hallgren, S.: Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. Journal of the ACM 54(1), 1–19 (2007)

[HH08] Hallgren, S., Harrow, A.W.: Superpolynomial speedups based on almost any quantum circuit (2008)

[HMR+06] Hallgren, S., Moore, C., Rötteler, M., Russell, A., Sen, P.: Limitations of quantum coset states for graph isomorphism. In: STOC 2006: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 604–617. ACM Press, New York (2006)

[IMS01] Ivanyos, G., Magniez, F., Santha, M.: Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. In: Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, Heraklion, Crete Island, Greece, pp. 263–270 (2001)

[PSW06] Popescu, S., Short, A.J., Winter, A.: The foundations of statistical mechanics from entanglement: Individual states vs. averages. Nature 2, 754–758 (2006)

[Sho97] Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. Siam J. Comp. 26(5), 1484–1509 (1997)

[Sim97] Simon, D.R.: On the power of quantum computation. Siam J. Comp. 26(5), 1474–1483 (1997)

[vDHI03] van Dam, W., Hallgren, S., Ip, L.: Quantum algorithms for some hidden shift problems. In: SODA 2003: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD (2003)

[Wat01] Watrous, J.: Quantum algorithms for solvable groups. In: STOC 2001: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, Crete, Greece, pp. 60–67. ACM Press, New York (2001)

The Speed of Convergence in Congestion Games under Best-Response Dynamics*

Angelo Fanelli, Michele Flammini, and Luca Moscardelli

Department of Computer Science
University of L'Aquila
Via Vetoio, Coppito 67100 L'Aquila
{angelo.fanelli,flammini,moscardelli}@di.univaq.it

Abstract. We investigate the speed of convergence of congestion games with linear latency functions under best response dynamics. Namely, we estimate the social performance achieved after a limited number of rounds, during each of which every player performs one best response move. In particular, we show that the price of anarchy achieved after k rounds, defined as the highest possible ratio among the total latency cost, that is the sum of all players latencies, and the minimum possible cost, is $O(\sqrt[2^k-1]{n})$, where n is the number of players. For constant values of k such a bound asymptotically matches the $\Omega(\sqrt[2^k-1]{n}/k)$ lower bound that we determine as a refinement of the one in [7]. As a consequence, we prove that order of $\log\log n$ rounds are not only necessary, but also sufficient to achieve a constant price of anarchy, i.e. comparable to the one at Nash equilibrium. This result is particularly relevant, as reaching an equilibrium may require a number of rounds exponential in n. We then provide a new lower bound of $\Omega(\sqrt[2^k-1]{n})$ for load balancing games, that is congestion games in which every strategy consists of a single resource, thus showing that a number of rounds proportional to $\log\log n$ is necessary and sufficient also under such a restriction.

Our results thus solve the important left open question of the polynomial speed of convergence of linear congestion games to constant factor solutions.

1 Introduction

Congestion games constitute a well-known class of non-cooperative games in which a set of facilities E is available to the players and the strategy set of each player i can be any $S_i \subseteq 2^E$. The cost of each facility $e \in E$ (usually called the latency of e) is given by a function f_e of the number of players using e and the latency experienced by each player i is the sum of the latencies of all the facilities used by i. The social or global cost function of the games is given by the sum of the latencies perceived by all the players.

* This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 796–807, 2008.

Congestion games have been introduced by Rosenthal [18] in 1973. By defining an elegant potential function he showed that they always possess (and always converge to) pure Nash equilibria [17]. Since then, different variations of the original model have been proposed (see for instance [15]) and, in the last decade, they have come across the analysis of the Computer Science community with the purpose of characterizing the complexity of computing their pure Nash equilibria [10] and evaluating their suboptimality in terms of price of anarchy [14], corresponding to the worst case cost ratio between the social cost at Nash equilibrium and the one of an optimal solution.

One of the most interesting and studied special cases is the class of the linear congestion games and its special case of load balancing games, in which the latency of each resource e is defined as a linear function of the number of players using e. Recent contributions [2,4,6] have fixed their price of anarchy to 5/2.

Often Nash equilibria may not exist or it may be hard to compute them or the time for convergence to Nash equilibria may be extremely long, even if the players always choose *best response moves*, i.e. moves providing them the smallest possible cost. Thus, recent research effort [16] concentrated in the evaluation of the speed of convergence (or non-convergence) to an equilibrium in terms of *rounds*, in which every player performs one best-response move. A k-round walk with $k \geq 1$ is defined as the concatenation of k rounds. An important issue raised by the authors is then that of evaluating the loss of social performance in selfish evolutions with a (polinomially) bounded number of rounds, not necessary terminating in a Nash equilibrium. On this line of research, several results have been obtained for different non-cooperative games [7,11,12,13,16].

Such an investigation is particular relevant for congestion games, as in [10] it has been proven that determining one of their Nash equilibrium is a PLS-complete problem [9], even for linear latency functions and in the symmetric case in which all the players have the same strategy set. As a consequence, a number of rounds (and more in general of moves) exponential in the number of players may be required to reach an equilibrium.

Starting from such a negative result, in [1] the authors outlined suitable structural properties of congestion games sufficient to guarantee a convergence in polynomial number of best responses, leading to the so-called class of the *matroid* congestion games.

Moreover, in [5] ϵ-Nash equilibria have been concerned, that is solutions in which every player cannot improve her cost by a multiplicative factor greater than ϵ. In particular, it has been shown that if players perform ϵ-bounded moves, that is moves reducing their latency cost at least ϵ times, the convergence of symmetric congestion games requires a number of steps polynomial in the number of players and ϵ^{-1}. While in the asymmetric case the convergence can still be exponential [19], recently in [3] the authors have shown that ϵ-Nash equilibria can still be reached in a polynomial number of steps under the weak conditions that the latency functions satisfy the so-called "γ-bounded jump" condition, i.e. $f_e(i+1) \leq \gamma \cdot f_e(i)$ for every $i \geq 1$, and every player moves at least once every fixed number of steps.

Finally, concerning linear congestion games, in [7] the authors showed a $\Theta(n)$ price of anarchy after one round, and a lower bound of $\sqrt[2^{O(k)}]{n}/k$ after k rounds, where n is the number of players. However, while the convergence to a constant price of anarchy in a polynomial number of random rounds can be inferred directly from the sink equilibria results of [13], to the best of our knowledge the same result for deterministic rounds is still open. To this aim, while the $\sqrt[2^{O(k)}]{n}/k$ lower bound of [7] implies that a number of rounds at least proportional to $\log\log n$ is necessary, a corresponding upper bound is missing and hopefully would also imply a sufficient low number of rounds.

In this paper we show that the price of anarchy achieved after k rounds is $O\left(\sqrt[2^k-1]{n}\right)$ and we refine the lower bound of [7] to $\Omega\left(\sqrt[2^k-1]{n}/k\right)$, which is asymptotically matching for constant values of k. As a consequence, we prove that $\log\log n$ rounds are not only necessary, but even sufficient to achieve a constant price of anarchy, i.e., comparable to the one at Nash equilibrium. We then provide a new lower bound of $\Omega\left(\sqrt[2^k-1]{n}\right)$ for load balancing games, that is congestion games in which every strategy consists of a single resource, thus showing that a number of rounds proportional to $\log\log n$ is necessary and sufficient also under such a restriction.

Our results close the important left open question of the speed of convergence to constant factor solutions in congestion games by deterministic rounds, which indeed we prove to be very fast.

2 Definitions and Preliminaries

A *strategic game* is defined by a tuple $\mathcal{G} = (N, (\Sigma_i)_{i\in N}, (c_i)_{i\in N})$, where $N = \{1, 2, \ldots, n\}$ denotes the set of the players, Σ_i a set of (pure) strategies for player i and $c_i : \times_{i\in N}\Sigma_i \mapsto \mathbb{R}_+ \cup \{0\}$ is the cost function for player i.

Let $\Sigma = \times_{i\in N}\Sigma_i$ be the *strategy profile set* or *state set* of the game and $S = (s_1, s_2, \ldots, s_n) \in \Sigma$ be a generic state in which each player i chooses strategy $s_i \in \Sigma_i$. Given the strategy profile $S = (s_1, s_2, \ldots, s_n)$ and a strategy $s'_i \in S_i$, let $(S \oplus s'_i) = (s_1, s_2, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_n)$, i.e., the strategy profile obtained from S if player i changes her strategy from s_i to s'_i.

In a *non-cooperative* strategic game we assume that each player acts selfishly and aims at choosing the strategy lowering its cost, given the strategy choices of other players. For a strategy profile $S = (s_1, s_2, \ldots, s_n)$, an *improvement move* of player i is a strategy s'_i such that $c_i(S \oplus s'_i) < c_i(S)$. Furthermore, a *best response move* of player i in S is a strategy $s^b_i \in S_i$ yielding the minimum possible cost, given the strategy choices of the other players, i.e., $c_i(S \oplus s^b_i) \leq c_i(S \oplus s'_i)$ for any other strategy $s'_i \in S_i$. Notice that a best response move corresponding to the strategy currently played in S by the involved player is not an improvement move.

A (pure) *Nash equilibrium* is a strategy profile in which every player plays her best response. Formally, $S = (s_1, s_2, \ldots, s_n)$ is a Nash Equilibrium if for all $i \in N$ and for any strategy $s'_i \in S_i$, $c_i(S) \leq c_i(S \oplus s'_i)$.

We are interested in evaluating the *social cost* or *social value* at equilibrium. More precisely, the social cost $C(S)$ of a given state S is defined as the sum

of all the players' costs, i.e., $C(S) = \sum_{i \in N} c_i(S)$. The *optimal* strategy profile $S^* = (s_1^*, s_2^*, \ldots, s_n^*)$ is one with minimum social value, that we denote by Opt. The *price of anarchy* $\mathrm{PoA}(\mathcal{G})$ of a game $\mathcal{G}$ is the worst case ratio between the social cost of a Nash equilibrium and Opt.

The selfish behavior of players can be modelled by the *(Best-Response) Nash Dynamics Graph*. Formally the Nash Dynamics Graph associated to a non-cooperative strategic game $\mathcal{G}$ is a directed graph $\mathcal{B} = (V, A)$ where each vertex in V corresponds to a strategy profile and there is an edge $(S, S') \in A$ with label i, where $S' = S \oplus s_i'$ and $s_i' \in \Sigma_i$, if and only if both the following conditions are met: (i) s_i' is a best response move of i in S; (ii) if $S \neq S'$, s_i' is also an improvement move of i in S. Observe that $\mathcal{B}$ may contain loops. A *best response walk* is a directed walk in $\mathcal{B}$. A player i plays its best response in a best response walk W if at least one edge in W has label i.

Given a best response walk in $\mathcal{B}$ starting from an arbitrary state, we are interested in the social value of its final state. In particular we aim at bounding the social value of a state obtained starting from an arbitrary strategy profile after a *fair* best response walk. To capture the notion of fairness, the following notions of best response walks have been considered in the literature [7,16]:

1-round walk : it's a best response walk $R = \left(\left(S_R^0, S_R^1\right), \left(S_R^1, S_R^2\right), \ldots, \left(S_R^i, S_R^{i+1}\right), \ldots, \left(S_R^{n-1}, S_R^n\right)\right)$ in $\mathcal{B}$ of length n, where the edge (S_R^i, S_R^{i+1}) has label $\pi_R(i)$ for every $0 \leq i \leq n-1$ and $\pi_R : N \mapsto \{1, 2, \ldots, n\}$ is an arbitrary ordering (permutation) of the players. S_R^0 is said the *initial* state of R and S_R^n its *final* state. For simplicity we denote R by a sequence of states, i.e., $R = \left(S_R^0, S_R^1, \ldots, S_R^n\right)$.

k-round walk : it's a best response walk $W = \langle R_1, R_2, \ldots, R_k \rangle$ in $\mathcal{B}$ corresponding to a sequence of k 1-round walks, i.e. such that each R_i is a 1-round walk in $\mathcal{B}$.

For the sake of simplicity, in the sequel we will often denote a round R_j of a walk W as $\left(S_j^0, S_j^1, \ldots, S_j^n\right)$, where S_j^i is the intermediate state $S_{R_j}^i$ of R_j. When clear from the context, we will drop the index R from the notation, writing $\left(S^0, S^1, \ldots, S^n\right)$ and S^i instead of $\left(S_R^0, S_R^1, \ldots, S_R^n\right)$ and S_R^i, respectively; moreover, we will assume $S^i = \left(s_1^i, \ldots, s_n^i\right)$.

Extending the classical definition, we let the *price of anarchy* yielded by k-round walks (denoted by $\mathrm{PoA}_k(\mathcal{G})$) be the worst case ratio among the social value of the last state of a k-round walk and Opt.

Congestion Games. A *congestion game* $\mathcal{G} = (N, E, (\Sigma_i)_{i \in N}, (f_e)_{e \in E}, (c_i)_{i \in N})$ is a non-cooperative strategic game characterized by the existence of a set E of *resources* to be shared by the players in N. Any (pure) strategy $s_i \in \Sigma_i$ of player i is a subset of resources, i.e., $\Sigma_i \subseteq 2^E$. Given a strategy profile $S = (s_1, s_2, \ldots, s_n)$ and a resource e, the number of players using e in S, called the *congestion* on e, is denoted by $n_e(S) = |\{i \in N \mid e \in s_i\}|$. A latency function $f_e : \mathbb{N} \mapsto \mathbb{R}_+$ associates to resource e a cost (latency) depending on the number of players currently using e, so that the cost of player i for the pure strategy s_i, depending on the congestion

of each resource in s_i, is given by $c_i(S) = \sum_{e \in S_i} f_e(n_e(S))$. Correspondingly, the social value of S is $C(S) = \sum_{i \in N} c_i(S) = \sum_{i \in N} \sum_{e \in S_i} f_e(n_e(S))$.

We denote by $E^* \subseteq E$ the set of resources used at a given optimal strategy profile S^*, i.e. $E^* = \bigcup_{i \in N} s_i^*$, and by o_e the number of players using resource e in S^*, i.e. $o_e = |\{i \in N \mid e \in s_i^*\}|$. Moreover, we refer to *singleton congestion* or *load balancing* games as the games in which all of the players' strategies consist of only a single resource, i.e., $\Sigma_i \subseteq E$.

In this paper we will focus on *linear* congestion games, that is having linear latency functions with nonnegative coefficients. More precisely, for every resource $e \in E$, $f_e(x) = a_e x + b_e$ with $a_e, b_e \geq 0$.

3 Upper Bound

In this section we bound the social cost after a k-round walk in a linear congestion game starting from an arbitrary state. All the results hold for linear congestion games having latency functions $f_e(x) = a_e x + b_e$ with $a_e, b_e \geq 0$ for every $e \in E$. Without loss of generality, we assume that $a_e = 1$ and $b_e = 0$ for every $e \in E$. In fact, given a congestion game $\mathcal{G}$ having latency functions $f_e(x) = a_e x + b_e$ with integer coefficient $a_e, b_e \geq 0$, it is possible to obtain an equivalent congestion game $\mathcal{G}'$, having the same set of players and identical latency functions $f(x) = x$ in the following way. For each resource e in $\mathcal{G}$, we include in $\mathcal{G}'$ a set A_e of a_e resources and n sets $B_e^1, \ldots, B_e^n$, each containing b_e resources; moreover, given any strategy set $s_i \in \Sigma_i$ in $\mathcal{G}$, $i = 1, \ldots, n$, we build a corresponding strategy set $s_i' \in \Sigma_i'$ (in $\mathcal{G}'$) by including in s_i', for each $e \in s_i$, all the resources in the sets A_e and B_e^i. If a_e and b_e are not integers we can perform a similar reduction by exploiting a simple scaling argument.

The following technical lemma will be useful in the sequel.

Lemma 1. *Given $\gamma \in \mathbb{R}_+$ and $\overrightarrow{d} = (d_1, d_2, \ldots, d_m) \in \mathbb{N}_+^m$, the maximum value of $\sum_{i=1}^m d_i x_i$ such that $\sum_{i=1}^m \frac{x_i^2}{2} \leq \gamma$ is $\sqrt{2\gamma \sum_{i=1}^m d_i^2}$.*

Let $R = \left(S^0, S^1, S^2, \ldots, S^n\right)$ be a 1-round walk and π_R be the moving ordering of the players in R. In the following we will often consider the *immediate* costs of players during R, that is the cost $c_{\pi_R(i)}(S^i)$ they experience right after having performed their best move. Clearly, given the optimal strategy profile S^*, since the i-th moving player $\pi_R(i)$ before moving can always select the strategy she would use in S^*, her immediate cost can be suitably upper bounded as $\sum_{e \in s^*_{\pi_R(i)}} \left(n_e(S^{i-1}) + 1\right)$, so that $\sum_{i=1}^n c_{\pi_R(i)}(S^i) \leq \sum_{i=1}^n \sum_{e \in s^*_{\pi_R(i)}} \left(n_e(S^{i-1}) + 1\right)$. Such an upper bound on the sum of the immediate costs in the sequel will be succinctly represented by the proportional quantity $\rho(R) = \frac{1}{\text{OPT}} \sum_{i=1}^n \sum_{e \in s^*_{\pi_R(i)}} \left(n_e(S^{i-1}) + 1\right)$, that is obtained dividing by OPT.

The following lemma shows that the social value at the end of round R is proportional to $\rho(R)$.

Lemma 2. *Given a 1-round walk R, $C(S^n) \leq 2\rho(R) \cdot \text{OPT}$.*

According to the above lemma, in order to prove our result, it is crucial to show that $\rho(R)$ fast decreases between two successive rounds.

Lemma 3. *Given a 2-round walk $\langle \overline{R}, R \rangle$, $\rho(R) \leq \frac{\alpha}{\alpha-1}\sqrt{2\rho(\overline{R})} + \frac{\alpha(\alpha+1)}{\alpha-1}$ for any $\alpha > 1$.*

Proof. For the sake of simplicity let us denote all the states $S^j_{\overline{R}}$ and S^j_R simply as $\overline{S}^j$ and S^j, respectively. Moreover, without loss of generality, let us assume that $\pi_R(i) = i$ for any $i \in 1, 2, \ldots, n$ and let $S^i = (s^n_1, s^n_2, \ldots, s^n_i, s^0_{i+1}, \ldots, s^0_n)$ be a generic intermediate state of R. The initial state S^0 of round R coincides with the final state $\overline{S}^n$ of round $\overline{R}$.

Given a state S, define $H(S) = \sum_{e \in E^*} n_e(S) o_e$. Consider the last state of round $\overline{R}$, i.e. S^0. In order to prove the claim, it is sufficient to determine suitable upper and lower bounds for $H(S^0)$ with respect to $\rho(\overline{R})$ and $\rho(R)$, respectively.

Let us first upper bound $H(S^0)$ with respect to $\rho(\overline{R})$. Given a resource $e \in E$ with congestion $n_e(S_0)$ at the end of round $\overline{R}$, by the definition of $\rho(\overline{R})$ and since for every integer i such that $1 \leq i \leq n_e(S^0)$ there must exist a player whose immediate cost on e paid during $\overline{R}$ is at least i, $\rho(\overline{R})\text{OPT} \geq \sum_{i=1}^{n} c_{\pi_{\overline{R}}(i)}(\overline{S}^i) \geq \sum_{e \in E^*} \sum_{i=1}^{n_e(S^0)} i \geq \sum_{e \in E^*} \frac{(n_e(S^0))^2}{2}$. Therefore, by applying Lemma 1 with $m = |E^*|$, $\gamma = \rho(\overline{R})\text{OPT}$, and $(d_1, \ldots, d_{|E^*|})$ and $(x_1, \ldots, x_{|E^*|})$ the vectors containing the o_e and $n_e(S^0)$ values for every $e \in E^*$, respectively, since $\sum_{e \in E^*} o_e^2 = \text{OPT}$, we obtain

$$H(S^0) \leq \sqrt{2\rho(\overline{R})\text{OPT} \sum_{e \in E^*} o_e^2} = \sqrt{2\rho(\overline{R})}\text{OPT}. \tag{1}$$

In order to lower bound $H(S^0)$ with respect to $\rho(R)$, we define the following suitable potential function $h_i(R) = \sum_{e \in E^*} n'_e(S^i) o^i_e$ for $i \in \{0, 1, \ldots, n-1\}$, where for a generic state S, $n'_e(S) = \max\{0,\ n_e(S) - \alpha o_e\}$ and $o^k_e = |\{j \in N \mid j > k \ \wedge \ e \in s^*_j\}|$. Informally speaking, such a potential function takes into account the congestion due to the not yet moving players during round R above a "virtual" congestion frontier given by all the values αo_e. While if players during round R would not choose any resource in E^* $H(S^0)$ would be lower bounded by $\rho(R)\text{OPT}$, the above potential function in bounding from below $h_0(R) \leq H(S^0)$ accounts also for the players selecting resources in E^*. Let $\Delta_i(R) = h_{i-1}(R) - h_i(R)$ for $i \in \{1, 2, \ldots, n\}$. Notice that by the definition of the potential function $h_i(R)$, since $h_n(R) = 0$, $\sum_{i=1}^{n} \Delta_i(R) = h_0(R) \leq H(S^0)$, that is a lower bound for $\sum_{i=1}^{n} \Delta_i(R)$ is also a lower bound for $H(S^0)$; therefore, in the following we focus on bounding $\sum_{i=1}^{n} \Delta_i(R)$. Consider a generic step i in round R, in which player i performs a best response move by selecting

resources in s_i^n. If i increases $n'_e(S^{i-1})$ for a given resource $e \in E^*$, that is $n'_e(S^i) = n'_e(S^{i-1}) + 1$, then $n_e(S^i) > \alpha o_e$, so that $n'_e(S^i) - n'_e(S^{i-1}) = 1 \leq \frac{n_e(S^i)}{\alpha o_e}$. If player i does not increase $n'_e(S^{i-1})$, that is $n'_e(S^i) - n'_e(S^{i-1}) = 0$, then trivially again $n'_e(S^i) - n'_e(S^{i-1}) = 0 \leq \frac{n_e(S^i)}{\alpha o_e}$. Therefore, since player i to obtain $h_i(R)$ removes at least $\sum_{e \in s_i^*} n'_e(S^{i-1})$ from $h_{i-1}(R)$ (due to the decrease of the coefficients o_e^{i-1} to o_e^i) and adds at most $\frac{n_e(S^i)}{\alpha o_e}$ to every resource $e \in s_i^n \cap E^*$, recalling that $\sum_{e \in s_i^n \cap E^*} n_e(S^i) = c_i(S^i) \leq \sum_{e \in s_i^*} \left(n_e(S^{i-1}) + 1\right)$,

$$\begin{aligned}
\Delta_i(R) &\geq \sum_{e \in s_i^*} n'_e(S^{i-1}) - \sum_{e \in s_i^n \cap E^*} \frac{n_e(S^i)}{\alpha o_e} o_e^i \\
&\geq \sum_{e \in s_i^*} n'_e(S^{i-1}) - \frac{1}{\alpha} \sum_{e \in s_i^n \cap E^*} n_e(S^i) \\
&\geq \sum_{e \in s_i^*} n'_e(S^{i-1}) - \frac{1}{\alpha} \sum_{e \in s_i^*} \left(n_e(S^{i-1}) + 1\right).
\end{aligned}$$

Since $n'_e(S^{i-1}) \geq n_e(S^{i-1}) - \alpha o_e$ and $o_e \geq 1$ for every $e \in E^*$, it follows that

$$\begin{aligned}
\Delta_i(R) &\geq \sum_{e \in s_i^*} \left(n_e(S^{i-1}) - \alpha o_e\right) - \frac{1}{\alpha} \sum_{e \in s_i^*} \left(n_e(S^{i-1}) + 1\right) \\
&\geq \left(1 - \frac{1}{\alpha}\right) \sum_{e \in s_i^*} n_e(S^{i-1}) - \left(\alpha + \frac{1}{\alpha}\right) \sum_{e \in s_i^*} o_e.
\end{aligned}$$

By summing up the values $\Delta_i(R)$, we obtain

$$\begin{aligned}
\sum_{i=1}^{n} \Delta_i(R) &\geq \left(1 - \frac{1}{\alpha}\right) \sum_{i \in N} \sum_{e \in s_i^*} n_e(S^{i-1}) - \left(\alpha + \frac{1}{\alpha}\right) \sum_{i \in N} \sum_{e \in s_i^*} o_e \\
&\geq \left(1 - \frac{1}{\alpha}\right) \rho(R) \textsc{Opt} - (1 + \alpha) \textsc{Opt},
\end{aligned}$$

and thus, since $H(S^0) \geq \sum_{i=1}^{n} \Delta_i(R)$,

$$H(S^0) \geq \left(1 - \frac{1}{\alpha}\right) \rho(R) \textsc{Opt} - (1 + \alpha) \textsc{Opt}. \tag{2}$$

Therefore, by combining inequalities (1) and (2),

$$\begin{aligned}
\rho(R) &\leq \frac{1}{\textsc{Opt}} \left(\sqrt{2\rho(\overline{R})} \textsc{Opt} + (\alpha + 1) \textsc{Opt}\right) \left(\frac{\alpha}{\alpha - 1}\right) \\
&\leq \frac{\alpha}{\alpha - 1} \sqrt{2\rho(\overline{R})} + \frac{\alpha(\alpha + 1)}{\alpha - 1},
\end{aligned}$$

hence the claim. □

We are now able to prove the following theorem.

Theorem 1. *For any linear congestion game* $\mathcal{G}$, $\mathrm{PoA}_k(\mathcal{G}) = O\left(\sqrt[2^{k-1}]{n}\right)$.

Proof. Given a k-round walk $W = \langle R_1, R_2, \ldots, R_k \rangle$, where each R_j is a 1-round walk, it follows from Lemma 3 that $\rho(R_j) \leq \frac{\alpha}{\alpha-1}\sqrt{2\rho(R_{j-1})} + \frac{\alpha(\alpha+1)}{\alpha-1}$ for any $\alpha > 1$. Applying such an argument to all the rounds of the walk, we obtain that $\rho(R_k) = O\left(\sqrt[2^{k-1}]{\rho(R_1)}\right)$. By Lemma 2, recalling that S_j^n denotes the final state of round R_j, we have $C(S_k^n) = O\left(\sqrt[2^{k-1}]{\rho(R_1)}\right)\mathrm{OPT}$.

By the definition of $\rho(R)$, since $\sum_{e\in E^*} o_e \leq \sum_{e\in E^*} o_e^2 = \mathrm{OPT}$, for any possible round R it holds that

$$\rho(R) = \frac{1}{\mathrm{OPT}}\sum_{i=1}^{n}\sum_{e\in s^*_{\pi_R(i)}}\left(n_e(S^{i-1})+1\right) \leq \frac{1}{\mathrm{OPT}}\sum_{i=1}^{n}\sum_{e\in s^*_{\pi_R(i)}}(n+1)$$

$$= \frac{n+1}{\mathrm{OPT}}\sum_{i=1}^{n}|s^*_{\pi_R(i)}| = \frac{n+1}{\mathrm{OPT}}\sum_{e\in E^*} o_e \leq n+1.$$

Therefore, $\rho(R_1) \leq n+1$ and we obtain $C(S_k^n) = O\left(\sqrt[2^{k-1}]{n}\right)\mathrm{OPT}$. □

The following corollary is an immediate consequence of Theorem 1.

Corollary 1. *For any linear congestion game* $\mathcal{G}$, $\mathrm{PoA}_{\log\log n}(\mathcal{G}) = O(1)$.

By numerical and optimization arguments, by choosing $\alpha \approx 3.69$, it turns that the price of anarchy after $\lceil \log\log n \rceil + 3$ rounds is lower than 28.

4 Lower Bound

In [7,8] the authors provide a lower bound to PoA_k equal to $\Omega\left(\sqrt[2^{2k+4}]{n}/k\right)$. By exploiting similar arguments, it is possible to prove the following theorem providing a more precise and tight bound. The proof will appear in the full version of the paper.

Theorem 2. *There exists a linear congestion game* $\mathcal{G}$ *such that* $\mathrm{PoA}_k(\mathcal{G}) = \Omega\left(\sqrt[2^{k-1}]{n}/k\right)$.

Moreover, we are able to provide an almost matching lower bound for the special case of linear singleton congestion games. To this aim, let us introduce the following definition.

Definition 1. *An* (h,q)-tree $T_{h,q}$ *(see Figure 1) is a rooted tree of height* $h-1$ *such that, denoting by* $p(v)$ *the parent of node* v *and by* $\delta(v)$ *the number of children of* v*:*

- ▷ $\delta(r) = q$*, where* r *is the root of* $T_{h,q}$*, i.e. the only node at level* 1*;*
- ▷ *every node* v *at level* i*,* $1 < i \leq h$*, being the* m*-th leftmost child of* $p(v)$ *is such that* $\delta(v) = m \cdot q^{2^{i-2}}$.

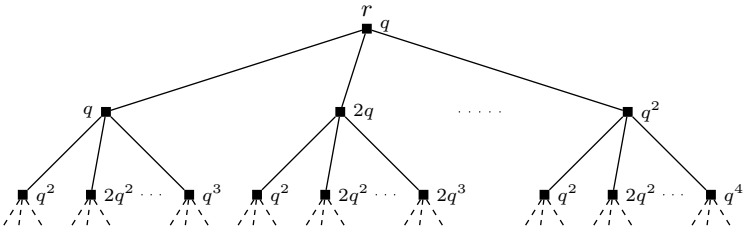


Fig. 1. An (h, q)-*tree* with the nodes labeled with the number of children

Given any tree T, let $L(T, j)$ be the set of nodes at level j and $Q(T, j)$ be the set of descendants of the rightmost child of the root of T at level j. Moreover let $P_j^L(T) = \sum_{u \in L(T,j)} \delta(u)$ and $P_j^Q(T) = \sum_{u \in Q(T,j)} \delta(u)$.

Lemma 4. *Given an* (h, q)-tree $T_{h,q}$ *and an arbitrary sub-tree* $\overline{T}$ *of* $T_{h,q}$ *rooted in* $\overline{r}$, $P_j^L(\overline{T}) \leq \frac{\delta(\overline{r})+1}{2} P_j^Q(\overline{T})$.

Lemma 5. *In every* (h, q)-tree $T_{h,q}$ *rooted at node* r *the number of nodes at level* j *is* $|L(T_{h,q}, j)| \leq q^{2^{j-1}-1} \left(\frac{1}{2} + \frac{1}{q}\right)^{j-2}$.

Theorem 3. *There exists a linear singleton congestion game* $\mathcal{G}$ *such that* $\mathrm{PoA}_k(\mathcal{G}) = \Omega\left(\sqrt[2^k-1]{n}\right)$.

Proof. In order to provide the lower bound instance, we associate to a tree T a singleton congestion game in which the resources correspond to the nodes of T and the players to the edges of T; each player (an edge of T) has two strategies corresponding to the resources being the nodes of T she connects. We refer to the resource corresponding to the node of lower level as the *red* strategy and to the one of higher level as the *green* strategy.

Let us consider a $(k+1, q)$-tree $\overline{T}$ rooted at r_1 with an additional node r_0 (of level 0) connected to r_1; in this way we obtain a new tree T rooted at r_0 and having height $k+1$ (see Figure 2).

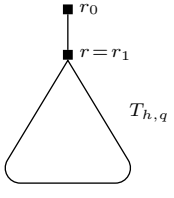


Fig. 2. The lower bound construction of Theorem 3

Letting $\xi_0 < \xi_1 < \cdots < \xi_{k+1}$ be arbitrary small values, the latency function of the resources are defined as follows:

- ▷ $f_{r_0}(x) = (q + \xi_0)\, x$;
- ▷ $f_{r_1}(x) = (1 + \xi_1)\, x$;
- ▷ for every level $j = 2, \ldots, k$ and every node $v \in L(\overline{T}, j)$, $f_v(x) = \left(\frac{1}{q^{2^{j-1}-1}} + \xi_j\right) x$;
- ▷ for every node $v \in L(\overline{T}, k+1)$, $f_v(x) = \xi_{k+1} \cdot x$.

In the following, we refer to the players corresponding to edges between levels j and $j+1$ as players of level j.

Let $\langle R_1, \ldots, R_k \rangle$ be a k-round walk starting from the initial state S_0 in which the players of level k, except the ones being first children of their parent nodes, use their red strategy, while all the other ones use their green strategy. Moreover, let S_j be the final state of round j. We want to show the following property: for every $j = 1, \ldots, k$, S_j is such that all the players of level $k - j$, except the ones being the first children of their parent nodes, use their red strategy, while all the other ones use their green strategy.

We consider that the players move in increasing order of level and for each level from left to right. We prove the claimed property by induction on $j = 0, 1, \ldots, k$. The base of the induction ($j = 0$) is trivially verified by the choice of the initial state. Let us assume the property true for a fixed $j \geq 0$; we want to show that it holds for $j + 1$.

- ▷ *Level i such that $i = 0, \ldots, k - j - 2$.* Since for every player of level i the latency of the resource belonging to the red strategy is greater than the latency of the resource belonging to the green strategy, and by the induction hypothesis the resource belonging to the green strategy of player i is not used by any player of level $i + 1$, all the players of level i select their green strategy as best response move.
- ▷ *Level $i = k - j - 1$.* By the induction hypothesis, all the players of level $k - j$, moving after those of level i, are using their red strategy. We consider a generic player of level i having as the resource of her green strategy a node v being the m-th child of its parent u (with $m \geq 2$). The number of players using v just before the player's move is $\delta(v) - 1$ (by recalling the definition of $(k+1, q)$-tree, $\delta(v) = m \cdot q^{2^{k-j-2}}$). Thus, she would suffer a latency equal to $f_v(\delta(v)) = m \cdot q^{2^{k-j-2}} \left(\frac{1}{q^{2^{k-j-1}-1}} + \xi_{k-j}\right)$ on her green strategy. Moreover, by assuming in an inductive way that all her siblings moving before her (except the first one) already selected their red strategy, she would experiment a latency equal to $f_u(m) = m \left(\frac{1}{q^{2^{k-j-2}-1}} + \xi_{k-j-1}\right)$ on her red strategy. Since $\xi_{k-j-1} < \xi_{k-j}$, $f_u(m) < f_v\left(\delta(v)\right)$; therefore, such a player chooses her red strategy.
- ▷ *Level $i = k - j$.* Since by the induction hypothesis the resource of the green strategies of players of level i are not used by any player, it is easy to check that every player of such a level makes an improving move by choosing her green strategy.

▷ *Level i such that $i = k - j + 1, \ldots, k$.* Clearly, such players do not perform any change in their strategy and remain on their green strategy.

By exploiting Lemma 5, since $q \geq 1$, for an arbitrarily small $\overline{\epsilon}$, the total number of players is

$$n = \sum_{j=1}^{k+1} |L(T_{h,q}, j)| \leq 1 + \sum_{j=2}^{k+1} \left(q^{2^{j-1}-1} \left(\frac{1}{2} + \frac{1}{q} \right)^{j-2} \right)$$
$$\leq 1 + \frac{1}{q} \sum_{j=1}^{k} q^{2^j} \leq (1 + \overline{\epsilon}) q^{2^k - 1}.$$

Thus, $q \geq \sqrt[2^k-1]{\frac{n}{1+\overline{\epsilon}}}$.

Moreover, the optimum social cost is upper bounded by the social cost of the state in which every player selects her green strategy:

$$\begin{aligned}
\textsc{Opt} &\leq \sum_{j=1}^{k+1} |L(T_{h,q}, j)| f_{e \in L(T_{h,q}, j)}(1) \\
&\leq 1 + \sum_{j=2}^{k} \left(|L(T_{h,q}, j)| \left(\frac{1}{q^{2^{j-1}-1}} + \xi_j \right) \right) + |L(T_{h,q}, k+1)| \xi_{k+1} \\
&\leq 1 + \sum_{j=2}^{k} \left(q^{2^{j-1}-1} \left(\frac{1}{2} + \epsilon \right)^{j-2} \left(\frac{1}{q^{2^{j-1}-1}} + \xi_{k+1} \right) \right) + 1 \\
&\leq 5
\end{aligned}$$

for ξ_{k+1} small enough and q suitably large.

Since the social cost of the last state of k-th round is lower bounded by the latency suffered by the unique player of level 0 (using resource r_0), $\text{PoA}_k = \Omega(q) = \Omega\left(\sqrt[2^k-1]{n}\right)$. □

5 Open Problems

Besides closing the gaps between the lower and upper bounds on the price of anarchy after k rounds, that however do not affect the tightness between the necessary and sufficient number of rounds for achieving a constant factor performance, many questions are left open.

First of all, it would be nice to extend our results to the case in which players are weighted, that is they increase the congestion of the selected resources of factors different from one.

Moreover, besides the total latency cost, it would be interesting to consider the case in which the social value is given by the maximum players' cost is another relevant issue.

Finally, another worth pursuing research direction concerns the adoption of other realistic non-linear latency functions.

References

1. Ackermann, H., Röglin, H., Vöcking, B.: On the impact of combinatorial structure on congestion games. In: FOCS, pp. 613–622. IEEE Computer Society, Los Alamitos (2006)
2. Awerbuch, B., Azar, Y., Epstein, A.: Large the price of routing unsplittable flow. In: STOC, pp. 57–66. ACM, New York (2005)
3. Awerbuch, B., Azar, Y., Epstein, A., Mirrokni, V.S., Shopalik, A.: Fast convergence to nearly optimal solutions in potential games. In: EC 2008 (to appear, 2008)
4. Caragiannis, I., Flammini, M., Kaklamanis, C., Kanellopoulos, P., Moscardelli, L.: Tight bounds for selfish and greedy load balancing. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 311–322. Springer, Heidelberg (2006)
5. Chien, S., Sinclair, A.: Convergence to approximate nash equilibria in congestion games. In: SODA, pp. 169–178. SIAM, Philadelphia (2007)
6. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: STOC, pp. 67–73. ACM, New York (2005)
7. Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and approximation in potential games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 349–360. Springer, Heidelberg (2006)
8. Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and approximation in potential games. Personal Communication (2007)
9. Yannakakis, M., Johnson, D.S., Papadimitriou, C.H.: How easy is local search? Journal of Computer and System Sciences 37, 79–100 (1988)
10. Fabrikant, A., Papadimitriou, C.H., Talwar, K.: The complexity of pure equilibria. In: Proceedings of the 36th ACM Symposium on Theory of Computing (STOC), pp. 604–612. ACM, New York (2004)
11. Fanelli, A., Flammini, M., Melideo, G., Moscardelli, L.: Multicast transmissions in non-cooperative networks with a limited number of selfish moves. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 363–374. Springer, Heidelberg (2006)
12. Fanelli, A., Flammini, M., Moscardelli, L.: On the convergence of multicast games in directed networks. In: SPAA, pp. 330–338. ACM, New York (2007)
13. Goemans, M.X., Mirrokni, V.S., Vetta, A.: Sink equilibria and convergence. In: FOCS, pp. 142–154. IEEE Computer Society, Los Alamitos (2005)
14. Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
15. Milchtaich, I.: Congestion games with player-specific payoff functions. Games and Economic Behavior 13, 111–124 (1996)
16. Mirrokni, V.S., Vetta, A.: Convergence issues in competitive games. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) RANDOM 2004 and APPROX 2004. LNCS, vol. 3122, pp. 183–194. Springer, Heidelberg (2004)
17. Nash, J.F.: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences 36, 48–49 (1950)
18. Rosenthal, R.W.: A class of games possessing pure-strategy nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
19. Skopalik, A., Vöcking, B.: Inapproximability of convergence in congestion games (manuscript, 2007)

Uniform Budgets and the Envy-Free Pricing Problem

Patrick Briest*

Dept. of Computer Science, University of Liverpool, UK
patrick.briest@liverpool.ac.uk

Abstract. We consider the unit-demand min-buying pricing problem, in which we want to compute revenue maximizing prices for a set of products $\mathcal{P}$ assuming that each consumer from a set of consumer samples $\mathcal{C}$ will purchase her cheapest affordable product once prices are fixed. We focus on the special uniform-budget case, in which every consumer has only a single non-zero budget for some set of products. This constitutes a special case also of the unit-demand envy-free pricing problem.

We show that, assuming specific hardness of the balanced bipartite independent set problem in constant degree graphs or hardness of refuting random 3CNF formulas, the unit-demand min-buying pricing problem with uniform budgets cannot be approximated in polynomial time within $\mathcal{O}(\log^{\varepsilon} |\mathcal{C}|)$ for some $\varepsilon > 0$. This is the first result giving evidence that unit-demand envy-free pricing, as well, might be hard to approximate essentially better than within the known logarithmic ratio.

We then introduce a slightly more general problem definition in which consumers are given as an explicit probability distribution and show that in this case the envy-free pricing problem can be shown to be inapproximable within $\mathcal{O}(|\mathcal{P}|^{\varepsilon})$ assuming NP $\not\subseteq \bigcap_{\delta>0}$ BPTIME$(2^{\mathcal{O}(n^{\delta})})$. Finally, we briefly argue that all the results apply to the important setting of pricing with single-minded consumers as well.

1 Introduction

Inspired by the possibility of gathering large amounts of data about the preferences and budgets of a company's potential customers by web sites designed for this purpose, Rusmevichientong [17] and Glynn et al. [12] introduced a class of so called *multi-product pricing* problems that aim at computing optimal pricing schemes for a company's product range. In the original version of the problem each consumer is represented by a budget and a set of products she is interested in. Given fixed prices for the products, she decides to buy one of the products she is interested in with a price not exceeding her budget. The decision is made corresponding to either the *min-buying*, *max-buying*, or *rank-buying* model, where the consumer buys the product with lowest price not exceeding the budget, highest price not exceeding the budget, or highest rank according to some consumer

* Supported by DFG grant Kr 2332/1-2 within Emmy Noether program.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 808–819, 2008.

specific ranking, respectively. All these problems are usually referred to as *unit-demand pricing*, as consumers will decide to buy exactly one product if they can afford to do so.

Aggarwal et al. [1] extend the problem definition and allow consumers with different budgets for the different products they are interested in. Assuming that a *price ladder constraint*, i.e., a predefined order on the prices of all products, is given, they derive a polynomial time approximation scheme (PTAS) for the max-buying and rank-buying (under another reasonable assumption) models. They also show how to obtain logarithmic approximation ratios for all three models if no price ladder is given. Briest and Krysta [6] show that both of these algorithms are essentially best possible.

Guruswami et al. [13] consider a different selection rule, which has already been proposed in [1]. In the *max-gain* model, a consumer buys the product maximizing her personal utility, i.e., the difference between the product's price and her respective budget. In the case of limited product supply, the definition in [13] additionally requires that each consumer must obtain the product she desires most whenever she can afford any product at all. Thus, the resulting pricing scheme must be *envy-free* and we obtain the *unit-demand envy-free pricing* problem, which has received a lot of attention. Guruswami et al. present an algorithm with logarithmic approximation guarantee for this problem and prove APX-hardness. Chawla et al. [7] consider the situation in which instead of having consumer samples, consumers are drawn from an explicit probability distribution. They show that for the case of product distributions (i.e., consumers' budgets are drawn independently for different products), results from optimal auction theory [16] yield constant approximation guarantees.

Another problem introduced in [13] is so called *single-minded pricing*, which is inspired by single-minded combinatorial auction design. In this scenario each consumer has a single budget value and buys the whole set of products she is interested in if the sum of prices does not exceed her budget. Among other results, Guruswami et al. show that techniques similar to those of [1] yield a logarithmic approximation for this problem, which is proven to be close to best possible by Demaine et al. [9]. Balcan and Blum [3] and Briest and Krysta [5] present improved approximation results for several different restricted versions of the problem.

Finally, Balcan et al. [4] show how algorithmic pricing feeds back into incentive-compatible auction design and present competitive auctions based on pricing algorithms combined with an appropriate random sampling procedure.

1.1 Preliminaries

Most of this paper will be focused on the unit-demand min-buying (or envy-free) pricing problem (Udp-Min) with uniform budgets, which is defined formally below. Throughout the paper we will assume that all products are available in unlimited supply and have zero marginal cost.

Definition 1. *In* uniform-budget Udp-Min *we are given products* $\mathcal{P}$ *and consumer samples* $\mathcal{C}$ *consisting of budgets* $b_c \in \mathbb{R}_0^+$ *and product sets* $S_c \subseteq \mathcal{P}$ *for all* $c \in \mathcal{C}$*. We want to find prices* $p : \mathcal{P} \to \mathbb{R}_0^+$ *that maximize*

$$rev_{min}(p) = \sum_{c \in \mathcal{A}(p)} \min\{p(e) \mid e \in S_c \wedge p(e) \leq b_c\},$$

where $\mathcal{A}(p) = \{c \in \mathcal{C} \mid \exists e \in S_c : p(e) \leq b_c\}$ *denotes the set of consumers that can afford to buy any product given prices* p*.*

Unit-demand pricing models the situation that products are *strict substitutes* and each consumer is interested in purchasing exactly one product out of a set of alternatives. The other extreme is reached if products constitute *strict complements* and every consumer seeks to purchase some specific set of products rather than a single alternative. In the single-minded pricing problem (Smp) we assume that each consumer is interested in exactly one such product set, which she purchases if the sum of prices does not exceed her budget.

Definition 2. *Given products* $\mathcal{P}$ *and consumer samples* $\mathcal{C}$ *consisting of budgets* $b_c \in \mathbb{R}_0^+$ *and product sets* $S_c \subseteq \mathcal{P}$*,* Smp *asks for prices* $p : \mathcal{P} \to \mathbb{R}_0^+$ *maximizing*

$$rev_{smp}(p) = \sum_{c \in \mathcal{A}(p)} \sum_{e \in S_c} p(e),$$

where $\mathcal{A}(p) = \{c \in \mathcal{C} \mid \sum_{e \in S_c} p(e) \leq b_c\}$*.*

A natural extension of both problems is obtained if we assume that our knowledge of consumer preferences does not stem from some sampling procedure, but that we know the explicit probability distribution over the space $\mathcal{C}^*$ of all possible consumers, which is a widely spread assumption in economics (see, e.g., [7] or [16]). Thus, in the *economist's version* of these problems we are given a probability distribution $\mathcal{D}$ over consumer space $\mathcal{C}^*$. In this situation our aim is to find prices p maximizing the expected revenue from a sale to a single consumer drawn according to distribution $\mathcal{D}$. In order to avoid additional complications (which are of no interest to this paper) we restrict ourselves to finite support distributions, i.e., consumer sets $\mathcal{C}$ with a discrete distribution $\mathcal{D}$ defined on $\mathcal{C}$.

1.2 New Results

We first focus on the sampling-based version of uniform-budget Udp-Min and prove that assuming specific hardness of refuting random 3SAT-instances or approximating the balanced bipartite independent set problem (Bbis) in constant degree graphs, this problem does not allow approximation guarantees essentially beyond the known logarithmic ratios. The connection between Bbis and Udp-Min is made via so-called *maximum expanding sequences* (Mes), which are a combinatorial formulation of the interaction between different price levels in Udp-Min and potentially also of independent interest. In order to show

hardness of sampling-based UDP-MIN we need hardness of very sparse MES instances, which we obtain from constant degree BBIS by the careful application of derandomized graph products [2] to scale hardness to the desired level. Unfortunately, no explicit hardness of approximation results are known for constant degree BBIS, although the problem has been receiving considerable attention. We show that hardness of constant degree BBIS can be derived from a hypothesis about the average case complexity of refuting random 3SAT-instances, which is almost identical to the one originally put forward by Feige [10] in a similar context. Since UDP-MIN with uniform budgets is a special case of the envy-free pricing problem from [13], for which previously only APX-hardness was known, our results yield the first (strong) evidence that this problem might be hard to approximate within $\mathcal{O}(\log^{\varepsilon} |\mathcal{C}|)$ for some $\varepsilon > 0$. Turning to the economist's version of envy-free pricing we obtain strong hardness of approximation under standard assumptions, in which case the reduction from BBIS to MES yields inapproximability within $\mathcal{O}(|\mathcal{P}|^{\varepsilon})$ for some $\varepsilon > 0$. Similar bounds in terms of the number of products can be shown for the sampling-based version of the problem, if we strengthen the underlying hypothesis a little further.

Finally, we point out that with a few minor modifications the same reductions yield similar hardness results for SMP as well. Even though SMP is known to be hard to approximate within semi-logarithmic ratios [9] in the number of consumer samples, this has some new and interesting implications. First, we obtain the first near-tight hardness results for approximation guarantees expressed in the number of products rather than the number of consumer samples. Second, we obtain lower bounds for the economist's problem version, which can provably not be derived from previous reductions. This also yields evidence that maximum expanding sequences are a combinatorial problem that is implicitly present in quite different combinatorial pricing problems.

Independently of this paper, Chuzhoy et al. [8] have quite recently considered uniform-budget UDP-MIN in a network setting with unit-sized flows, obtaining semi-logarithmic lower bounds on the approximability in terms of the network size. While their result holds under standard complexity theoretic assumptions even for unit-sized flows, our results imply stronger bounds for the non-unit flow case, which corresponds to the economist's version of UDP-MIN.

The rest of this paper is organized as follows. We proceed by giving an exposition of our results on UDP-MIN in Section 2. Section 3 briefly describes the application of our results to SMP. Section 4 concludes.

2 Unit-Demand Pricing

As the main result of this section we describe a reduction from the *Balanced Bipartite Independent Set Problem* (BBIS) in constant degree bipartite graphs to uniform-budget UDP-MIN. This will prove that, assuming there are no randomized polynomial time algorithms approximating constant degree BBIS within arbitrarily small constant factors, there are no polynomial time algorithms approximating UDP-MIN within $\mathcal{O}(\log^{\varepsilon} |\mathcal{C}|)$ for some $\varepsilon > 0$. At the very end of the

section we state similar (yet stronger) results for the economist's version of the problem, which hold under standard complexity theoretic assumptions.

Up to now, no explicit hardness results have been proven for Bbis in constant degree graphs, although the problem has been receiving a lot of attention. The first result for general Bbis using a quite moderate complexity theoretic assumption was obtained by Khot [14]. Previous results by Feige [10] and Feige and Kogan [11] are deriving hardness of Bbis under more specific assumptions. In [10] Feige shows an interesting connection between the average case complexity of refuting 3CNF-formulas and the worst case approximation complexity of several notorious optimization problems including Bbis. We are going to formulate a slightly stronger version of the hypothesis in [10] and show that this is enough for our purposes.

Remember that a 3CNF-formula is a conjunction of clauses, each of which is the disjunction of 3 literals over variables $x_1, \ldots, x_n$, where a literal is a variable or its negation. Before stating the hypothesis we need to describe the random sampling procedure used to obtain 3CNF formulas in [10]. Given n variables we create formulas consisting of $m = \Delta n$ clauses for some large constant $\Delta \in \mathbb{N}$ by independently picking each literal of every clause uniformly at random. When Δ is large enough, every truth assignment satisfies roughly $(7/8)m$ clauses of such a random 3CNF formula. Thus, a *typical* random 3CNF formula does not have significantly more than $(7/8)m$ simultaneously satisfiable clauses. On the other hand, for a sufficiently small $\varepsilon > 0$, formulas with $(1-\varepsilon)m$ simultaneously satisfiable clauses can be considered *exceptional*. Hypothesis 1 states that it is hard to detect exceptional formulas on average.

Hypothesis 1. *For every fixed $\varepsilon > 0$ and sufficiently large constant $\Delta \in \mathbb{N}$, there is no (randomized) algorithm that runs in time $\mathcal{O}(t(n))$ and, given a random 3CNF formula with n variables and $m = \Delta n$ clauses, outputs* typical *with probability at least $1/2$ (randomization over input), but outputs* exceptional *on every formula with $(1-\varepsilon)m$ simultaneously satisfiable clauses with probability at least $1 - 1/2^{poly(n)}$ (randomization over algorithm's coin flips).*

Choosing $t(n) = \text{poly}(n)$ the only difference between Hypothesis 1 and the hypothesis in [10] is that we also exclude randomized algorithms that have exponentially small error probability when it comes to detecting exceptional formulas. We need this stronger version as a result of our reduction from Bbis to uniform-budget Udp-Min, which is partially based on a random construction that introduces an exponentially small one-sided error probability for detecting large independent sets. We are mostly interested here in the case of $t(n) = \text{poly}(n)$. However, going to other subexponential time bounds will allow us to obtain lower bounds for differently parametrized approximation goals. In analogy to [10] we define a notion of hardness based on Hypothesis 1. We use slightly different notation compared to [10] to reflect the difference in the underlying hypotheses.

Definition 3. *A problem is said to be* R3SAT$^\star(t(n))$*-hard, if the existence of a (randomized) polynomial time algorithm (with exponentially small failure probability) for it refutes Hypothesis 1.*

Most importantly, R3SAT$^\star(t(n))$-hard problems do not allow polynomial time algorithms if we believe that Hypothesis 1 is true for the given choice of $t(n)$. As a byproduct of the fact that Hypothesis 1 also excludes certain randomized algorithms, R3SAT$^\star(t(n))$-hardness rules out the existence of this type of algorithm, too. We continue by giving a formal definition of Bbis, the base problem of our reduction.

Definition 4. *In the* Balanced Bipartite Independent Set Problem *(*Bbis*) we are given a bipartite graph $G = (V, W, E)$. We want to find maximum cardinality subsets of vertices $V' \subset V$, $W' \subset W$ with $|V'| = |W'|$, such that $\{v, w\} \notin E$ for all $v \in V'$, $w \in W'$.*

A slightly refined version of the analysis presented in [10] can be used to obtain R3SAT$^\star(\text{poly}(n))$-hardness of Bbis in constant degree graphs. We point out that this part of the reduction can be replaced by the following weaker hypothesis, which states that the gap variant of Bbis in constant degree graphs does not have randomized polynomial time algorithms with one-sided error (i.e., the decision variant does not belong to class RP). More formally, let $\mathcal{G}^-(a, d)$, $\mathcal{G}^+(b, d)$ be two families of bipartite graphs on $2n$ vertices with constant degree $d \in \mathbb{N}$ and maximum balanced independent set of size at most an or at least bn, respectively. Given $0 < a < b < 1$ and $d \in \mathbb{N}$ the problem Bbis(a, b, d) requires deciding whether $G \in \mathcal{G}^-(a, d)$ or $G \in \mathcal{G}^+(b, d)$ for a given graph $G \in \mathcal{G}^-(a, d) \cup \mathcal{G}^+(b, d)$. For our purposes Hypothesis 2 is fully sufficient.

Hypothesis 2. *There exist constants $0 < a < b < 1$ and $d \in \mathbb{N}$, such that* Bbis$(a, b, d) \notin RP$.

Without expressing too much of an opinion about the validity of Hypothesis 2, it should be noted that it is in accordance with our current knowledge and backed by the fact that strong super-constant approximability thresholds have been proven for general Bbis [14]. Having hardness of constant degree Bbis we apply the method of derandomized graph products [2] to obtain hardness of approximation within $\mathcal{O}(f(n)^\varepsilon)$ for Bbis in graphs with maximum degree $\mathcal{O}(f(n))$, where the appropriate choice for $f(n)$ will become apparent later on. The main part of our result consists of the reduction to Udp-Min. As an intermediate step in the reduction we modify the Bbis instance by adding a number of random edges and interpret vertices on one side of the bipartition as sets. The connection to Udp-Min is made by considering sequences of these sets that have a certain expansion property. This is formalized in the following definition.

Definition 5. *In the* Maximum Expanding Sequence Problem *(*Mes*) we are given an ordered collection $S_1, \ldots, S_m$ of sets. An expanding sequence $\phi = (\phi(1) < \cdots < \phi(\ell))$ of length $|\phi| = \ell$ is a selection of sets $S_{\phi(1)}, \ldots, S_{\phi(\ell)}$, such that*

$$S_{\phi(j)} \nsubseteq \bigcup_{i=1}^{j-1} S_{\phi(i)}$$

for $2 \leq j \leq \ell$. Mes *asks for finding such a sequence of maximum length.*

We are not aware that MES has been considered explicitly before. We briefly point out that a reduction similar to the one given in the proof of Lemma 1 below yields hardness of approximation under a standard assumption, which is formally stated in Theorem 4. In order to reduce MES to UDP-MIN with consumer samples we have to focus our attention on severely restricted problem instances. BBIS instances with bounded maximum degree yield MES instances that exhibit a nicely sparse structure. Definition 6 formalizes our notion of *sparse*.

Definition 6. *We say that an* MES *instance* $S_1, \ldots, S_m$ *is* κ-separable *if it can be partitioned into* κ *subsequences* $\mathcal{C}_1, \ldots, \mathcal{C}_\kappa$, $\mathcal{C}_j = \{S_{k(j)}, S_{k(j)+1}, \ldots, S_{\ell(j)}\}$, *where* $k(1) = 1$, $\ell(\kappa) = m$, $k(j+1) = \ell(j) + 1$ *for* $1 \leq j \leq \kappa - 1$ *and each* $\mathcal{C}_j$ *contains only non-intersecting sets.*

Our starting point to prove hardness of approximation for sufficiently sparse MES instances is Theorem 1, which can be derived by applying the method of derandomized graph products [2] to constant degree BBIS, and states super-constant approximability thresholds for BBIS parametrized in the graph's super-constant maximum degree $f(n)$.

Theorem 1. *Let* $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ *be non-decreasing with* $f(n) \leq n$ *and* $f(n^c) \leq f(n)^c$ *for all* $c \geq 1$, $n \in \mathbb{N}$. *Let* $\mathcal{G}^-(a(n), f(n))$ *and* $\mathcal{G}^+(b(n), f(n))$ *be the families of balanced bipartite graphs on* $2n$ *vertices, with maximum degree bounded by* $f(n)$ *and maximum* BBIS *of size at most* $a(n)n$ *or at least* $b(n)n$, *respectively. There exist* $0 < a(n) < b(n) < 1$ *with* $b(n)/a(n) = \Omega(f(n)^\varepsilon)$ *for some* $\varepsilon > 0$, *such that given* $G \in \mathcal{G}^-(a(n), f(n)) \cup \mathcal{G}^+(b(n), f(n))$ *it is* R3SAT*(poly(n))*-hard to decide whether* $G \in \mathcal{G}^-(a(n), f(n))$ *or* $G \in \mathcal{G}^+(b(n), f(n))$.

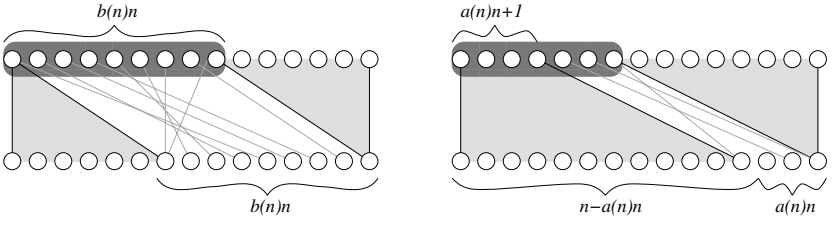


Fig. 1. Reducing BBIS to MES. (a) Adding random edges with probability $1/(b(n)n)$ each implants an expanding sequence of expected size $\Omega(b(n)n)$ into an independent set of size $b(n)n$. (b) An independent set of size $a(n)n$ allows for expanding sequences of at most twice that size.

Lemma 1. *There exists* $\varepsilon > 0$, *such that* MES *with* $f(m)$*-separable instances (*f *as in Theorem 1) is* R3SAT*(poly(n))*-hard to approximate within* $\mathcal{O}(f(m)^\varepsilon)$.

Proof. Let some $G \in \mathcal{G}^-(a(n), f(n)) \cup \mathcal{G}^+(b(n), f(n))$, $G = (V, W, E)$, $|V| = |W| = n$, with $a(n)$, $b(n)$ and $f(n)$ as in Theorem 1 be given. We will reduce the problem of deciding whether $G \in \mathcal{G}^-(a(n), f(n))$ or $G \in \mathcal{G}^+(b(n), f(n))$ to

solving a separable instance of MES, essentially by implanting large expanding sequences into large balanced independent sets. As neither the independent set nor its size are known at reduction time, we do this by adding a number of random edges to the graph, which create a long expanding sequence in expectation if a large balanced independent set exists (see Fig. 1).

More precisely, every possible edge is independently added to G with probability $(b(n)n)^{-1}$ if it is not already present in the original graph. We then remove vertices whose degree has become too high. In expectation the random procedure above tries to add $b(n)^{-1}$ edges to every vertex $v \in V \cup W$. We remove a vertex v and all its incident edges if more than $c \cdot b(n)^{-1}$ edges are added to it, where c is some sufficiently large constant to be determined later. Let A_v be the random variable counting the number of edges added to v. Applying the Chernoff bound [15] we obtain $\Pr(v \text{ is removed}) = \Pr(A_v \geq c \cdot b(n)^{-1}) \leq \mathrm{e}^{-c/b(n)}$ for any constant $c \geq 3\mathrm{e} - 1$. We denote the modified graph by $G' = (V', W', E')$. For every vertex $v_i \in V'$ we define a corresponding set S_i by $S_i = \{w_j \in W' \mid \{v_i, w_j\} \in E'\}$, i.e., vertices V' will correspond to sets over the universe W' in our MES instance.

In order to obtain a feasible MES instance we need to define an order on sets S_i, which we do next. Observe that vertices in G' have degree at most $f'(n) \leq f(n) + c \cdot b(n)^{-1} = \mathcal{O}(f(n))$, where we use the fact that bipartite graphs with bounded degree $f(n)$ have a balanced independent set of size at least $n/(f(n)+1)$ and, thus, it must be the case that $b(n)^{-1} = \mathcal{O}(f(n))$ in order for the problem to be non-trivial. Furthermore, if the maximum degree of G' is $f'(n)$, then the sets S_i can be partitioned into $f'(n)^2$ classes, such that sets in each class do not intersect, since every set contains at most $f'(n)$ elements, each of which is contained in at most $f'(n) - 1$ further sets. Reordering the sets appropriately we obtain an $\mathcal{O}(f(n)^2)$-separable MES instance.

Soundness: Let $G \in \mathcal{G}^+(b(n), f(n))$. Assume for the moment that no vertices are removed from G and let $\mathcal{S}^* = \{S_{\phi(1)}, \ldots, S_{\phi(\ell)}\}$, $\ell = \lceil b(n)n \rceil$, be the sets in the MES instance corresponding to vertices from V that belong to a maximum balanced bipartite independent set. Analogously, let $W^* \subset W$ denote the vertices from W belonging to the balanced bipartite independent set. For $1 \leq j \leq \ell/2$ consider set $S_{\phi(j)}$. We say that $S_{\phi(j)}$ is *successful* if we can use it to construct a large expanding sequence or, more formally, if the following conditions are satisfied:

- A_j: $|S_{\phi(j)} \cap W^*| = 1$, i.e., $S_{\phi(j)}$ contains exactly one element from W^*.
- B_j: $S_{\phi(j)} \cap S_{\phi(i)} \cap W^* = \emptyset$ for all $1 \leq i \leq \ell/2$, $i \neq j$, i.e., the intersection of $S_{\phi(j)}$ with any other set from $\mathcal{S}^*$ lies outside W^*.
- C_j: The vertex corresponding to set $S_{\phi(j)}$ is not removed due to the degree constraint.
- D_j: None of the vertices in $S_{\phi(j)} \cap W^*$ are removed due to the degree constraint.

It is not difficult to check that successful sets belong to the MES-instance and form an expanding sequence, since their corresponding vertices are not removed from the graph and each set covers a unique element in W^*, which yields the necessary expansion property. Let us now determine the probability that set $S_{\phi(j)}$ is successful. We can write that

$$\begin{aligned}\Pr(S_{\phi(j)} \text{ is successful}) &= 1 - \Pr(\overline{A}_j \vee \overline{B}_j \vee \overline{C}_j \vee \overline{D}_j)\\ &= 1 - \Pr(\overline{A}_j) - \Pr(\overline{B}_j|A_j)\cdot\Pr(A_j)\\ &\quad - \Pr(\overline{C}_j|A_j \wedge B_j)\cdot\Pr(A_j \wedge B_j)\\ &\quad - \Pr(\overline{D}_j|A_j \wedge B_j \wedge C_j)\cdot\Pr(A_j \wedge B_j \wedge C_j).\end{aligned}$$

We first consider event A_j and obtain

$$\begin{aligned}\Pr(A_j) &= \sum_{w\in W^*} \Pr\big(S_{\phi(j)} \cap W^* = \{w\}\big) = \sum_{w\in W^*} \frac{1}{b(n)n}\left(1 - \frac{1}{b(n)n}\right)^{\lceil b(n)n\rceil - 1}\\ &\approx b(n)n\frac{1}{\mathrm{e}b(n)n} \approx \frac{1}{\mathrm{e}},\end{aligned}$$

where the above holds with arbitrary precision for large values of n. Let us then consider $\Pr(\overline{B}_j|A_j)$. Sets $S_{\phi(i)}$ and $S_{\phi(j)}$ contain every element from W^* with equal probability $1/(b(n)n)$. Furthermore, $S_{\phi(i)} \cap W^*$ and $S_{\phi(j)} \cap W^*$ are independent by construction. Applying the union bound yields

$$\Pr(\overline{B}_j|A_j) \le \sum_{i=1}^{\ell/2}\sum_{w\in W^*} \Pr(w \in S_{\phi(i)})\Pr(w \in S_{\phi(j)}) \le \frac{\lceil b(n)n\rceil^2}{2}\frac{1}{(b(n)n)^2} \approx \frac{1}{2},$$

again with arbitrary precision for large n. We have already seen that the probability of any specific vertex being removed due to the degree constraint is bounded above by $\mathrm{e}^{-c/b(n)}$. We conclude that

$$\Pr(\overline{C}_j|A_j \wedge B_j)\cdot\Pr(A_j \wedge B_j) \le \Pr(\overline{C}_j) \le \mathrm{e}^{-c/b(n)},$$

and the same estimate holds for $\Pr(\overline{D}_j|A_j \wedge B_j \wedge C_j)\cdot\Pr(A_j \wedge B_j \wedge C_j)$. Thus,

$$\Pr(S_{\phi(j)} \text{ is successful}) \ge 1 - \left(1 - \frac{1}{\mathrm{e}}\right) - \frac{1}{2}\cdot\frac{1}{\mathrm{e}} - 2\mathrm{e}^{-c/b(n)} \approx \frac{1}{2\mathrm{e}}$$

for a sufficiently large constant c. Let Y denote the number of successful sets. By linearity of expectation and the above bounds it holds that $\mathrm{E}[Y] \ge (1/4\mathrm{e})b(n)n$. Using that the value of Y is bounded above by $b(n)n$ and applying a Markov type inequality then yields that $\Pr(Y \le 1/(8\mathrm{e})b(n)n) \le 1 - 1/(8\mathrm{e})$. This implies that with probability $\Omega(1)$ there exists an expanding sequence of length $\Omega(b(n)n)$.

Completeness: Let $G \in \mathcal{G}^-(a(n), f(n))$ and consider any expanding sequence ϕ in $S_1, \ldots, S_m$. Since the maximum balanced bipartite independent set in G is of size $a(n)n$, every selection of $a(n)n + 1$ vertices from V must be adjacent to all but $a(n)n$ vertices from W. Thus, the first $a(n)n + 1$ sets from ϕ leave at most $a(n)n$ elements uncovered. Since the expansion property requires that every further set in the sequence must contain a previously uncovered element, it follows that $|\phi| \le 2a(n)n + 1$.

We have shown a randomized reduction with constant one-sided error probability. By repeating the algorithm a polynomial number of times, we obtain error probabilities that are exponentially close to 0. This proves Lemma 1. □

To encode MES in terms of UDP-MIN we translate sets into collections of consumers with exponentially decreasing budgets. Reducing $\log m$-separable MES ensures the resulting instances are of polynomial size.

Theorem 2. *There exists a constant $\varepsilon > 0$, such that it is* R3SAT*(poly(n))*-hard to approximate uniform-budget* UDP-MIN *within* $\mathcal{O}(\log^\varepsilon |\mathcal{C}|)$. *Hardness of approximation holds even under the weaker assumption of Hypothesis 2.*

Proof. Let MES instance $S_1, \ldots, S_m$ be separable into $\mathcal{C}_1, \ldots, \mathcal{C}_\kappa$, $\kappa = \mathcal{O}(f(m))$. For each element e in the universe of the MES instance we have a corresponding product e. For every set S_i in class $\mathcal{C}_k$ we define a collection of 2^{k-1} identical consumers $C_i = \{c_i^1, c_i^2, \ldots, c_i^{2^{k-1}}\}$. Each of these consumers has budget $b_i = 2^{1-k}$ and is interested in products $e \in S_i$. Note that the total number of consumer samples in this construction is bounded above by $m2^{\mathcal{O}(f(m))}$.

Soundness: Let $\phi = (\phi(1) < \cdots < \phi(\ell))$ be an expanding sequence of length ℓ. For every $1 \leq i \leq \ell$ let $N_{\phi(i)}$ denote the elements that are newly covered by $S_{\phi(i)}$. Now, for $i = 1, \ldots, \ell$ determine $N_{\phi(i)}$ and set the prices of all products $e \in N_{\phi(i)}$ to b_i. For consumers $C_{\phi(i)}$ it then holds that $p(e) = b_i$ for all $e \in N_{\phi(i)}$, $p(e) > b_i$ for all $e \in S_{\phi(i)} \backslash N_{\phi(i)}$. As a result, all 2^{k-1} consumers belonging to a set $S_{\phi(i)}$ in the expanding sequence will buy at their budget value $b_i = 2^{1-k}$ and jointly contribute revenue 1. Thus, the overall revenue from consumers corresponding to the expanding sequence is at least ℓ.

Completeness: Assume that we are given a price assignment resulting in overall revenue r. First observe that w.l.o.g. all prices are from the set of distinct budget values, i.e., all prices are powers of 2. Then note that w.l.o.g. revenue at least $r/2$ is due to consumers buying at their budget values, since otherwise we could increase overall revenue by doubling all prices. Finally, it is not difficult to see that consumers buying at their budget values form an expanding sequence, as each such consumer must be purchasing a product that none of the consumers with higher budgets is interested in. It follows that we obtain an expanding sequence ϕ of length at least $r/2$. Choosing $f(m) = \log m$ yields the theorem. □

Our reduction is flexible enough to yield inapproximability results also in the maximum number ℓ of non-zero budgets per consumer and, allowing UDP-MIN instances of arbitrary subexponential size, we can stretch the construction to the limit and obtain lower bounds on the approximability in terms of the number of products $|\mathcal{P}|$ as well.

Theorem 3. *There exist constants $\ell_0 \in \mathbb{N}$ and $\varepsilon > 0$, such that for every $\ell \geq \ell_0$ it is* R3SAT*(poly(n))*-hard to approximate uniform-budget* UDP-MIN *with at most ℓ non-zero budgets per consumer within ℓ^ε. Furthermore, for every $\delta > 0$ there exists $\varepsilon > 0$, such that it is* R3SAT$^\star(2^{\mathcal{O}(n^\delta)})$*-hard to approximate uniform-budget* UDP-MIN *within* $\mathcal{O}(|\mathcal{P}|^\varepsilon)$.

Next, let us consider the economist's versions of uniform-budget UDP-MIN and the unit-demand envy-free pricing problem. As mentioned before, a reduction

similar to the one given in the proof of Lemma 1 in combination with the known hardness results for general Bbis from [14] yields a strong hardness result for general Mes. Applying the reduction from the proof of Theorem 2 we obtain inapproximability results for uniform-budget Udp-Min (economist's version) under standard complexity theoretic assumptions. These immediately extend to the more general unit-demand envy-free pricing problem.

Theorem 4. Mes *is inapproximable within* $\mathcal{O}(m^\varepsilon)$ *for some* $\varepsilon > 0$*, assuming that* NP $\notin \bigcap_{\delta>0}$ BPTIME$(2^{\mathcal{O}(n^\delta)})$.

Theorem 5. Udp-Min *(economist's version) with uniform budgets is hard to approximate within* $\mathcal{O}(|\mathcal{P}|^\varepsilon)$ *for some* $\varepsilon > 0$*, if* NP $\notin \bigcap_{\delta>0}$ BPTIME$(2^{\mathcal{O}(n^\delta)})$.

Finally, let us mention that it is not difficult to achieve approximation guarantee $\mathcal{O}(|\mathcal{P}|)$ for the economist's envy-free pricing problem, e.g., by using the known single-price algorithm [1].

3 Single-Minded Pricing

We can adapt the reduction in Theorem 2 to work for single-minded consumers as well. In fact, all we need to do is to choose price levels as powers of $2|\mathcal{P}|$ rather than 2 and define consumers in the opposite direction, i.e., start on the lowest price level and work our way up. Again only consumers corresponding to an expanding sequence allow extraction of full revenue. Demaine et al. [9] prove an approximation threshold of $\Omega(\log^\varepsilon |\mathcal{C}|)$ for Smp under standard complexity theoretic assumptions. Our reduction yields these and asymptotically stronger bounds in the number of products based on the notion of R3SAT*-hardness.

Theorem 6. *For every* $\delta > 0$ *there exists* $\varepsilon > 0$*, such that it is R3SAT*$^\star(2^{\mathcal{O}(n^\delta)})$*-hard to approximate* Smp *within* $\mathcal{O}(|\mathcal{P}|^\varepsilon)$.

Once more, turning to the economist's version of the problem, we obtain strong inapproximability results under standard assumptions. We point out that it is not possible to achieve these bounds by previous reductions.

Theorem 7. Smp *(economist's version) is hard to approximate within* $\mathcal{O}(|\mathcal{P}|^\varepsilon)$ *for some* $\varepsilon > 0$*, if* NP $\notin \bigcap_{\delta>0}$ BPTIME$(2^{\mathcal{O}(n^\delta)})$.

4 Conclusions

In this paper we have made progress towards understanding the difficulty of different combinatorial pricing problems. First, we have shown that assuming specific hardness of constant degree Bbis or hardness on average of refuting random 3CNF-formulas, the unit-demand min-buying pricing problem with uniform budgets, which constitutes a special case also of unit-demand envy-free pricing, does not allow sub-logarithmic approximation guarantees. Secondly, we have shown that our techniques apply to the case of single-minded pricing as well, which indicates that expanding sequences are a common source of hardness for quite different combinatorial pricing problems.

Acknowledgments

The author thanks Piotr Krysta for insightful discussions and several anonymous referees for their valuable comments.

References

1. Aggarwal, G., Feder, T., Motwani, R., Zhu, A.: Algorithms for Multi-Product Pricing. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142. Springer, Heidelberg (2004)
2. Alon, N., Feige, U., Wigderson, A., Zuckerman, D.: Derandomized Graph Products. Computational Complexity 5, 60–75 (1995)
3. Balcan, N., Blum, A.: Approximation Algorithms and Online Mechanisms for Item Pricing. In: Proc. of 7th ACM Conference on Electronic Commerce (EC) (2006)
4. Balcan, N., Blum, A., Hartline, J., Mansour, Y.: Mechanism Design via Machine Learning. In: Proc. of 46th IEEE Symposium on Foundations of Computer Science (FOCS) (2005)
5. Briest, P., Krysta, P.: Single-Minded Unlimited-Supply Pricing on Sparse Instances. In: Proc. of 17th ACM-SIAM Symposium on Discrete Algorithms (SODA) (2006)
6. Briest, P., Krysta, P.: Buying Cheap is Expensive: Hardness of Non-Parametric Multi-Product Pricing. In: Proc. of 18th ACM-SIAM Symposium on Discrete Algorithms (SODA) (2007)
7. Chawla, S., Hartline, J., Kleinberg, R.: Algorithmic Pricing via Virtual Valuations. In: Proc. of 8th ACM Conference on Electronic Commerce (EC) (2007)
8. Chuzhoy, J., Kannan, S., Khanna, S.: Network Pricing for Multicommodity Flows (unpublished manuscript, 2007)
9. Demaine, E., Feige, U., Hajiaghayi, M., Salavatipour, M.: Combination Can Be Hard: Approximability of the Unique Coverage Problem. In: Proc. of 17th ACM-SIAM Symposium on Discrete Algorithms (SODA) (2006)
10. Feige, U.: Relations between Average Case Complexity and Approximation Complexity. In: Proc. of 34th ACM Symposium on Theory of Computing (STOC) (2002)
11. Feige, U., Kogan, S.: Hardness of Approximation of the Balanced Complete Bipartite Subgraph Problem. Technical Report MCS04-04, Dept. of Computer Science and Applied Math., The Weizmann Institute of Science (2004)
12. Glynn, P., Rusmevichientong, P., van Roy, B.: A Non-Parametric Approach to Multi-Product Pricing. Operations Research 54, 82–98 (2006)
13. Guruswami, V., Hartline, J., Karlin, A., Kempe, D., Kenyon, C., McSherry, F.: On Profit-Maximizing Envy-Free Pricing. In: Proc. of 16th ACM-SIAM Symposium on Discrete Algorithms (SODA) (2005)
14. Khot, S.: Ruling out PTAS for Graph Min-Bisection, Densest Subgraph and Bipartite Clique. In: Proc. of 45th IEEE Symposium on Foundations of Computer Science (FOCS) (2004)
15. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
16. Myerson, R.: Optimal Auction Design. Mathematics of Operations Research 6, 58–73 (1981)
17. Rusmevichientong, P.: A Non-Parametric Approach to Multi-Product Pricing: Theory and Application. PhD thesis, Stanford University (2003)

Bayesian Combinatorial Auctions

George Christodoulou[1], Annamária Kovács[2], and Michael Schapira[3]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
gchristo@mpi-inf.mpg.de
[2] Institute for Computer Science, J. W. Goethe University, 60325 Frankfurt/Main, Germany
panni@cs.uni-frankfurt.de
[3] The School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel
mikesch@cs.huji.ac.il*

Abstract. We study the following Bayesian setting: m items are sold to n selfish bidders in m independent second-price auctions. Each bidder has a *private* valuation function that expresses complex preferences over *all* subsets of items. Bidders only have *beliefs* about the valuation functions of the other bidders, in the form of probability distributions. The objective is to allocate the items to the bidders in a way that provides a good approximation to the optimal social welfare value. We show that if bidders have submodular valuation functions, then every Bayesian Nash equilibrium of the resulting game provides a 2-approximation to the optimal social welfare. Moreover, we show that in the full-information game a pure Nash always exists and can be found in time that is polynomial in both m and n.

1 Introduction

Combinatorial Auctions. In a *combinatorial auction* m items $M = \{1, \ldots, m\}$ are offered for sale to n bidders $N = \{1, \ldots, n\}$. Each bidder i has a *valuation function* (or valuation, in short) v_i that assigns a non-negative real number to *every* subset of the items. v_i expresses i's preferences over *bundles* of items. The value $v_i(S)$ can be thought of as specifying i's maximum willingness to pay for S. Two standard assumptions are made on each v_i: $v_i(\emptyset) = 0$ (normalization), and $v_i(S) \leq v_i(T)$ for every two bundles $S \subseteq T$ (monotonicity). The objective is to find a partition of the items among the bidders $S_1, \ldots, S_n$ (where $S_i \cap S_j = \emptyset$ for all $i \neq j$) such that the social welfare $\Sigma_i v_i(S_i)$ is maximized.

The interplay between selfishness and computational optimization in combinatorial auctions is well-studied. Each of these aspects alone can be handled in a satisfactory way: The celebrated VCG mechanisms [19,3,10] motivate agents to truthfully report their private information, and optimize the social-welfare. The caveat is that this may take exponential time [15,16] (in the natural parameters of the problem m and n). On the other hand, if we disregard strategic issues, it

* Supported by a grant from the Israel Science Foundation.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 820–832, 2008.

is possible to obtain good approximations to the optimal social-welfare in polynomial time, for restricted, yet very expressive special cases of combinatorial auctions (e.g., combinatorial auctions in which bidders have *submodular* valuations, known as combinatorial auctions with *submodular bidders* [13,4,5,6,7,20]). It is the combination of the two challenges that proves to be problematic.

Bayesian Combinatorial Auctions. In this paper we approach the problem of handling selfishness and computational hardness from an old-new perspective: Harsanyi [11] introduced *Bayesian games* as an elegant way of modeling selfishness in partial-information settings. In a Bayesian game, players do not know the private information of the other players, but only have *beliefs*, expressed by probability distributions over the different possible realizations of this private information. In combinatorial auctions this translates to probability distributions over the possible valuation functions of the other bidders. We are interested in maximizing the social-welfare in a way that is aligned with the interests of the different bidders. We ask the following question: Can we design an auction for which *any* Bayesian Nash equilibrium provides a good approximation to the optimal social-welfare? This question is, of course, an extension of the well known "*price of anarchy*" question [12,17] to Bayesian settings.

Inspired by eBay, we study the simple auction in which the m items are sold in m independent second-price auctions. This auction induces a game in which a bidder's strategy is the m-dimensional vector of bids he submits in the different single-item auctions, and his payoff is his value for the set of items he is allocated minus his payments. Unfortunately, in this general setting, some unnatural problems may arise: Consider the following simple example: $m = 1$, $n = 2$, and the bidders have complete information about each other. Let $v_1(1) = 1$ and $v_2(1) = 0$. Observe that the optimal social welfare is 1 (assign item 1 to bidder 1). Also observe that if bidder 1 bids 0 and bidder 2 bids 1 then this is a pure Nash equilibrium with a social-welfare value of 0. Therefore, the price of anarchy of this full-information game is infinity.

In the above scenario, the second bidder bid for (and eventually got) an object he was not interested in possessing. However, such a situation is unlikely to occur in real-life situations, especially if bidders are only partially informed and are therefore more inclined to avoid risks. To handle this, we adopt the well-known assumption in decision theory, that the players (bidders) are ex-post individually-rational. Informally, ex-post individual-rationality means that bidders play it safe, in the sense that a bidder will never make decisions that might (in some scenario) result in getting a negative payoff.

Our Results. Our main result is the following: We exhibit an auction (we refer to as a *Bayesian auction*) in which essentially all items are sold in a second-price auction. We prove that *any* mixed Nash equilibrium for this auction provides a good approximation to the optimal social welfare.

Theorem: If bidders are ex-post individually-rational, and have submodular valuation functions, then every (mixed) Bayesian Nash equilibrium of a Bayesian auction provides a 2-approximation to the optimal social welfare.

A bidder i is said to have a *submodular* valuation function if for all $S, T \subseteq M$ $v_i(S \cup T) + v_i(S \cap T) \leq v_i(S) + v_i(T)$. This definition of submodularity is known (see, e.g., [13]) to be equivalent to the following definition: A valuation function is submodular if for every two bundles $S \subseteq T$ that do not contain an item j it holds that

$$v_i(S \cup \{j\}) - v_i(S) \geq v_i(T \cup \{j\}) - v_i(T).$$

This last inequality has a natural interpretation as it implies that the bidders have decreasing marginal utilities (the marginal value of the item decreases as the number of items a bidder has increases). In fact, our theorem holds even if bidders have fractionally-subadditive valuation functions [6] (defined, and termed XOS, in [14]), a class of valuation functions that strictly contains all submodular ones [14,13].

We stress that this "*Bayesian price of anarchy*" result is independent of the bidders' beliefs. That is, the 2-approximation ratio is guaranteed for *any* common probability distribution ("common prior") over the valuation functions (we do require the common prior to be the product of independent probability distributions). This suggests a middle-ground between the classical economic and the standard computer science approaches: Works in economics normally assume that the "input" is drawn from some *specific* probability distribution, and prove results that apply to that specific one. In contrast, computer scientists prefer a worst-case analysis that holds for every possible input. We require the assumption that the input is drawn from some (known) probability distribution but expect to obtain a good approximation ratio regardless of what it is. We note that an approach similar to ours was applied to selfish routing problems in [9,8]

Open Question: Can a (mixed) Bayesian Nash equilibrium be computed in polynomial time?

Simple examples show that a *pure* Bayesian Nash is not guaranteed to exist in our Bayesian setting[1]. An interesting special case of the Bayesian game is the full-information game in which every bidder's valuation function is known to all other bidders. We show, that in such full-information games a *pure* Nash equilibrium *always* exists. This is true even if the bidders are *not* ex-post individually rational. In fact, it is easy to show that even the optimal allocation of items can be achieved in a pure Nash equilibrium. So, while the price of anarchy in this game is infinity (without ex-post individual-rationality), the *price of stability* (the social welfare value of the *best* Nash equilibrium [1]) is 1. However, optimizing the social-welfare in combinatorial auctions with submodular bidders in known to be NP-hard [13]. Therefore, we are left with the following natural question: Can we find a pure Nash equilibrium that provides a good approximation to the optimal social welfare in polynomial time?

[1] In fact, even a mixed Nash equilibrium is not guaranteed to exist unless one discretizes the strategy space (e.g., only allows bids that are multiples of some small $\epsilon > 0$).

We give the following answer for submodular bidders:

Theorem: If bidders have submodular valuation functions then a pure Nash equilibrium of the complete-information game that provides a 2-approximation to the optimal social welfare exists and can be computed in polynomial time.

We prove the theorem by showing that the approximation algorithm for maximizing social welfare in combinatorial auctions with submodular bidders, proposed by Lehmann et al. [13], can be used to compute the bids in a pure Nash equilibrium. We note that similar questions have been studied by Vetta in [18].[2]

For the wider class of fractionally-subadditive valuation functions, we provide a constructive way of finding a pure Nash that provides a 2-approximation via a simple and natural myopic procedure. This procedure is inspired by the greedy approximation-algorithm in [4]. Unfortunately, while this myopic procedure does compute a pure Nash equilibrium in polynomial time for some interesting (non-submodular) subcases, this is not true in general. We prove that the myopic procedure may take exponential time by exhibiting a non-trivial construction of an instance on which this can occur.

Open Question: Can a pure Nash equilibrium that obtains a 2-approximation to the optimal social-welfare be computed in polynomial time if bidders have fractionally-subadditive valuation functions?

The proofs omitted due to space limitations can be found in the full version of the paper.

2 Bayesian Price of Anarchy

In Subsection 2.1 we present the Bayesian setting we explore in this paper, which we term "Bayesian combinatorial auctions". In Subsection 2.2 we exhibit our main result, which is that the Bayesian price of anarchy of Bayesian combinatorial auctions is 2.

2.1 The Setting - Bayesian Combinatorial Auctions

The Auction. m items are sold to n bidders in m independent second-price auctions (with some tie-breaking rule). A bidder's strategy is a *bid-vector* $b_i \in R^m_{\geq 0}$ ($b_i(j)$ represents i's bid for item j).[3] A *(pure) strategy profile* of all players is an n-tuple $b = (b_1, \ldots, b_n)$. We will use the notation $b = (b_i, b_{-i})$, to denote

[2] Vetta considers a general setting in which decisions are made by non-cooperative agents, and the utility functions are submodular. He proves that in this setting the price of anarchy is at most 2. The framework discussed there is such that the players' pure strategies are *subsets* of a ground set (e.g., the items). This framework is not applicable to our auction, where the bids play a crucial role.

[3] So as to have a finite (discrete) model, we assume that all acceptable bids are multiples of an arbitrary ϵ. Furthermore, they cannot exceed some maximum value B_{max}.

a strategy profile in which bidder i bids b_i and other bidders bid as in $b_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$.

Given a strategy profile b, the items are allocated according to the second price rule, i.e., every object is sold to the highest bidder at a price equal to the second highest bid. For technical reasons, to be explained later, in each such second-price auction with negligible probability the item is randomly allocated to one of the bidders (who submitted a bid higher than zero). In this case, that bidder is charged his bid for the item.

For some fixed b we denote by $X_i(b)$ the set of items obtained by player i in the auction[4]. For a set $S \subseteq M$, let the sum of the (highest) bids be denoted by

$$Bids(S, b) = \sum_{j \in S} \max_k b_k(j),$$

$$Bids_{-i}(S, b_{-i}) = \sum_{j \in S} \max_{k \neq i} b_k(j),$$

and

$$Bids_i(S, b_i) = \sum_{j \in S} b_i(j).$$

The *utility* (payoff) of player i is then given by

$$u_i(b) = v_i(X_i(b)) - Bids_{-i}(X_i(b), b_{-i}).$$

We make two assumptions about the bidders: ex-post individual-rationality, and that the v_is are *fractionally-subadditive*. A valuation is fractionally-subadditive if it is the pointwise maximum of a set of additive valuations: A valuation a_i is additive if for every $S \subseteq M$ $a_i(S) = \Sigma_{j \in S} a_i(\{j\})$. A valuation v_i is fractionally-subadditive if there are additive valuations $A = \{a_1, \ldots, a_l\}$ such that for every $S \subseteq M$ $v_i(S) = \max_{a \in A} a(S)$. (We will call $a_k \in A$ a *maximizing additive valuation* for the set S if $v_i(S) = a_k(S)$.)

The class of fractionally-subadditive valuations is known to be strictly contained in the class of subadditive valuations and to strictly contain the class of submodular valuations [14,13].

Bayesian Nash Equilibria. For all i, let V_i denote the finite set of possible valuations of player i. The set of possible valuation profiles of the players is then $V = V_1 \times \ldots \times V_n$. There is a *known* probability distribution D over the valuations V (a *common prior*). D can be regarded as some market statistics that is known to all bidders (and to the auctioneer), and specifies their beliefs. We assume that $D = D_1 \times \ldots \times D_n$ is the cartesian product of independent probability distributions D_i: any valuation profile $v = (v_1, \ldots, v_n)$ occurs with probability $D(v) = \Pi_{i=1}^n D_i(v_i)$, where $D_i(v_i)$ is the probability that bidder i

[4] Observe that $X_i(b)$ is actually a random variable, but since the event that not all items are sold in second-price auctions only occurs with very low probability we shall often refer to $X_i(b)$ as indicating a specific bundle of items.

has the valuation function v_i. Let $V_{-i} = \times_{k \neq i} V_k$, $D_{-i} = \times_{k \neq i} D_k$, and $v_{-i} = (v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$.

A *bidding-function* B_i for player i is a function that assigns a bid-vector $b_i = B_i(v_i)$ to every valuation function $v_i \in V_i$. The reader may find it helpful to think of B_i as a suggestion made to player i by the auctioneer as to which bid to submit. An n-tuple of bidding-functions $B = (B_1, \ldots, B_n)$ is a *Bayesian Nash equilibrium* if for every $i \in [n]$, and for every valuation function v_i, the bid $B_i(v_i)$ maximizes i's expected utility given that his valuation function is v_i, and that the bid of every other bidder j is $B_j(v_j)$, where v_j is drawn from D_j. That is, a Bayesian Nash maximizes i's expected payoff for any valuation function he may have, given his beliefs about the other bidders.

Bayesian Price of Anarchy. For a fixed valuation profile of the bidders $v = (v_1, \ldots, v_n)$, the optimal social-welfare is $OPT(v) = max_{S_1,\ldots,S_n} \Sigma_i v_i(S_i)$, where the maximum is taken over all partitions of M into disjoint bundles $S_1, \ldots, S_n$. For given D, the (expected) optimal social-welfare $SW(OPT)$ is the expectation $E[OPT(v)]$, where v is drawn from D. That is,

$$SW(OPT) = \sum_{v \in V} D(v) OPT(v)$$

Given a profile v, every pure strategy profile b induces a social-welfare value $SW(b) = \sum_{i \in [n]} v_i(X_i(b))$. For an n-tuple of bidding-functions $B = (B_1, \ldots, B_n)$, we denote by $SW(B)$ the expected social welfare $E[SW(B_1(v_1), \ldots, B_n(v_n))]$, where the $v = (v_1, \ldots, v_n)$ is drawn from D :

$$SW(B) = \sum_{v \in V} D(v) SW(B(v)).$$

We are interested in Bayesian Nash equilibria B for which the ratio $\frac{SW(OPT)}{SW(B)}$ is small. The *Bayesian price of anarchy* of a game is

$$PoA = \max_{D,\ B\ BayesianNash} \frac{SW(OPT)}{SW(B)},$$

that is the maximum of the expression $\frac{SW(OPT)}{SW(B)}$, taken over *all* probability distributions D, and *all* Bayesian Nash equilibria B (for these probability distributions). Intuitively, a Bayesian price of anarchy of α means that no matter what the bidders' beliefs are, every Bayesian Nash equilibrium provides an α-approximation to the optimal social-welfare.

Supporting Bids. Due to ex-post individual-rationality, a bidder will never submit a bid that might result in a negative payoff. Recall that the rules of the auction dictate that it is possible that a bidder get *any* subset of the items for which he submitted his non-zero bids and be charged the sum of these bids. Hence, a bidder's bid must uphold the following property:

Definition 1. *A bid vector b_i is said to be a supporting bid given a valuation v_i, if for all $S \subseteq M$ $v_i(S) \geq \Sigma_{j \in S} b_i(j)$.*

Recall that the event that not all items are sold in second-price auctions only occurs with negligible probability. Hence, for simplicity, we shall disregard this event and prove our theorem for the case that all items are sold in second-price auctions.

2.2 Bayesian Price of Anarchy of 2

This subsection exhibits our main result. For ease of exposition we prove the theorem regarding the Bayesian price of anarchy for pure Bayesian Nash equilibria. The (more complicated) proof for mixed Nash is omitted due to space limitations and appears in the full version of the paper. The proof of the theorem exploits the fractional subadditivity of the valuations via the following lemma:

Lemma 1. *Let S be a set of items, and a_i be a maximizing additive valuation of player i for this set, restricted so that $a_i = 0$ on $M \backslash S$. If i bids according to a_i, while all the others bid according to any pure profile b_{-i}, then*

$$u_i(a_i, b_{-i}) \geq v_i(S) - Bids_{-i}(S, b_{-i}).$$

Proof. Let $X_i := X_i(a_i, b_{-i})$ be the set of items that player i is going to get. Note that if i wins any item $j \notin S$ then the maximum bid on this j was 0. Thus we can assume w.l.o.g. that $X_i \subseteq S$. Moreover, $a_i(j) - Bids_{-i}(\{j\}, b_{-i}) \leq 0$ holds for every non-obtained item $j \in S - X_i$. Therefore, we have

$$\begin{aligned} u_i(a, b_{-i}) &= v_i(X_i) - Bids_{-i}(X_i, b_{-i}) \\ &\geq \sum_{j \in X_i} a_i(j) - Bids_{-i}(X_i, b_{-i}) \\ &\geq \sum_{j \in S} a_i(j) - Bids_{-i}(S, b_{-i}) \\ &= v_i(S) - Bids_{-i}(S, b_{-i}). \end{aligned}$$

Theorem 1. *Let D be a distribution over fractionally-subadditive valuations of the bidders. If $B = (B_1, \ldots, B_n)$ is a Bayesian Nash, such that each B_i maps every valuation function v_i to a supporting bid (given v_i) then $\frac{SW(OPT)}{SW(B)} \leq 2$.*

Proof. Let $v = (v_1, \ldots, v_n)$ be a fixed valuation profile. We denote by $O^v = (O_1^v, \ldots, O_n^v)$ the optimum allocation with respect to profile v.

Now for every player i, let a_i denote the maximizing additive valuation for the set O_i^v, (in particular, $a_i(j) = 0$ if $j \notin O_i^v$). For all i, we consider a_i as an alternative strategy to $B_i(v_i)$.

Let us fix a bidder i. Let w_{-i} be an arbitrary valuation profile of all bidders except for i. We introduce the short notation

$$X_i^{w_{-i}} \stackrel{\text{def}}{=} X_i(B_i(v_i), B_{-i}(w_{-i})).$$

Furthermore, for any $S \subseteq M$ we will use

$$Bids_{-i}^{w_{-i}}(S) \stackrel{\text{def}}{=} Bids_{-i}(S, B_{-i}(w_{-i})),$$

resp.

$$Bids^{w}(S) \stackrel{\text{def}}{=} Bids(S, B(w)),$$

where $w = (w_i, w_{-i})$ is a valuation profile.

Since B is a Bayesian Nash, the strategy $B_i(v_i)$ provides higher expected utility to player i than the strategy a_i :

$$\sum_{w_{-i} \in V_{-i}} D(w_{-i}) u_i(B_i(v_i), B_{-i}(w_{-i})) \geq \sum_{w_{-i} \in V_{-i}} D(w_{-i}) u_i(a_i, B_{-i}(w_{-i})).$$

The utility values on the left-hand-side are

$$u_i(B_i(v_i), B_{-i}(w_{-i})) = v_i(X_i^{w_{-i}}) - Bids_{-i}^{w_{-i}}(X_i^{w_{-i}}) \leq v_i(X_i^{w_{-i}}).$$

On the right-hand-side, applying Lemma 1 yields

$$u_i(a_i, B_{-i}(w_{-i})) \geq v_i(O_i^v) - Bids_{-i}^{w_{-i}}(O_i^v).$$

By merging the inequalities above, we get

$$\begin{aligned}
\sum_{w_{-i} \in V_{-i}} D(w_{-i}) v_i(X_i^{w_{-i}}) &\geq \sum_{w_{-i} \in V_{-i}} D(w_{-i}) [v_i(O_i^v) - Bids_{-i}^{w_{-i}}(O_i^v)] \\
&= v_i(O_i^v) \sum_{w_{-i} \in V_{-i}} D(w_{-i}) - \sum_{w_{-i} \in V_{-i}} D(w_{-i}) Bids_{-i}^{w_{-i}}(O_i^v) \\
&= v_i(O_i^v) \cdot 1 - \sum_{w \in V} D(w) Bids_{-i}^{w_{-i}}(O_i^v) \\
&\geq v_i(O_i^v) - \sum_{w \in V} D(w) Bids^{w}(O_i^v).
\end{aligned}$$

Here the expected highest bids $\sum_{w_{-i} \in V_{-i}} D(w_{-i}) Bids_{-i}^{w_{-i}}(O_i^v)$, and $\sum_{w \in V} D(w) Bids_{-i}^{w_{-i}}(O_i^v)$ are equal, because D is independent for all bidders. Finally, $Bids_{-i}^{w_{-i}}(O_i^v) \leq Bids^{w}(O_i^v)$ obviously holds, since in the latter case we consider maximum bids over a larger set of players. We obtained

$$v_i(O_i^v) \leq \sum_{w_{-i} \in V_{-i}} D(w_{-i}) v_i(X_i^{w_{-i}}) + \sum_{w \in V} D(w) Bids^{w}(O_i^v).$$

We sum over all i, and then take the expectation over all valuations $v = (v_1, \ldots, v_n)$ on both sides:

$$\begin{aligned}
\sum_{v \in V} D(v) \sum_{i \in [n]} v_i(O_i^v) \leq &\sum_{v \in V} D(v) \sum_{i \in [n]} \sum_{w_{-i} \in V_{-i}} D(w_{-i}) v_i(X_i^{w_{-i}}) \\
&+ \sum_{v \in V} D(v) \sum_{i \in [n]} \sum_{w \in V} D(w) Bids^{w}(O_i^v).
\end{aligned}$$

Note that $\sum_{v\in V} D(v) \sum_{i\in[n]} v_i(O_i^v) = SW(OPT)$. Furthermore, we claim that both summands on the right-hand-side are at most $SW(B)$, so that $SW(OPT) \leq 2SW(B)$, which will conclude the proof. The first summand is

$$\begin{aligned}
&\sum_{i\in[n]} \sum_{v_i\in V_i} D(v_i) \sum_{v_{-i}\in V_{-i}} D(v_{-i}) \sum_{w_{-i}\in V_{-i}} D(w_{-i}) v_i(X_i^{w_{-i}}) \\
&= \sum_{i\in[n]} \sum_{v_i\in V_i} D(v_i) \sum_{w_{-i}\in V_{-i}} D(w_{-i}) v_i(X_i^{w_{-i}}) \sum_{v_{-i}\in V_{-i}} D(v_{-i}) \\
&= \sum_{i\in[n]} \sum_{v_i\in V_i} \sum_{w_{-i}\in V_{-i}} D(v_i) D(w_{-i}) v_i(X_i^{w_{-i}}) \cdot 1 \\
&= \sum_{i\in[n]} \sum_{v\in V} D(v) v_i(X_i^{v_{-i}}) \\
&= \sum_{v\in V} D(v) \sum_{i\in[n]} v_i(X_i(B(v))) = SW(B).
\end{aligned}$$

Finally, the second summand is

$$\begin{aligned}
\sum_{v\in V} D(v) \sum_{w\in V} D(w) \sum_{i\in[n]} Bids^w(O_i^v) &= \sum_{v\in V} D(v) \sum_{w\in V} D(w) Bids^w(M) \\
&= \sum_{w\in V} D(w) Bids^w(M) \sum_{v\in V} D(v) \\
&= \sum_{w\in V} D(w) Bids^w(M) \cdot 1 \\
&= \sum_{w\in V} D(w) \sum_{i\in[n]} Bids_i(X_i(B(w)), B_i(w_i)) \\
&\leq \sum_{w\in V} D(w) \sum_{i\in[n]} w_i(X_i(B(w))) = SW(B).
\end{aligned}$$

The last inequality holds since for all i, the $B_i(w_i)$ contains supporting bids for any set of items including the obtained set $X_i(B(w))$.

A simple example shows that even in the full-information setting, this Bayesian price of anarchy result is tight.

3 Computing Pure Nash Equilibria

In this section we consider the following full-information game: The m items are sold to n bidders with fractionally-subadditive valuation functions in m independent second-price auctions. The players' valuation functions are assumed to be common knowledge.

In Subsection 3.1, we show that a pure Nash that provides a good approximation to the social welfare always exists in such games and provide a constructive way of finding one. In fact, we also prove that the price of stability [1] is 1, i.e. the optimum can always be achieved in a Nash equilibrium.

In Subsection 3.2 we show that if bidders have submodular valuation functions then such a pure Nash can be reached in polynomial time.

3.1 Fractionally-Subadditive Valuation Functions

Despite the fact that (as shown in the Introduction) some Nash equilibria may fail to provide good approximation to the social-welfare, we present a constructive way for finding a pure Nash that yields a 2-approximation. We introduce a natural procedure we call the Potential Procedure which always reaches such an equilibrium. The Potential Procedure is a simple myopic procedure for fractionally-subadditive bidders[5].

For every i let $A_i = \{a_1^i, ..., a_{l_i}^i\}$ be a set of additive valuations such that for every $S \subseteq M$ $v_i(S) = \max_{a \in A_i} a(S)$. Recall that since v_i is fractionally subadditive such A_i must exist. Informally, the Potential Procedure simply starts with some arbitrary supporting bids (corresponding to some maximizing additive valuations) and let players best-reply, one by one, to the bids of other players by switching to new supporting bids.

The Procedure

1. Initialize $b_i^*(j) \leftarrow 0, S_i \leftarrow \emptyset, r_j \leftarrow 0$, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.
2. While there is a bidder i such that $S_i \neq \arg\max_{S \subseteq M}(v_i(S) - \Sigma_{j \in (S \setminus S_i)} r_j)$:
   (a) Let $T = \arg\max_{S \subseteq M}(v_i(S) - \Sigma_{j \in (S \setminus S_i)} r_j)$. Let $a \in A_i$ be such that $v_i(T) = a(T)$.
   (b) Set $b_i^*(j) \leftarrow 0$ and $r_j \leftarrow 0$ for all $j \in S_i$.
   (c) Set $b_i^*(j) \leftarrow a(j)$ for all $j \in T$. Set $r_j \leftarrow a(j)$ for all $j \in T$.
   (d) Set $S_i \leftarrow T$.
   (e) For all $k \neq i$ set $S_k \leftarrow S_k \setminus S_i$, and set $b_k^*(j) \leftarrow 0$ for all $j \in S_i$.
3. Output $b^* = (b_1^*, \ldots, b_n^*)$.

Observe that in the definition we do not require $a(j) \geq r_j$ in lines (a)–(c) (i.e., that r_j increase). However, this follows from the fact that T maximizes $v_i(S) - \Sigma_{j \in (S \setminus S_i)} r_j$. We use a potential-function argument and the fractional-subadditivity of the bidders to show that the Potential Procedure eventually converges to a "good" pure Nash.

Theorem 2. *If the valuation functions are fractionally subadditive, then the* Potential Procedure *converges to a pure Nash equilibrium that provides a* 2*-approximation to the optimal social-welfare.*

[5] This procedure requires bidder i to be able to determine which bundle he would prefer most, given a vector of per-item payments $r = (r_1, ..., r_m)$. That is, to declare a bundle S for which $v_i(S) - \Sigma_{j \in S} r_j$ is maximized. This type of query is called a *demand query* and is very common in combinatorial auctions literature (see, e.g., [2,4,6,7]).

We note that the proof shows that even the optimal social-welfare can be obtained in a pure Nash equilibrium and so the price of stability is 1. So, we have a natural procedure, that is essentially a best response sequence of the players, that leads to a pure Nash equilibrium. But, how long will it take the POTENTIAL PROCEDURE to converge? A non-trivial construction shows that unfortunately the worst case running time is exponential in n and m.

Theorem 3. *There is an instance with* 2 *bidders, each with a fractionally-subadditive valuation function, on which the* POTENTIAL PROCEDURE *converges after* $\Omega(2^m)$ *steps.*

Theorem 3 leaves us with two interesting open questions: First, will the POTENTIAL PROCEDURE converge in polynomial time if the valuation functions are submodular? Second, does the POTENTIAL PROCEDURE always run in time that is polynomial in the size of the sets of additive valuations that underlie every fractionally-subadditive valuation? An affirmative answer to this question would imply that the POTENTIAL PROCEDURE runs in polynomial time if the bidders have fractionally-subadditive valuations encoded in a bidding language (see [14,13,4,5]). We note that in the instance in the proof of Theorem 3 the size of the sets of additive valuations was exponential n and m.

3.2 Submodular Valuation Functions

In this subsection, we focus on submodular valuation functions. We show that one can find, in polynomial time, a pure Nash equilibrium that also satisfies the premises of Theorem 1. The procedure we present exploits the algorithm due to Lehmann et al. [13]. This procedure, which we will call the MARGINAL-VALUE PROCEDURE, therefore provides a 2-approximation to the optimal social welfare.

The Procedure

1. Fix an arbitrary order on the items. W.l.o.g. let this order be $1, \ldots, m$.
2. Initialize $S_i \leftarrow \emptyset$, and $r_j \leftarrow 0$, for $i = 1, \ldots, n$, and $j = 1, \ldots, m$.
3. For each item $j = 1, \ldots, m$:
   (a) Let $i = \arg\max_{t \in N} \; v_t(S_t \cup \{j\}) - v_t(S_t)$. Set $S_i \leftarrow S_i \cup \{j\}$.
   (b) Set $r_j \leftarrow \max_{t \in N} \; v_t(S_t \cup \{j\}) - v_t(S_t)$.
4. For every bidder i set $b_i^*(j) \leftarrow r_j$ for all $j \in S_i$ and $b_i^*(j) \leftarrow 0$ for all $j \notin S_i$.
5. Output $b^* = (b_1^*, ..., b_n^*)$.

Observe, that the resulting n-tuple of bid-vectors b^* is such that for each item, only one bidder offers a non-zero bid for that item. This is due to the fact that we are dealing with a complete-information setting (intuitively, if a bidder does not win an item he might as well not bid on it). Also notice that the MARGINAL-VALUE PROCEDURE only requires m rounds and so ends in polynomial time.

Theorem 4. *If the valuation functions are submodular then a pure Nash equilibrium that provides a* 2*-approximation to the optimal social-welfare can be computed in polynomial time.*

Acknowledgements

We thank Noam Nisan for enriching conversations that lead to the writing of this paper.

References

1. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, E., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: FOCS 2004, pp. 295–304 (2004)
2. Blumrosen, L., Nisan, N.: On the computational power of iterative auctions I: demand queries. Discussion paper no. 381, The Center for the Study of Rationality, The Hebrew University. An extended abstract in EC 2005 contained preliminary results (2005)
3. Clarke, E.H.: Multipart pricing of public goods. Public Choice, 17–33 (1971)
4. Dobzinski, S., Nisan, N., Schapira, M.: Approximation algorithms for combinatorial auctionss with complement-free bidders. In: The 37th ACM symposium on theory of computing (STOC) (2005)
5. Dobzinski, S., Schapira, M.: An improved approximation algorithm for combinatorial auctions with submodular bidders. In: SODA 2006 (2006)
6. Feige, U.: On maximizing welfare where the utility functions are subadditive. In: STOC 2006 (2006)
7. Feige, U., Vondrak, J.: The allocation problem with submodular utility functions (manuscript, 2006)
8. Gairing, M., Monien, B., Tiemann, K.: Selfish routing with incomplete information. In: Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms (SPAA), pp. 203–212 (2005)
9. Garg, D., Narahari, Y.: Price of anarchy of network routing games with incomplete information. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 1066–1075. Springer, Heidelberg (2005)
10. Groves, T.: Incentives in teams. Econometrica, 617–631 (1973)
11. Harsanyi, J.C.: Games with incomplete information played by 'bayesian' players, parts i ii and iii. Management science 14 (1967-1968)
12. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science, pp. 404–413 (1999)
13. Lehmann, B., Lehmann, D., Nisan, N.: Combinatorial auctions with decreasing marginal utilities. In: ACM conference on electronic commerce (2001)
14. Nisan, N.: Bidding and allocation in combinatorial auctions. In: ACM Conference on Electronic Commerce (2000)
15. Nisan, N., Ronen, A.: Algorithmic mechanism design. In: STOC (1999)
16. Nisan, N., Ronen, A.: Computationally feasible vcg-based mechanisms. In: ACM Conference on Electronic Commerce (2000)
17. Roughgarden, T., Tardos, E.: How bad is selfish routing? Journal of the ACM 49(2), 236–259 (2002)

18. Vetta, A.: Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In: FOCS. IEEE Computer Society, Los Alamitos (2002)
19. Vickrey, W.: Counterspeculation, auctions and competitive sealed tenders. Journal of Finance, 8–37 (1961)
20. Vondrak, J.: Optimal approximation for the submodular welfare problem in the value oracle model. In: STOC 2008 (2008)

Truthful Unification Framework for Packing Integer Programs with Choices*

Yossi Azar[1] and Iftah Gamzu[2]

[1] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA and School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel
azar@post.tau.ac.il
[2] School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel
iftgam@post.tau.ac.il

Abstract. One of the most interesting research directions within the field of algorithmic mechanism design revolves the study of hard combinatorial optimization problems. In this setting, many common algorithmic techniques cannot be utilized as they violate certain monotonicity properties that are imperative for truthfulness. Consequently, it seems of the essence to develop alternative methods, which can underlie truthful mechanisms. In particular, since many problems can be formulated as instances of integer linear programs, it seems that devising techniques that apply to integer linear programs is significantly important.

In this paper, we focus our attention on packing integer programs with choices. Our main findings can be briefly summarized as follows:

1. We develop a framework, which can be used as a building block to approximately solve packing integer programs with choices. The framework is built upon a novel unification technique that approximately solves an instance of a packing integer program with choices, given algorithms that approximately solve sub-instances of it. The framework is deterministic and monotone, and hence can underlie truthful *deterministic* mechanisms.
2. We demonstrate the applicability of the framework by applying it to several NP-hard problems. In particular, we focus on the *bandwidth allocation problem in tree networks*, and the *multiple knapsack problem on bipartite graphs*. Notably, using the mentioned framework, we attain the first non-trivial approximation guarantees for these problems in a game theoretic setting.

1 Introduction

The field of *algorithmic mechanism design*, which was introduced by Nisan and Ronen [25], studies the design of protocols or mechanisms for algorithmic

* Proofs and details omitted from this extended abstract appear in the full version of this paper. The first author was supported in part by the Israel Science Foundation and by the German-Israeli Foundation.The second author was supported by the Binational Science Foundation, by the Israel Science Foundation, and by the European Commission under the Integrated Project QAP funded by the IST directorate as Contract Number 015848.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 833–844, 2008.

problems in scenarios where the input is presented by *strategic agents*. A strategic agent might be dishonest about its part of the input in order to manipulate the protocol in a way that will maximize its own gain. A primary interest of algorithmic mechanism design is in the development of efficiently computable *truthful* protocols, which are robust against manipulation by agents, i.e., every agent is rationally motivated to truthfully report its input.

One of the most intriguing research directions within the field of algorithmic mechanism design revolves the study of hard combinatorial optimization problems. These problems are hard in a sense that any computationally-efficient algorithm designed to solve them can only attain sub-optimal guarantees, that is, they only admit approximation algorithms. The motivation for this research realm originates in two inherent difficulties that arise in the alluded setting. The first is that the most fundamental technique of mechanism design, namely the Vickrey-Clarke-Groves (VCG) mechanism [32,14,19], cannot be utilized since it must built upon an algorithm that obtains an exact optimal solution [21,24]. The other is that many common algorithmic methods cannot be employed as they violate certain *monotonicity* properties that are imperative for truthfulness.

In light of this state of affairs, it seems of the essence to develop modifications and alternatives to generic algorithmic techniques, so they could underlie truthful mechanisms. In particular, since many optimization problems can be formulated as instances of integer linear programs, it seems that devising methods that apply to integer linear programs is significantly important. This perception reinforces by the observation that one of the most efficient ways to approximately solve such programs, i.e., linear programming-based randomized rounding [29,28,30], generally fails to be monotone (see, e.g., the discussion in [2]).

1.1 Our Results

The main contribution of this paper is in presenting a framework that can be used as a building block to approximately solve packing integer programs, and packing integer programs with choices. The framework is built upon a unification technique, which approximately solves an instance of a packing integer program with choices, given algorithms that approximately solve sub-instances of it. The framework is deterministic and monotone, and thus can underlie truthful *deterministic* mechanisms. In order to demonstrate the applicability and strength of this framework, we apply it to the bandwidth allocation problem in tree networks, and the multiple knapsack problem on bipartite graphs.

Truthful deterministic unification framework. We focus on truthfully approximating maximization problems, which can be posed as packing integer programs, and packing integer programs with choices of the forms exhibited in Figure 1. Essentially, we concentrate on the class of packing integer programs with choices since it incorporates the class of packing integer programs as a special case. In a packing integer program with choices, there are n variables that are partitioned into ℓ pairwise-disjoint sets $C_1, C_2, \ldots, C_\ell$. The objective is to maximize a linear value function over these variables, subject to a set of packing constraints, and a set of choices constraints, which prohibit the selection of more than one

$\textbf{max} \quad v \cdot x$	$\textbf{max} \quad v \cdot (C \cdot x)$
$\textbf{s.t.} \quad A \cdot x \leq b$	$\textbf{s.t.} \quad A \cdot x \leq b$
$x \in \{0,1\}^n$	$C \cdot x \leq 1^\ell$
	$x \in \{0,1\}^n$

Fig. 1. Packing integer program (left) and packing integer program with choices (right). Note that $A \in \mathbb{R}_+^{m \times n}$, $b \in \mathbb{R}_+^m$, and $v \in \mathbb{R}_+^n$ ($v \in \mathbb{R}_+^\ell$) in the packing (packing with choices) case. Also remark that $C \in \{0,1\}^{\ell \times n}$ is a *choices matrix* in which the i-th row corresponds to the variables set C_i, that is, the i-th row has ones in the entries that coincide with the variables in C_i and zeros elsewhere. Notice that the class of packing integer programs is obtained as a special case of packing integer programs with choices when C is the $n \times n$ identity matrix.

variable from each set. From a mechanism design point of view, there are ℓ strategic *single parameter* agents, each of which coincides with a valuation and may be untruthful about it. The goal is to maximize the *social welfare*. A natural approach for approximating a packing integer program with choices is to partition it to sub-instances, solve each of them, and output the sub-solution that has a maximum value. Unfortunately, this approach fails to be monotone, unless the underlying sub-algorithms are *bitonic* [23]. In order to avoid this shortcoming, we devise a deterministic unification technique, easing the requirements from the sub-algorithms, and still attaining monotonicity and provable approximation guarantee. This result appears in Section 3.

Bandwidth allocation in tree networks. An instance of the *bandwidth allocation problem in tree networks* consists of an undirected tree T with m edges, and a set $\mathcal{R}$ of ℓ connection requests such that every request $r \in \mathcal{R}$ is characterized by a quadruple (s_r, t_r, d_r, v_r), where s_r and t_r are the respective *source* and *target* vertices of the request, $d_r \in (0,1]$ is the *demand* associated with the request, and v_r is the positive *value* gained by allocating the request. The goal is to select a maximum value subset of requests $S \subseteq \mathcal{R}$ so that the aggregate demands of the requests in S, which cross any edge, does not exceed the *bandwidth capacity*, which is unit. Letting P_r denote the unique simple path between s_r and t_r in T, this problem can be posed by the following packing integer program:

$$\begin{aligned} \text{maximize} \quad & \sum_{r \in \mathcal{R}} v_r x_r & \\ \text{subject to} \quad & \sum_{r \in \mathcal{R} \mid e \in P_r} d_r x_r \leq 1 & \forall\, e \in E \\ & x_r \in \{0,1\} & \forall\, r \in \mathcal{R} \end{aligned}$$

In the game theoretic version of this problem there are ℓ strategic agents, each of which controls a request and may be dishonest about its value. We study this variant using the aforesaid framework, and devise a monotone deterministic algorithm, which achieves an approximation ratio of $O(\ln \ln m)$. This result implies

a corresponding truthful mechanism, and is the first non-trivial deterministic result for this problem. The specifics of this finding are presented in Section 4.

Multiple knapsack on bipartite graphs. In the *multiple knapsack problem on bipartite graphs*, we are given a set M of m unit-capacity knapsacks, and a set $\mathcal{I}$ of ℓ items such that every item $i \in \mathcal{I}$ is characterized by a pair (s_i, v_i), where $s_i \in (0, 1]$ is the *size* of the item, and v_i is the positive *value* gained by placing the item in one of the knapsacks. An additional ingredient of the input is an items-knapsacks bipartite graph, which represents the assignment restrictions of the items to the knapsacks. Particularly, the bipartite graph exhibits a set of admissible knapsacks $M_i \subseteq M$, for every $i \in \mathcal{I}$. The goal is to select a maximum value subset of items $S \subseteq \mathcal{I}$, along with an admissible knapsack for each selected item, so that all the items in S can be simultaneously placed in their designated knapsacks, while preserving the capacities of the knapsacks. This problem can be modelled by the following packing integer program with choices:

$$\begin{aligned}
\text{maximize} \quad & \sum_{i \in \mathcal{I}} v_i \cdot \Big(\sum_{j \in M_i} x_{ij} \Big) & \\
\text{subject to} \quad & \sum_{i \in \mathcal{I} | j \in M_i} s_i x_{ij} \leq 1 & \forall\, j \in [m] \\
& \sum_{j \in M_i} x_{ij} \leq 1 & \forall\, i \in \mathcal{I} \\
& x_{ij} \in \{0, 1\} & \forall\, i \in \mathcal{I},\, j \in M_i
\end{aligned}$$

In the game theoretic version of this problem there are ℓ strategic agents, each of which owns an item, and may be untruthful about its value. We consider this variant using the mentioned framework, and develop a monotone deterministic 11-approximation algorithm, which implies a corresponding truthful mechanism. This result is the first deterministic result for this problem, which does not assume constant number of knapsacks. Further details are provided in Section 5.

1.2 Related Work

It seems fundamentally important to develop modifications and alternatives to common techniques, which cannot be utilized in the context of algorithmic mechanism design. However, there are only a handful of results addressing this goal. Mu'alem and Nisan [23] seem to have been the first to pay attention to the development of such general tools. Primarily, they exhibited necessary conditions for the truthful utilization of two basic building blocks, which are the max and if-then-else operators. Briest, Krysta and Vöcking [9] continued this line of research. They devised a general approach to transform a pseudopolynomial algorithm into a monotone FPTAS, and demonstrated that primal-dual greedy algorithms may be the key to truthfully solve some integer linear programs. Later on, Lavi and Swamy [20] developed a general technique to convert an approximation algorithm in packing domains to a randomized approximation mechanism that is *truthful in expectation*. Finally, Babaioff, Lavi and Pavlov [5] presented a method that turns any given algorithm to a dominant-strategy mechanism

in single parameter domains. However, their method degrades the performance guarantee of the resulting mechanism by a factor of $O(\log \rho)$, where ρ denotes the ratio between the largest and smallest valuations.

Focusing on the two problems under consideration, the first observation one can make is that they are NP-hard as they generalize the *knapsack problem*. Accordingly, most of past research focused on developing approximation algorithms for them. The best known result for the bandwidth allocation problem in tree networks is by Lewin-Eytan, Naor and Orda [22], who presented a 5-approximation algorithm, based on the local ratio technique [7]. Recent years have also seen ever-growing line of work addressing variants of this problem using various algorithmic tools such as primal-dual approach [18], linear programming-based methods [27,13,6], and mixtures of several techniques [10,11,8]. Unfortunately, it is easy to demonstrate that most of these positive results are not monotone. Consequently, attaining a non-trivial deterministic approximation for the bandwidth allocation problem in an algorithmic mechanism design setting is still an open question, even when the underlying network is a line. On a different note, it is relevant to point out that Briest, Krysta and Vöcking [9] studied the mechanism design variant of the *unsplittable flow problem*, which is a generalization of the bandwidth allocation problem. Hence, their upper bound result follows to our case. Yet, when the demands of the requests are arbitrary, their algorithm achieves a performance guarantee that can be as high as a polynomial in m. Turning to the multiple knapsack problem on bipartite graphs, it is known to be approximable within a factor that is slightly better than $e/(e-1)$ by the work of Feige and Vondrák [16]. Similarly to the bandwidth allocation problem, past years have also seen respectable amount of research addressing variants of the multiple knapsack problem [15,12,26,17]. From an algorithmic mechanism design point of view, the only provable result is a monotone PTAS for a generalization of the problem, referred to as the *generalized assignment problem*. However, this result holds only when the number of knapsacks is fixed.

2 Preliminaries

In this section, we introduce the notation and terminology to be used throughout the paper, and describe a characterization that interlinks monotone algorithms with truthful mechanisms. We begin with the notation:

- Let Π denote a problem that can be posed as a packing integer program with choices. Essentially, Π can be considered to be a collection of (infinite) input instances, each of which represented by a quadruple (v, A, C, b), consisting of a concrete valuations vector v, constraints matrix A, choices matrix C, and constraints vector b.
- Let $\Pi_{|a}$ be a collection of input instances achieved by the restriction of the instances in Π according to an attribute a of the columns of the constraints matrix A. Namely, every instance $(v, A, C, b) \in \Pi$ gives rise to a sub-instance $(v_{|a}, A_{|a}, C_{|a}, b) \in \Pi_{|a}$ such that $A_{|a}$ consists of the subset of the columns of

A that satisfy the attribute a, $C_{|a}$ consists of the matching subset of columns of C, the set of variables $x_{|a}$ comprises of the suitable subset of variables of x, and $v_{|a}$ is the implied subset of entries of v, that is, the set of entries that have at least one allied variable in $x_{|a}$. For instance, in the bandwidth allocation problem in tree networks, every column of A represents a request. Thus, a non-trivial attribute may state "having a distance of 1". This attribute gives rise to sub-instances that only consists of requests that have a distance of 1 between their terminals.

– Given an input instance $I = (v, A, C, b)$, let $\mathrm{PIP}(I)$ denote the corresponding integer program instance, and let $\mathrm{LP}(I)$ be the relaxation of $\mathrm{PIP}(I)$ obtained by replacing the integrality constraint $x \in \{0,1\}^n$ with $x \in [0,1]^n$. Moreover, let $\mathrm{OPT}_I^{\mathrm{int}}$ and $\mathrm{OPT}_I^{\mathrm{frac}}$ be the values of the optimal solutions of $\mathrm{PIP}(I)$ and $\mathrm{LP}(I)$, respectively. Finally, let d_{Π} denote the integrality gap of problem Π, formally defined as $d_{\Pi} = \sup_{I \in \Pi} \mathrm{OPT}_I^{\mathrm{frac}} / \mathrm{OPT}_I^{\mathrm{int}}$. Note that for notational simplicity we will mark the integrality gap of $\Pi_{|a}$ by d_a.

We now present the notion of *monotonicity*, and then turn to describe a characterization that reduces the goal of designing truthful mechanisms to that of designing monotone algorithms. Remark that the illustrated terms are refined to the specific setting considered. Thus, the keen reader is encouraged to refer to [9], and the references therein for a brief introduction to the field of algorithmic mechanism design, and more comprehensive overview of the underlying concepts.

Definition 1. An algorithm $\mathcal{M}$ is said to be *monotone w.r.t. Π* if it satisfies the following property, for every $(v, A, C, b) \in \Pi$: if the solution x generated by $\mathcal{M}$ w.r.t. $\mathrm{PIP}(v, A, C, b)$ satisfies that agent i is chosen (i.e., $x_j = 1$ for some $j \in C_i$) then the solution $\tilde{x}$ generated by $\mathcal{M}$ w.r.t. $\mathrm{PIP}(\tilde{v}, A, C, b)$, where $\tilde{v}$ is a valuations vector in which $\tilde{v}_i \geq v_i$ and the other values are fixed, also satisfies that agent i is chosen (i.e., $\tilde{x}_{j'} = 1$ for some $j' \in C_i$, which might satisfy $j' \neq j$).

Theorem 2. ([23]) *If algorithm $\mathcal{M}$ is monotone w.r.t. Π then there is a matching truthful mechanism for Π, which can be efficiently computed using $\mathcal{M}$.*

3 A Truthful Unification Framework

In this section, we present a deterministic unification framework that can be used to truthfully approximate maximization problems, which can be represented by the packing integer program with choices described in Figure 1. As priorly indicated, one widely accepted approach for approximating a packing integer program with choices is to solve sub-instances of it, and then to pick the best sub-solution. Unfortunately, this algorithmic method fails to be monotone, unless the underlying algorithms are monotone and bitonic[1]. The main problem in

[1] Informally, an algorithm is *bitonic* if its outcome value as a function of the value of any single agent i has the pattern that it does not increase as long as agent i is not chosen, and does not decrease as long as agent i is chosen. A formal definition can be found in [23], and in [9].

picking the best sub-solution resides in the "selection-outcome linkage", that is, the selection of which sub-solution to output coincides with the outcomes of the underlying algorithms. Consequently, if one of the underlying algorithms is not bitonic, e.g., if its outcome value decreases when the value of a chosen agent increases, there may be settings in which an increase in the value of this agent will result in the selection of a different sub-solution in which this agent is not chosen. In the following, we design a method that breaks this link. Prior to describing the finer details of our approach, we introduce a definition that formalizes what is a monotone algorithm that generates sub-solutions for sub-instances.

Definition 3. $\mathcal{M}$ is *monotone a-restrictive c-approximation algorithm w.r.t.* Π if it is monotone w.r.t. Π, and its solution x to PIP(I) satisfies the following properties, for any $I = (v, A, C, b) \in \Pi$:

- x is restricted w.r.t. the attribute a, i.e., $x_j = 1$ only if $j \in C_i$ and $v_i \in v_{|a}$.
- the value of the solution is at least $\mathrm{OPT}^{\mathrm{int}}_{I_a}/c$, where $I_a = (v_{|a}, A_{|a}, C_{|a}, b)$.

We are now ready to establish the main result of this section. Let $a_1, a_2, \ldots, a_k$ be a collection of *pairwise-disjoint choices-consistent* attributes w.r.t. Π. Specifically, these attributes partition every instance $(v, A, C, b) \in \Pi$ to k sub-instances $(v_{|a_1}, A_{|a_1}, C_{|a_1}, b), \ldots, (v_{|a_k}, A_{|a_k}, C_{|a_k}, b)$ in a way that maintains the following two properties. The first property is *pairwise-disjointness*, which means that every column of A satisfies exactly one attribute. This property guarantees that every column of A, every column of C, and every variable of x appears in exactly one restricted sub-instance. The second property is *choices-consistent*, which means that all the variables allied to some entry of v are picked by the same attribute. This property assures that every entry of v appears in exactly one restricted sub-instance. Notice that the last attribute is trivially maintained for (pure) packing integer programs by any pairwise-disjoint collection of attributes.

Algorithm 1. MAX-Select(I)

Input: An instance $I = (v, A, C, b) \in \Pi$
Output: A solution x

1: **for** $i = 1$ to k **do**
2: **Calculate** $\mathrm{OPT}^{\mathrm{frac}}_{I_i}$ exactly by solving LP(I_i), where $I_i = (v_{|a_i}, A_{|a_i}, C_{|a_i}, b)$
3: **end for**
4: **Let** j be the minimal index for which $\mathrm{OPT}^{\mathrm{frac}}_{I_j}/c_{a_j}d_{a_j} \geq \mathrm{OPT}^{\mathrm{frac}}_{I_i}/c_{a_i}d_{a_i}$, $\forall\, i \in [k]$
5: **Simulate** algorithm $\mathcal{M}_j(I)$ to obtain x
6: **return** x

Theorem 4. *Given a family* $\{\mathcal{M}_i\}_{i=1}^k$ *of deterministic algorithms, where each* $\mathcal{M}_i$ *is monotone* a_i*-restrictive* c_{a_i}*-approximation algorithm w.r.t.* Π*,* MAX-Select *is monotone deterministic* $\sum_{i=1}^k c_{a_i}d_{a_i}$*-approximation algorithm w.r.t.* Π*.*

Corollary 5. *Given a family of algorithms as specified in the above-mentioned theorem,* MAX-Select *supports a truthful deterministic mechanism that approximates* Π *to within a ratio of* $\sum_{i=1}^k c_{a_i}d_{a_i}$*.*

Before we turn to demonstrate the applicability of the framework, let us briefly outline the key goals one needs to address in order to utilize it. The first issue that needs to be dealt is the *attributes*. Namely, one should define a collection of pairwise-disjoint choices-consistent attributes, which partition every instance of the problem under consideration. The second issue that ought to be resolved is the *algorithms*. In particular, one needs to develop a family of monotone deterministic algorithms, which are restrictive w.r.t. the different attributes. The last matter to be handled is the *integrality gaps*. That is, one should analyze the integrality gaps of the sub-problems defined w.r.t. each attribute.

4 The Bandwidth Allocation Problem in Tree Networks

In this section, we study the bandwidth allocation problem in tree networks, and develop a monotone deterministic $O(\ln \ln m)$-approximation algorithm, which utilizes the framework exhibited in Section 3. Our approach is framework-guided. Namely, we begin by studying restricted versions of the problem under consideration in which some characteristics of the requests are guaranteed to have a certain pattern. Essentially, these characteristics are the attributes that partition every unrestrictive input instance to sub-instances. For each restricted case, we design a monotone deterministic algorithm that approximately solves it, and analyze the integrality gap of the corresponding packing integer program. Later on, we show how to consolidate these results within the framework, and yield the aforementioned outcome.

Restricted demands. We consider a restricted version of the problem in which the demand of every request is guaranteed to have a certain pattern. Particularly, we investigate instances in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$, where $i \in \mathbb{N}$. The integrality gap of this restricted version is known to be bounded by a constant, for any $i \in \mathbb{N}$, by the work of Chekuri, Mydlarz and Shepherd [13]. We now introduce a family of monotone deterministic algorithms $\{\mathsf{BAP}_i\}_{i \in \mathbb{N}}$, which approximately solve the restricted versions of the problem. Namely, algorithm BAP_i handles restricted instances in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$. Prior to describing the algorithm, let us consider an inherently simpler scenario, in which the demand of each request is unit. This special case is equivalent to the *edge disjoint paths problem in a tree*, and is known to admit a deterministic polynomial-time optimal algorithm [31], which will henceforth be referred to as algorithm wEDP. Now, we are ready to describe algorithm BAP_i. Algorithm BAP_i has two steps. First, it rounds-up all the demands of the requests to 2^{-i}. Then, it executes algorithm wEDP for 2^i times, where the input to the j-th execution of wEDP is the set of requests that have not been selected in the first $j-1$ executions of wEDP. Informally, one may picture the second step as an iterative packing of edge disjoint paths in different layers, each of bandwidth 2^{-i}. It is worth noting that the algorithm, and its analysis are partially inspired by the technique suggested by Awerbuch et al. [3] for reducing a call admission problem in multi-wavelength scenario to that of a single-wavelength case.

Algorithm 2. $\mathsf{BAP}_i(\mathcal{R})$

Input: A requests instance $\mathcal{R}$ in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$
Output: A subset $S \subseteq \mathcal{R}$ of selected connection requests

1: **Let** $k = 2^i$
2: **Let** $\mathcal{R}'$ be the set of requests $\{(s_r, t_r, 1/k, v_r) : r \in \mathcal{R}\}$
3: **for** $j = 1$ to k **do**
4: **Simulate** algorithm wEDP on $\mathcal{R}'$ to obtain S_j
5: Let $\mathcal{R}' = \mathcal{R}' \setminus S_j$
6: **end for**
7: **return** $\bigcup_{j=1}^{k} S_j$

Theorem 6. *Let $\mathcal{R}$ be an instance in which $d_r \in (2^{-(i+1)}, 2^{-i}]$, for every $r \in \mathcal{R}$. Algorithm $\mathsf{BAP}_i(\mathcal{R})$ is monotone, deterministic, and outputs a feasible solution whose value is at least $1/12$ of an optimal solution's value.*

Narrow demands. We turn to examine the restricted version of the problem in which the demand of every request is guaranteed to be *narrow*, that is $d_r \in (0, O(1/\ln m)]$, for every $r \in \mathcal{R}$. The integrality gap of this restricted version is known to be $1 + \epsilon$ by the outcome of algorithms that employ the randomized rounding technique [29,28,30]. Furthermore, this restricted setting is a special case of the $\Omega(\ln m)$-*bounded unsplittable flow problem*, which is known to admit a polynomial-time monotone deterministic algorithm, attaining constant approximation [9,4]. Consequently, referring to the algorithm of [9] as BAP, we obtain the following theorem.

Theorem 7. ([9]) *Let $\mathcal{R}$ be an instance in which $d_r \in (0, 1/(100 \ln m)]$, for every $r \in \mathcal{R}$. Algorithm $\mathsf{BAP}(\mathcal{R})$ is monotone, deterministic, and outputs a feasible solution whose value is at least $1/3$ of an optimal solution's value.*

Integration of the results within the framework. We now demonstrate how to consolidate the results for the restricted versions of the problem under the umbrella of the framework. Fundamentally, we have already attained the key ingredients needed to employ the main theorem of the framework. Let $k = c \ln \ln m + 1$, where c is a sufficiently large constant for which $2^{-(c \ln \ln m)} \leq 1/(100 \ln m)$. Let $a_1, a_2, \ldots, a_k$ be attributes that restrict input instances of the problem according to the demands of the requests. Explicitly, attribute a_1 picks out all the requests that have a demand in the range of $(2^{-1}, 1]$, attribute a_2 picks out the requests with a demand in the range of $(2^{-2}, 2^{-1}]$, and so on until attribute a_{k-1}. The last attribute, a_k, picks out all the remaining requests. Namely, requests that have a demand in the range of $(0, 1/(100 \ln m)]$. It is clear that this collection of attributes partition every input instance of the problem in a pairwise-disjoint (and trivially choices-consistent) way. In addition, let $\{\mathsf{BAP}_i\}_{i=1}^{k-1} \cup \{\mathsf{BAP}\}$ be the corresponding family of deterministic algorithms. Specifically, BAP_i is a monotone a_i-restrictive 12-approximation algorithm, and BAP is a monotone a_k-restrictive 3-approximation algorithm. Finally, recall that the integrality gap

of each a_i-restrictive variant of the problem is constant, and the integrality gap of the a_k-restrictive variant is $1 + \epsilon$. Accordingly, and in correspondence with Theorem 4, we achieve the following theorem.

Theorem 8. *There is a monotone deterministic $O(\ln \ln m)$-approximation algorithm for the bandwidth allocation problem in tree networks.*

5 The Multiple Knapsack Problem on Bipartite Graphs

In this section, we study the multiple knapsack problem on bipartite graphs, and design a relatively simple monotone deterministic algorithm that approximately solves the problem within a constant factor of 11.

Narrow sizes. We consider a restricted version of the problem in which the size of every item is *narrow*, that is, $s_i \in (0, 1/2]$, for every $i \in \mathcal{I}$. We begin by presenting a monotone deterministic algorithm, formally described below, that achieves an approximation ratio of 3. Basically, this algorithm is a greedy w.r.t. a non-increasing *profit density* ratio, that is, a value to size ratio. Note that we use the phrase *knapsack j is feasible w.r.t. item i* to designate a knapsack $j \in M_i$ with a residual capacity of at least s_i.

Algorithm 3. $\mathsf{MKP}_{\mathrm{N}}(\mathcal{I})$

Input: An items instance $\mathcal{I}$ in which $s_i \in (0, 1/2]$, for every $i \in \mathcal{I}$
Output: An (item, knapsack) set S of selected items, and their assigned knapsacks

1: **while** $\mathcal{I} \neq \emptyset$ **do**
2: **Remove** the item i that has a maximum profit density from $\mathcal{I}$
3: **Let** j be a feasible knapsack w.r.t. i having a minimal index (∞ if no such knapsack exists)
4: **if** $j \neq \infty$ **then**
5: **Add** (i, j) to $\mathcal{S}$
6: **end if**
7: **end while**
8: **return** S

Theorem 9. *Let $\mathcal{I}$ be an input instance in which $s_i \in (0, 1/2]$, for every $i \in \mathcal{I}$. Algorithm $\mathsf{MKP}_{\mathrm{N}}(\mathcal{I})$ is monotone, deterministic, and outputs feasible solution whose value is at least $1/3$ of an optimal solution's value.*

We now argue that the integrality gap of this variant is at most 3. Essentially, this follows from the insight that the integral solution returned by algorithm $\mathsf{MKP}_{\mathrm{N}}$ is a 3-approximation for the optimal fractional solution, and not only for the optimal integral one.

Wide sizes. We turn to inspect the restricted version of the problem in which the size of every item is guaranteed to be *wide*, i.e., $s_i \in (1/2, 1]$, for every $i \in \mathcal{I}$. We begin by arguing that there is a monotone deterministic algorithm that attains optimal outcome for this variant. The key observation one needs to

make is that no pair of items can be put in the same knapsack simultaneously. This implies that one may disregard the sizes of the items, and presume that all of them are exactly unit. In consequence, this variant is equivalent to the *maximum weighted matching problem on bipartite graph*, which is known to admit a polynomial-time optimal deterministic algorithm (see, e.g., [1]). Referring to this algorithm as $\mathsf{MKP}_{\mathrm{W}}$, we obtain the following theorem. Note that the monotonicity directly results from the optimality of the solution.

Theorem 10. *Let $\mathcal{I}$ be an input instance in which $s_i \in (1/2, 1]$, for every $i \in \mathcal{I}$. Algorithm $\mathsf{MKP}_{\mathrm{W}}(\mathcal{I})$ is monotone, deterministic, and outputs feasible solution whose value is optimal.*

Theorem 11. *The integrality gap of the restricted version of the problem in which the size of every item is guaranteed to be wide is at most* 2*.*

Integration of the results within the framework. We now illustrate how to integrate the results within the confines of the framework. Let a_1 and a_2 be attributes that select all the narrow-sized items and the wide-sized items, respectively. This collection of attributes partition every input instance of the problem in a pairwise-disjoint choices-consistent way. Furthermore, let $\{\mathsf{MKP}_{\mathrm{N}}, \mathsf{MKP}_{\mathrm{W}}\}$ be the corresponding family of deterministic algorithms. Specifically, $\mathsf{MKP}_{\mathrm{N}}$ is a monotone a_1-restrictive 3-approximation algorithm, and $\mathsf{MKP}_{\mathrm{W}}$ is a monotone a_2-restrictive 1-approximation algorithm. Lastly, recollect that the integrality gap of the a_1-restrictive variant of the problem is 3, and the integrality gap of the a_2-restrictive variant is 2. Consequently, and in accordance with Theorem 4, we obtain the following theorem.

Theorem 12. *There is a monotone deterministic* 11*-approximation algorithm for the multiple knapsack problem on bipartite graphs.*

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. In: The weighted bipartite matching problem, ch. 12, Prentice Hall, Englewood Cliffs (1993)
2. Archer, A., Papadimitriou, C.H., Talwar, K., Tardos, É.: An approximate truthful mechanism for combinatorial auctions with single parameter agents. In: 14th SODA, pp. 205–214 (2003)
3. Awerbuch, B., Azar, Y., Fiat, A., Leonardi, S., Rosén, A.: On-line competitive algorithms for call admission in optical networks. Algorithmica 31(1), 29–43 (2001)
4. Azar, Y., Gamzu, I., Gutner, S.: Truthful unsplittable flow for large capacity networks. In: 19th SPAA, pp. 320–329 (2007)
5. Babaioff, M., Lavi, R., Pavlov, E.: Single-value combinatorial auctions and implementation in undominated strategies. In: 17th SODA, pp. 1054–1063 (2006)
6. Bansal, N., Chakrabarti, A., Epstein, A., Schieber, B.: A quasi-ptas for unsplittable flow on line graphs. In: 38th STOC, pp. 721–729 (2006)
7. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. J. ACM 48(5), 1069–1090 (2001)

8. Bar-Yehuda, R., Beder, M., Cohen, Y., Rawitz, D.: Resource allocation in bounded degree trees. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 64–75. Springer, Heidelberg (2006)
9. Briest, P., Krysta, P., Vöcking, B.: Approximation techniques for utilitarian mechanism design. In: 37th STOC, pp. 39–48 (2005)
10. Calinescu, G., Chakrabarti, A., Karloff, H.J., Rabani, Y.: Improved approximation algorithms for resource allocation. In: 9th IPCO, pp. 439–456 (2001)
11. Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. In: 5th APPROX, pp. 51–66 (2002)
12. Chekuri, C., Khanna, S.: A polynomial time approximation scheme for the multiple knapsack problem. SICOMP 35(3), 713–728 (2005)
13. Chekuri, C., Mydlarz, M., Shepherd, F.B.: Multicommodity demand flow in a tree. In: 30th ICALP, pp. 410–425 (2003)
14. Clarke, E.H.: Multipart pricing of public goods. Public Choice 8, 17–33 (1971)
15. Dawande, M., Kalagnanam, J., Keskinocak, P., Salman, F.S., Ravi, R.: Approximation algorithms for the multiple knapsack problem with assignment restrictions. Journal of Combinatorial Optimization 4(2), 171–186 (2000)
16. Feige, U., Vondrák, J.: Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In: 47th FOCS, pp. 667–676 (2006)
17. Fleischer, L., Goemans, M.X., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: 17th SODA, pp. 611–620 (2006)
18. Garg, N., Vazirani, V.V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica 18(1), 3–20 (1997)
19. Groves, T.: Incentives in teams. Econemetrica 41(4), 617–631 (1973)
20. Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. In: 46th FOCS, pp. 595–604 (2005)
21. Lehmann, D.J., O'Callaghan, L., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. J. ACM 49(5), 577–602 (2002)
22. Lewin-Eytan, L., Naor, J., Orda, A.: Admission control in networks with advance reservations. Algorithmica 40(4), 293–304 (2004)
23. Mu'alem, A., Nisan, N.: Truthful approximation mechanisms for restricted combinatorial auctions. In: 18th AAAI, pp. 379–384 (2002)
24. Nisan, N., Ronen, A.: Computationally feasible vcg mechanisms. In: 2nd EC, pp. 242–252 (2000)
25. Nisan, N., Ronen, A.: Algorithmic mechanism design. GEB 35, 166–196 (2001)
26. Nutov, Z., Beniaminy, I., Yuster, R.: A (1-1/e)-approximation algorithm for the generalized assignment problem. ORL 34(3), 283–288 (2006)
27. Phillips, C.A., Uma, R.N., Wein, J.: Off-line admission control for general scheduling problems. In: 11th SODA, pp. 879–888 (2000)
28. Raghavan, P.: Probabilistic construction of deterministic algorithms: Approximating packing integer programs. JCSS 37(2), 130–143 (1988)
29. Raghavan, P., Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica 7(4), 365–374 (1987)
30. Srinivasan, A.: Improved approximation guarantees for packing and covering integer programs. SICOMP 29(2), 648–670 (1999)
31. Tarjan, R.E.: Decomposition by clique separators. DM 55(2), 221–232 (1985)
32. Vickery, W.: Counterspeculation, auctions and competitive sealed tender. Journal of Finance 16, 8–37 (1961)

Upper Bounds on the Noise Threshold for Fault-Tolerant Quantum Computing

Julia Kempe[1], Oded Regev[1], Falk Unger[2], and Ronald de Wolf[2]

[1] Department of Computer Science, Tel-Aviv University, Tel-Aviv, Israel
[2] CWI, Amsterdam, The Netherlands

Abstract. We prove new upper bounds on the tolerable level of noise in a quantum circuit. Our circuits consist of unitary k-qubit gates each of whose input wires is subject to depolarizing noise of strength p, and arbitrary one-qubit gates that are essentially noise-free. We assume the output of the circuit is the result of measuring some designated qubit in the final state. Our main result is that for $p > 1 - \Theta(1/\sqrt{k})$, the output of any such circuit of large enough depth is essentially independent of its input, thereby making the circuit useless. For the important special case of $k = 2$, our bound is $p > 35.7\%$. Moreover, if the only gate on more than one qubit is the CNOT, then our bound becomes 29.3%. These bounds on p are notably better than previous bounds, yet incomparable because of the somewhat different circuit model that we are using. Our main technique is a Pauli basis decomposition, which we believe should lead to further progress in deriving such bounds.

1 Introduction

The field of quantum computing faces two main tasks: to build a large-scale quantum computer, and to figure out what it can do once it exists. In general the first task is best left to (experimental) physicists and engineers, but there is one crucial aspect where theorists play an important role, and that is in analyzing the level of noise that a quantum computer can tolerate before breaking down.

The physical systems in which qubits may be implemented are typically tiny and fragile (electrons, photons and the like). This raises the following paradox: On the one hand we want to isolate these systems from their environment as much as possible, in order to avoid the noise caused by unwanted interaction with the environment—so-called "decoherence". But on the other hand we need to manipulate these qubits very precisely in order to carry out computational operations. A certain level of noise and errors from the environment is therefore unavoidable in any implementation, and in order to be able to compute one would have to use techniques of error correction and fault tolerance.

Unfortunately, the techniques that are used in classical error correction and fault tolerance do not work directly in the quantum case. Moreover, extending these techniques to the quantum world seems at first sight to be nearly impossible due to the continuum of possible quantum states and error patterns. Indeed, when the first important quantum algorithms were discovered [1,2,3,4], many dismissed the whole model of quantum computing as a pipe dream, because it

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 845–856, 2008.

was expected that decoherence would quickly destroy the necessary quantum properties of superposition and entanglement.

It thus came as a great surprise when, in the mid-1990s, *quantum error correcting codes* were developed by Shor and Steane [5,6,7], and these ideas later led to the development of schemes for *fault-tolerant quantum computing* [8,9,10,11,12]. Such schemes take any quantum algorithm designed for an ideal noiseless quantum computer, and turn it into an implementation that is robust against noise, as long as the amount of noise is below a certain threshold, known as the *fault-tolerance threshold*. The overhead introduced by the fault-tolerant schemes is typically modest (a polylogarithmic factor in the running time of the algorithm).

The existence of fault-tolerant schemes turns the problem of building a quantum computer into a hard engineering problem: if we just manage to store our qubits and operate upon them with a level of noise below the fault-tolerance threshold, then we can perform arbitrarily long quantum computations. The actual *value* of the fault-tolerance threshold is far from determined, but will have a crucial influence on the future of the area—the more noise a quantum computer can tolerate in theory, the more likely it is to be realized in practice.[1]

The first fault-tolerant schemes were only able to tolerate noise on the order of 10^{-6}, which is way below the level of accuracy that experimentalists can hope to achieve in the foreseeable future. These initial schemes have been substantially improved in the past decade. In particular, Knill has recently developed various schemes which, according to numerical calculations, seem to be able to tolerate more than 1% noise [13,14]. If we insist on provable constructions, the best known threshold is on the order of 0.1% [15,16,17,18].

Constructions of fault-tolerant schemes provide a *lower bound* on the fault-tolerance threshold. A very interesting question, which is the topic of the current paper, is whether one can prove *upper bounds* on the fault-tolerance threshold. Such bounds give an indication on how far away we are from finding optimal fault-tolerant schemes. They can also give hints as to how one should go about constructing improved fault-tolerant schemes. Such upper bounds are statements of the form "any quantum computation performed with noise level higher than p is essentially useless", where "essentially useless" is some strong indication that interesting quantum computations are impossible in such a model. For instance, Buhrman et al. [19] quantify this by giving a classical simulation of such noisy quantum computation, and Razborov [20] shows that if the computation is too long, the output of the circuit is essentially independent of its input.

The best known upper bounds on the threshold are 50% by Razborov [20] and 45.3% by Buhrman et al. [19]. (These bounds are incomparable because they work in different models; see the end of this section for more details.) As one can see, there are still about two orders of magnitude between our best upper and lower bounds on the fault-tolerance threshold. This leaves experimentalists in the dark as to the level of accuracy they should try to achieve. In this paper, we somewhat reduce this gap. So far, much more work has been spent on lower

[1] The "fault-tolerance threshold" is actually not a universal constant, but rather depends on the details of the circuit model (allowed set of gates, type of noise, etc.).

bounds than on upper bounds. Our approach will be the less-trodden road from above, hoping to bring new techniques to bear on this problem.

Our model. In order to state our results, we need to describe our circuit model. We consider parallel circuits, composed of n *wires* and T *levels* of gates (see Figure 1). We sometimes use the term *time* to refer to one of the $T+1$ "vertical cuts" between the levels. For convenience, we assume that the number of qubits n does not change during the computation. Each level is described by a partition of the qubits, as well as a gate assigned to each set in the partition. Notice that at each level, all qubits must go through some gate (possibly the identity). For each gate the number of input qubits equals the number of output qubits.

We assume the circuit is composed of k-qubit gates that are probabilistic mixtures of unitary operations, as well as arbitrary (i.e., all completely-positive trace-preserving) one-qubit gates. In particular, it is possible to do intermediate one-qubit measurements. We assume the output of the circuit is the outcome of a measurement of a designated output qubit in the computational basis. Finally, we assume that the circuit is subject to noise as follows. Recall that p-depolarizing noise on a certain qubit replaces that qubit by the completely mixed state with probability p, and does not alter the qubit otherwise. Formally, this is described by the superoperator $\mathcal{E}$ acting on a qubit ρ as $\mathcal{E}(\rho) = (1-p)\rho + pI/2$. We assume that each one-qubit gate is followed by at least ε_1-depolarizing noise on its output qubit, where $\varepsilon_1 > 0$ is an arbitrarily small constant. Thus one-qubit gates can be essentially noise-free. We also assume that each k-qubit gate is preceded by at least ε_k-depolarizing noise on each of its input qubits, where $\varepsilon_k > 1 - \sqrt{2^{1/k} - 1} = 1 - \Theta(1/\sqrt{k})$.

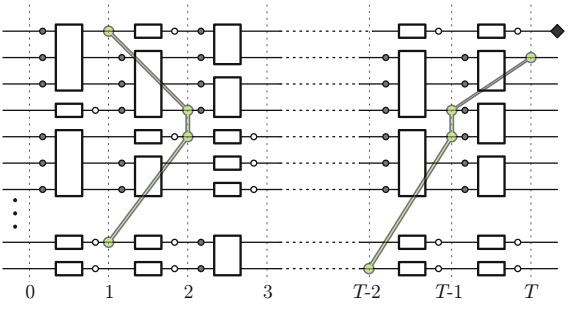


Fig. 1. Parallel circuit with $k = 3$ and T levels. Dark circles are ε_k-depolarizing noise, light circles are ε_1-depolarizing noise. We marked two consistent 4-qubit sets (defined in Section 3). The first has distance 1, the second $T-2$. The upper right qubit is the output.

Our results. In Section 3 we prove our main result:

Theorem 1. *Fix any T-level quantum circuit as above. Then for any two states ρ and τ, the probabilities of obtaining measurement outcome 1 at the output qubit starting from ρ and starting from τ, respectively, differ by at most $2^{-\Omega(T)}$.*

In other words, for any $\eta > 0$, the probability of measuring 1 at the output qubit of a circuit running for $T = O(\log(1/\eta))$ levels is independent of the input (up to $\pm\eta$). This makes the output essentially independent of the starting state, and renders long computations "essentially useless".

Of special interest from an experimental point of view is the case $k = 2$, for which our bound becomes about 35.7%. Furthermore, for the case in which the only allowed two-qubit gate is the controlled-NOT (CNOT) gate, we can improve our bound further to about 29.3%, as we show in the full version of this paper [21]. This case is interesting both theoretically and experimentally. Note also that the CNOT gate together with all one-qubit gates forms a universal set [22]. The same noise-bound applies if we also allow controlled-Y and controlled-Z gates.

Significance of results. First, it is known that fault-tolerant quantum computation is impossible (for any positive noise level) without a source of fresh qubits. Our model takes care of this by allowing arbitrary one-qubit gates—in particular, this includes gates that take any input, and output a fixed one-qubit state, for instance $|0\rangle$. This justifies our assumption that the number of qubits in the circuit remains the same throughout the computation: all qubits can be present from the start, since we can reset them to whatever we want whenever needed.

Second, our assumption that all k-qubit gates are mixtures of unitaries does slightly restrict generality. Not every completely-positive trace-preserving map can be written as a mixture of unitaries.[2] However, we believe that it is still a reasonable assumption. For instance, to the best of our knowledge, all known fault-tolerant constructions can be implemented using such gates (in addition to arbitrary one-qubit gates). Moreover, all known quantum algorithms obtain their speed-up over classical algorithms by using only unitary gates.

Third, we only analyze depolarizing noise acting independently on each qubit. Depolarizing noise is the "most symmetric" type of one-qubit noise and therefore a natural choice for our analysis. Also, it is a relatively weak type of noise: it is not adversarial and does not have correlations between the errors occurring on different qubits. However, since we are proving an *upper bound* on the fault-tolerance threshold, this weakness is actually a good thing, making our result stronger. In principle one can extend our results to various other one-qubit noise models, using an analysis similar to the one developed in Lemma 2. However, not all noise models can actually yield a result like ours. For instance, if we have Toffoli gates with only phaseflip errors, then we can do perfect classical computation. Statements like Theorem 1 are just false in that case.

A more severe restriction is the assumption that the output consists of one qubit. However, we believe that in many instances this is still a reasonable assumption, for instance when the circuit is solving a decision problem. Moreover, our results can easily be extended to the case where the output is obtained by a measurement on a small number of qubits, instead of only one.

By allowing essentially noise-free one-qubit gates, our model addresses the fact that gates on more than one qubit are generally much harder to implement than one-qubit gates. It should also be noted that the exact value of the constant ε_1 is inessential and can be chosen arbitrarily small, as this just affects the constant in the $\Omega(\cdot)$ of Theorem 1. In fact, $\varepsilon_1 > 0$ is only necessary because otherwise

[2] One can implement an arbitrary gate by a unitary gate on the original qubits and additional ancilla qubits in a fixed pure state. However, this increases the arity of the gate, and the ancilla qubits will be affected by the noise before the unitary.

it would be possible to let $\rho := |0\rangle\langle 0| \otimes \rho'$ and $\tau := |1\rangle\langle 1| \otimes \tau'$, do nothing for T levels (i.e., apply noise-free identity gates on all wires) and then measure the first qubit. The resulting difference between output probabilities is 1. Instead of assuming $\varepsilon_1 > 0$ noise, we could alternatively deal with this issue by requiring that every path from the input to the output qubit goes through enough k-qubit gates. Our proof can easily be adapted to this case.

Since our theorem applies to arbitrary starting states, it applies to the case that the initial state is encoded in a good quantum error-correcting code, or is some sort of "magic state" [23,24]. Also in these case, the computation becomes essentially independent of the input after sufficiently many levels.

Finally, it is interesting to note that our bound on the threshold behaves like $1 - \Theta(1/\sqrt{k})$. This matches what is known for classical circuits [25,26], and therefore probably represents the correct asymptotic behavior. Previous bounds only achieved an asymptotic behavior of $1 - \Theta(1/k)$ [20].

Techniques. We believe that a main part of our contribution is introducing a new technique for obtaining upper bounds on the fault-tolerance threshold. Namely, we use a Pauli basis decomposition in order to track the state of the computation. A finer analysis of the Pauli coefficients might improve the bounds we achieve here, and possibly obtain bounds for other computational models.

Related work. The work most closely related to ours is that of Razborov [20]. There, he proves an upper bound of $\varepsilon_k = 1 - 1/k$ on the fault-tolerance threshold. On one hand, his result is stronger than ours as it allows arbitrary k-qubit gates and not just mixtures of unitaries. Razborov also has a second result, namely the trace distance between the two states obtained by applying the circuit to starting states ρ and τ, respectively, is upper bounded by $n2^{-\Omega(T)}$. Hence even the results of arbitrary n-qubit measurement on the full final state become essentially independent of the initial state after $T = O(\log n)$ levels. On the other hand, the value of our bound is better for all values of k, and we also allow essentially noise-free one-qubit gates. Hence the two results are incomparable. Razborov's proof is based on tracking how the trace distance evolves during the computation. Our proof is similar in flavor, but instead of working with the trace distance, we work with the Frobenius distance (since it can easily be expressed in terms of the Pauli decomposition).

Buhrman et al. [19] show that classical circuits can efficiently simulate any quantum circuit that consists of perfect, noise-free *stabilizer operations* (meaning Clifford gates (Hadamard, phase gate, CNOT), preparations of states in the computational basis, and measurements in the computational basis) and arbitrary one-qubit unitary gates that are followed by 45.3% depolarizing noise. Hence such circuits are not significantly more powerful than classical circuits.[3] This

[3] The 45.3%-bound of [19] is in fact *tight* if one additionally allows perfect classical control (i.e., the ability to condition future gates on earlier classical measurement outcomes): circuits with perfect stabilizer operations and arbitrary one-qubits gates suffering from less than 45.3% noise, can simulate perfect quantum circuits. See [27] and [19, Section 5]. These assumptions are not very realistic: in particular, assuming perfect, noise-free CNOTs is a far cry from experimental practice.

result is incomparable to ours: the noise models and the set of allowed gates are different (and we feel ours is more realistic). In particular, in our case noise hits the qubits going into the k-qubit gates but barely affects the one-qubit gates, while in their case the noise only hits the non-Clifford one-qubit unitaries.

Another related result is by Virmani et al. [28]. Instead of depolarizing noise, they consider "dephasing noise". This models phase-errors: rather than replacing a qubit by the completely mixed state with some probability p, dephasing noise applies the Z-gate with probability $p/2$. Virmani et al. [28] show, among other results, that we can efficiently classically simulate any quantum circuit consisting of perfect stabilizer operations, and one-qubit unitary gates that are diagonal in the computational basis and are followed by more than 29.3% dephasing noise. Their result is incomparable to ours for essentially the same reasons as why the Buhrman et al. result is incomparable: a different noise model and a different statement about the resulting power of their noisy quantum circuits.

Finally, it is known that it is impossible to transmit quantum information through a p-depolarizing channel for $p > 1/3$ [29]. As Razborov [20] noticed, this seems to suggest that quantum computation is impossible with depolarizing noise of strength greater than $1/3$, but there is no proof that this is the case.

2 Preliminaries

Let $\mathcal{P} = \{I, X, Y, Z\}$ be the set of one-qubit Pauli matrices,

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \; X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \; Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \; Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

and let $\mathcal{P}_* = \{X, Y, Z\}$. We use $\mathcal{P}^n$ to denote the set of all tensor products of n one-qubit Pauli matrices. For a Pauli matrix $S \in \mathcal{P}^n$ we define its *support*, denoted $\mathrm{supp}(S)$, to be the qubits on which S is not identity. We sometimes use superscripts to indicate the qubits on which certain operators act. Thus $I^{\mathcal{A}}$ denotes the identity operator applied to the qubits in set $\mathcal{A}$.

The set of all $2^n \times 2^n$ Hermitian matrices forms a 4^n-dimensional real vector space. On this space we consider the Hilbert-Schmidt inner product, given by $\langle A, B\rangle := \mathrm{Tr}(A^\dagger B) = \mathrm{Tr}(AB)$. Note that for any $S, S' \in \mathcal{P}^n$, $\mathrm{Tr}(SS') = 2^n$ if $S = S'$ and 0 otherwise, and hence $\mathcal{P}^n$ is an orthogonal basis of this space. It follows that we can uniquely express any Hermitian matrix δ in this basis as

$$\delta = \frac{1}{2^n} \sum_{S \in \mathcal{P}^n} \widehat{\delta}(S) S$$

where $\widehat{\delta}(S) := \mathrm{Tr}(\delta S)$ are the (real) coefficients.

We now state some observations. By the orthogonality of $\mathcal{P}^n$, for any δ,

$$\mathrm{Tr}(\delta^2) = \frac{1}{2^n} \sum_{S \in \mathcal{P}^n} \widehat{\delta}(S)^2.$$

Observation 2 (Unitary preserves sum of squares). *For any unitary matrix U and any Hermitian matrix δ, if we denote $\delta' = U\delta U^\dagger$, then*

$$\sum_{S\in\mathcal{P}^n} \widehat{\delta'}(S)^2 = 2^n \mathrm{Tr}(\delta'^2) = 2^n \mathrm{Tr}(U\delta U^\dagger U\delta U^\dagger) = 2^n \mathrm{Tr}(\delta^2) = \sum_{S\in\mathcal{P}^n} \widehat{\delta}(S)^2.$$

This also shows that conjugating by a unitary matrix, when viewed as a linear operation on the vector of Pauli coefficients, is an orthogonal transformation.

Observation 3 (Tracing out qubits). *Let δ be some Hermitian matrix on a set of qubits W. For $V \subseteq W$, let $\delta_V = \mathrm{Tr}_{W\setminus V}(\delta)$. Then,*

$$\widehat{\delta}(SI^{W\setminus V}) = \mathrm{Tr}(\delta \cdot SI^{W\setminus V}) = \mathrm{Tr}(\delta_V \cdot S) = \widehat{\delta_V}(S).$$

Observation 4 (Noise in the Pauli basis). *Applying a p-depolarizing noise $\mathcal{E}$ to the j-th qubit of Hermitian matrix δ changes the coefficients as follows:*

$$\widehat{\mathcal{E}(\delta)}(S) = \begin{cases} \widehat{\delta}(S) & \text{if } S_j = I \\ (1-p)\widehat{\delta}(S) & \text{if } S_j \neq I \end{cases}$$

In other words, $\mathcal{E}$ "shrinks" by a factor $1-p$ all coefficients that have support on the j-th coordinate.

Observation 5. *Let ρ and τ be two one-qubit states and let $\delta = \rho - \tau$. Consider the two probability distributions obtained by performing a measurement in the computational basis on ρ and τ, respectively. Then the variation distance between these two distributions is $\frac{1}{2}|\widehat{\delta}(Z)|$.*

Proof: Since there are only two possible outcomes for the measurements, the variation distance between the two distributions is exactly the difference in the probabilities of obtaining the outcome 0, which (using $\mathrm{Tr}(\delta) = 0$) is given by

$$|\mathrm{Tr}((\rho-\tau)\cdot|0\rangle\langle 0|)| = \left|\mathrm{Tr}\left(\delta \cdot \frac{I+Z}{2}\right)\right| = \frac{1}{2}|\mathrm{Tr}(\delta\cdot Z)| = \frac{1}{2}|\widehat{\delta}(Z)|. \qquad \blacksquare$$

Our final observation follows immediately from the convexity of the function x^2.

Observation 6 (Convexity). *Let p_i be any probability distribution, and δ_i a set of Hermitian matrices. Let $\delta = \sum_i p_i\delta_i$. Then* $\sum_{S\in\mathcal{P}^n} \widehat{\delta}(S)^2 \leq \sum_i p_i \sum_{S\in\mathcal{P}^n} \widehat{\delta_i}(S)^2$.

3 Proof of Theorem 1

In this section we prove Theorem 1. The idea is the following. Fix two arbitrary initial states ρ and τ. Our goal is to show that after applying the noisy circuit, the state of the output qubit is nearly the same with both starting states. Equivalently, we can define $\delta = \rho - \tau$ and show that after applying the noisy

circuit to δ, the "state" of the output qubit is essentially 0 (the noisy circuit is a linear operation, and hence there is no problem in applying it to δ, which is the difference of two density matrices). In order to show this, we will examine how the coefficients of δ in the Pauli basis develop through the circuit. Initially we might have many large coefficients. Our goal is to show that the coefficients of the output qubit are essentially 0. This is established by analyzing the balance between two opposing forces: noise, which shrinks coefficients by a constant factor (as in Observation 4), and gates, which can increase coefficients. As we saw in Observation 2, unitary gates preserve the sum of squares of coefficients. They can, however, "concentrate" several small coefficients into one large coefficient. One-qubit operations need not preserve the sum of squares (a good example is the gate that resets a qubit to the $|0\rangle$ state), but we can still deal with them by using a known characterization of one-qubit gates. This allows us to bound the amount by which one-qubit gates can increase the Pauli coefficients, and (roughly) shows that the gate that resets a qubit to $|0\rangle$ is "as bad as it gets".

We introduce some terminology. From now on we use the term *qubit* to mean a wire at a specific time, so there are $(T+1)n$ qubits (although during the proof we will also consider qubits that are located between a gate and its associated noise). We say that a set of qubits V is *consistent* if we can meaningfully talk about a "state of the qubits of V" (see Figure 1). More formally, we define a consistent set as follows. The set of all qubits at time 0 and all its subsets are consistent. If V is some consistent set of qubits, which contains all input qubits IN of some gate (possibly a one-qubit identity gate), then also $(V \setminus IN) \cup OUT$ and all its subsets are consistent, where OUT denotes the gate's output qubits. Note that here we think of the noise as being part of the gate. For a consistent set V and a state (or more generally, a Hermitian matrix) ρ, we denote the state of V when the circuit is applied with the initial state ρ, by ρ_V. In other words, ρ_V is the state one obtains by applying some initial part of the circuit to ρ, and then tracing out from the resulting state all qubits that are not in V.

If v is a qubit, we use $\mathrm{dist}(v)$ to denote its distance from the input, i.e., the level of the gate just preceding it. The qubits of the starting state have $\mathrm{dist}(v) = 0$. For a nonempty set V of qubits we define $\mathrm{dist}(V) = \min\{\mathrm{dist}(v) \mid v \in V\}$, and extend it to the empty set by $\mathrm{dist}(\emptyset) = \infty$. Note that $\mathrm{dist}(V)$ does not increase if we add qubits to V. In the rest of this section we prove the following lemma, showing that a certain invariant holds for all consistent sets V.

Lemma 1. *For all $\varepsilon_1 > 0$ and $\varepsilon_k > 1 - \sqrt{2^{1/k} - 1}$ there is a $\theta < 1$ such that the following holds. For any T-level circuit in our model, initial states ρ and τ, $\delta = \rho - \tau$, and any consistent V, we have $\mathrm{Tr}(\delta_V^2) \le 2 \cdot \theta^{\mathrm{dist}(V)}$. Equivalently:*

$$\sum_{S \in \mathcal{P}^V} \widehat{\delta_V}(S)^2 \le 2 \cdot 2^{|V|} \cdot \theta^{\mathrm{dist}(V)}. \tag{1}$$

If we consider the consistent set V containing the output qubit at time T, then we get that $\widehat{\delta_V}(Z)^2 \le 4\theta^T$. By Observation 5, this implies Theorem 1.

3.1 Proof of Lemma 1

The proof of the invariant is by induction on the sets V. At the base are all sets V contained entirely within time 0. All other sets are handled in the induction step. To justify the inductive proof, we need an ordering on the consistent sets V such that for each V, the proof for V uses the inductive hypothesis only on sets V' that appear before V. As will become apparent from the proof, if we denote by latest(V) the maximum time at which V contains a qubit, then each V' for which we use the induction hypothesis has strictly less qubits than V at time latest(V). Therefore, we can order the sets V first in increasing order of latest(V) and then in increasing order of the number of qubits at time latest(V).

Base case. Here V is fully contained within time 0. If $V = \emptyset$ then both sides of the invariant are zero, so from now on assume V is nonempty. In this case $\mathrm{dist}(V) = 0$. The matrix δ_V is the difference of two density matrices ρ_V and τ_V. Hence $\mathrm{Tr}(\delta_V^2) = \mathrm{Tr}(\rho_V^2) + \mathrm{Tr}(\tau_V^2) - 2\mathrm{Tr}(\rho_V \tau_V) \leq 2$, and the invariant is satisfied.

Induction step. Let V'' be any consistent set containing at least one qubit at time greater than zero. Our goal in this section is to prove the invariant for V''. Consider any of the qubits of V'' located at time latest(V'') and let G be the gate that has this qubit as one of its output qubits. We now consider two cases, depending on whether G is a k-qubit gate or a one-qubit gate.

Case 1: G is a k-qubit gate. Here G is a probabilistic mixture of k-qubit unitaries. First, by Observation 6 it suffices to prove the invariant for k-qubit unitaries. So assume G is a k-qubit unitary acting on the qubits $\mathcal{A} = \{A_1, \ldots, A_k\}$. Let $\mathcal{A}' = \{A'_1, \ldots, A'_k\}$ be the qubits after the ε_k-noise but before the gate G and $\mathcal{A}'' = \{A''_1, \ldots, A''_k\}$ the qubits after G (see Figure 2). By our choice of G, $\mathcal{A}'' \cap V'' \neq \emptyset$. Define $V' = (V'' \setminus \mathcal{A}'') \cup \mathcal{A}'$ and $V = (V'' \setminus \mathcal{A}'') \cup \mathcal{A}$. Note that V and its subsets are consistent sets with strictly fewer qubits than V'' at time latest(V''), hence we can apply the induction hypothesis to them. Our goal is to prove the invariant Eq. (1) for V''. First, by Observation 3,

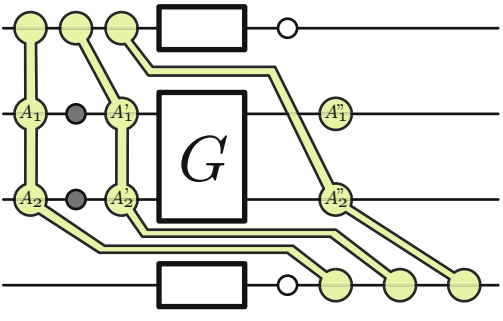


Fig. 2. An example showing the sets V, V', and V'' for a two-qubit gate G

$$\sum_{S \in \mathcal{P}^{V''}} \widehat{\delta_{V''}}(S)^2 \leq \sum_{S \in \mathcal{P}^{V'' \cup \mathcal{A}''}} \widehat{\delta_{V'' \cup \mathcal{A}''}}(S)^2. \tag{2}$$

Because G (which maps $\delta_{V'}$ to $\delta_{V'' \cup \mathcal{A}''}$) is unitary, it preserves the sum of squares of $\widehat{\delta}$-coefficients (see Observation 2), so the right hand side of (2) is equal to

$$\sum_{S \in \mathcal{P}^{V'}} \widehat{\delta_{V'}}(S)^2 = \sum_{S \in \mathcal{P}^{V' \setminus \mathcal{A}'}} \sum_{R \in \mathcal{P}^{\mathcal{A}'}} \widehat{\delta_{V'}}(RS)^2.$$

Since the only difference between δ_V and $\delta_{V'}$ is noise on the qubits $A_1, \ldots, A_k$, using Observation 4 and denoting $\mu = 1 - \varepsilon_k$, we get that the above is at most

$$\sum_{S \in \mathcal{P}^{V \setminus \mathcal{A}}} \sum_{R \in \mathcal{P}^{\mathcal{A}}} \mu^{2|\mathrm{supp}(R)|} \widehat{\delta_V}(RS)^2$$
$$= \sum_{S \in \mathcal{P}^{V \setminus \mathcal{A}}} \sum_{a \subseteq \mathcal{A}} \mu^{2|a|}(1-\mu^2)^{k-|a|} \sum_{R \in \mathcal{P}^a \otimes I^{\mathcal{A} \setminus a}} \widehat{\delta_V}(RS)^2,$$

where the equality is because for any fixed S and any $R \in \mathcal{P}^{\mathcal{A}}$, the term $\widehat{\delta_V}(RS)^2$, which appears with coefficient $\mu^{2|\mathrm{supp}(R)|}$ on the left, appears with the same coefficient $\sum_{a \supseteq \mathrm{supp}(R)} \mu^{2|a|}(1-\mu^2)^{k-|a|} = \mu^{2|\mathrm{supp}(R)|}$ on the right. By rearranging and using Observation 3 we get that the above is equal to

$$\sum_{a \subseteq \mathcal{A}} \mu^{2|a|}(1-\mu^2)^{k-|a|} \sum_{S \in \mathcal{P}^{(V \setminus \mathcal{A}) \cup a}} \widehat{\delta_{(V \setminus \mathcal{A}) \cup a}}(S)^2$$
$$\leq \sum_{a \subseteq \mathcal{A}} \mu^{2|a|}(1-\mu^2)^{k-|a|} 2 \cdot 2^{|(V \setminus \mathcal{A}) \cup a|} \cdot \theta^{\mathrm{dist}((V \setminus \mathcal{A}) \cup a)}$$

where we used the inductive hypothesis. Note that $\mathrm{dist}((V \setminus \mathcal{A}) \cup a) \geq \mathrm{dist}(V)$, so the above is

$$\leq 2 \cdot 2^{|V \setminus \mathcal{A}|} \cdot \theta^{\mathrm{dist}(V)} \sum_{a \subseteq \mathcal{A}} 2^{|a|} \mu^{2|a|}(1-\mu^2)^{k-|a|}$$
$$= 2 \cdot 2^{|V \setminus \mathcal{A}|} \cdot \theta^{\mathrm{dist}(V)}((1-\mu^2) + 2\mu^2)^k = 2 \cdot 2^{|V \setminus \mathcal{A}|} \cdot \theta^{\mathrm{dist}(V)}(1+\mu^2)^k. \quad (3)$$

Note that $|V \setminus \mathcal{A}| \leq |V''| - 1$ and $\mathrm{dist}(V'') - 1 \leq \mathrm{dist}(V)$, so the right hand side is bounded by

$$\leq 2 \cdot 2^{|V''|-1} \cdot \theta^{\mathrm{dist}(V'')-1}(1+\mu^2)^k.$$

Since $\varepsilon_k > 1 - \sqrt{2^{1/k} - 1}$, we have that $(1+\mu^2)^k \leq 2\theta$ if θ is close enough to 1, so we can finally bound the last expression to prove the invariant for V''

$$\leq 2 \cdot 2^{|V''|} \cdot \theta^{\mathrm{dist}(V'')}.$$

Case 2: G is a one-qubit gate. Before proving the invariant, we need to prove the following property of completely-positive trace-preserving (CPTP) maps on one qubit. The proof appears in the full version of this paper [21].

Lemma 2. *For any CPTP map G on one qubit there exists a $\beta \in [0,1]$ such that the following holds. For any Hermitian matrix δ, if we let δ' denote the result of applying G to δ, then we have*

$$\widehat{\delta'}(X)^2 + \widehat{\delta'}(Y)^2 + \widehat{\delta'}(Z)^2 \leq (1-\beta) \cdot \widehat{\delta}(I)^2 + \beta \cdot (\widehat{\delta}(X)^2 + \widehat{\delta}(Y)^2 + \widehat{\delta}(Z)^2).$$

Let A be the qubit G is acting on, and recall that our goal is to prove the invariant for the set V''. Denote by A' the qubit of G after the gate but before the ε_1 noise, and by A'' the qubit after the noise. As before, by our choice of G, we

have $A'' \in V''$. Let $\mathcal{A} = \{A\}$, $\mathcal{A}' = \{A'\}$, $\mathcal{A}'' = \{A''\}$. Define $V' = (V'' \setminus \mathcal{A}'') \cup \mathcal{A}'$ and $V = (V'' \setminus \mathcal{A}'') \cup \mathcal{A}$ and notice that $|V| = |V'| = |V''|$. By using Lemma 2, we obtain a $\beta \in [0,1]$ such that

$$\sum_{S \in \mathcal{P}^{V''}} \widehat{\delta_{V''}}(S)^2 \leq \sum_{S \in \mathcal{P}^{V' \setminus \mathcal{A}'}} \left(\widehat{\delta_{V'}}(IS)^2 + (1-\varepsilon_1)^2 \sum_{R \in \mathcal{P}_*^{\mathcal{A}'}} \widehat{\delta_{V'}}(RS)^2 \right)$$
$$\leq \sum_{S \in \mathcal{P}^{V \setminus \mathcal{A}}} \left((1+(1-\varepsilon_1)^2(1-2\beta))\widehat{\delta_V}(IS)^2 + (1-\varepsilon_1)^2\beta \sum_{R \in \mathcal{P}^{\mathcal{A}}} \widehat{\delta_V}(RS)^2 \right).$$

By applying the induction hypothesis to both $V \setminus \mathcal{A}$ and V, we can upper bound the above by

$$(1 + (1-\varepsilon_1)^2(1-2\beta)) \cdot 2 \cdot 2^{|V|-1} \cdot \theta^{\mathrm{dist}(V \setminus \mathcal{A})} + (1-\varepsilon_1)^2 \beta \cdot 2 \cdot 2^{|V|} \cdot \theta^{\mathrm{dist}(V)}$$
$$\leq \frac{1 + (1-\varepsilon_1)^2}{2\theta} \cdot 2 \cdot 2^{|V''|} \cdot \theta^{\mathrm{dist}(V'')}$$

where we used that $|V| = |V''|$, and $\mathrm{dist}(V'') - 1 \leq \mathrm{dist}(V) \leq \mathrm{dist}(V \setminus \mathcal{A})$. Hence the invariant remains valid if we choose $\theta < 1$ such that $1 + (1-\varepsilon_1)^2 \leq 2\theta$.

Acknowledgment. We thank Mary Beth Ruskai for a pointer to [30] and for sharing her insights on one-qubit operations; Peter Shor for a discussion on entanglement-breaking channels which is related to the discussion of [29] at the end of Section 1; and an anonymous ICALP referee for helpful comments.

All authors acknowledge support by the European Commission under the Integrated Project Qubit Applications (QAP) funded by the IST directorate as Contract Number 015848. JK is supported by an Alon Fellowship and by the Israeli Science Foundation, OR by the Binational Science Foundation and by the Israel Science Foundation, and RdW is partially supported by a Veni grant from the Netherlands Organization for Scientific Research (NWO).

References

1. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM Journal on Computing 26(5), 1411–1473 (1997); Earlier version in STOC 1993
2. Simon, D.: On the power of quantum computation. SIAM Journal on Computing 26(5), 1474–1483 (1997); Earlier version in FOCS 1994
3. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing 26(5), 1484–1509 (1997); Earlier version in FOCS 1994
4. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of 28th ACM STOC, pp. 212–219 (1996)
5. Shor, P.W.: Scheme for reducing decoherence in quantum memory. Physical Review A 52, 2493 (1995)
6. Shor, P.W.: Fault-tolerant quantum computation. In: 37th FOCS, pp. 56–65 (1996)
7. Steane, A.: Multiple particle interference and quantum error correction. Proceedings of the Royal Society of London 452, 2551–2577 (1996)

8. Knill, M., Laflamme, R., Zurek, W.: Accuracy threshold for quantum computation (October 15, 1996) quant-ph/9610011
9. Knill, E., Laflamme, R., Zurek, W.H.: Resilient quantum computation. Science 279(5349), 342–345 (1998)
10. Aharonov, D., Ben-Or, M.: Fault tolerant quantum computation with constant error. In: Proceedings of 29th ACM STOC, pp. 176–188 (1997)
11. Kitaev, A.Y.: Quantum computations: Algorithms and error correction. Russian Mathematical Surveys 52(6), 1191–1249 (1997)
12. Gottesman, D.: Stabilizer Codes and Quantum Error Correction. PhD thesis, Caltech (1997) quant-ph/9702052
13. Knill, M.: Quantum computing with realistically noisy devices. Nature 434, 39–44 (2005)
14. Knill, M.: Fault-tolerant postselected quantum computation: Threshold analysis (April 19, 2004) quant-ph/0404104
15. Aliferis, P., Gottesman, D., Preskill, J.: Accuracy threshold for postselected quantum computation. Quantum Information and Computation 8(3), 181–244 (2008)
16. Aliferis, P.: Threshold lower bounds for Knill's Fibonacci scheme (September 22, 2007) quant-ph/0709.3603
17. Aliferis, P.: Level Reduction and the Quantum Threshold Theorem. PhD thesis, Caltech (2007) quant-ph/0703264
18. Reichardt, B.: Error-Detection-Based Quantum Fault Tolerance Against Discrete Pauli Noise. PhD thesis, UC Berkeley (2006) quant-ph/0612004
19. Buhrman, H., Cleve, R., Laurent, M., Linden, N., Schrijver, A., Unger, F.: New limits on fault-tolerant quantum computation. In: 47th FOCS, pp. 411–419 (2006)
20. Razborov, A.: An upper bound on the threshold quantum decoherence rate. Quantum Information and Computation 4(3), 222–228 (2004)
21. Kempe, J., Regev, O., Unger, F., de Wolf, R.: Upper bounds on the noise threshold for fault-tolerant quantum computing (2008) quant-ph/0802.1464
22. Barenco, A., Bennett, C., Cleve, R., DiVincenzo, D., Margolus, N., Shor, P., Sleator, T., Smolin, J., Weinfurter, H.: Elementary gates for quantum computation. Physical Review A 52, 3457–3467 (1995)
23. Bravyi, S., Kitaev, A.: Universal quantum computation with ideal Clifford gates and noisy ancillas. Physical Review A 71 (022316) (2005)
24. Reichardt, B.: Quantum universality from Magic States Distillation applied to CSS codes. Quantum Information Processing 4, 251–264 (2005)
25. Evans, W.S., Schulman, L.J.: Signal propagation and noisy circuits. IEEE Trans. Inform. Theory 45(7), 2367–2373 (1999)
26. Evans, W.S., Schulman, L.J.: On the maximum tolerable noise of k-input gates for reliable computation by formulas. IEEE Trans. Inform. Theory 49(11), 3094–3098 (2003)
27. Reichardt, B.: Quantum universality by distilling certain one- and two-qubit states with stabilizer operations (2006) quant-ph/0608085
28. Virmani, S., Huelga, S., Plenio, M.: Classical simulability, entanglement breaking, and quantum computation thresholds. Physical Review A 71 (042328) (2005)
29. Bruss, D., DiVincenzo, D., Ekert, A., Fuchs, C., Macchiavello, C., Smolin, J.: Optimal universal and state-dependent quantum cloning. Physical Review A 43, 2368–2378 (1998)
30. Ruskai, M.B., Szarek, S., Werner, E.: An analysis of completely-positive trace-preserving maps on $\mathcal{M}_2$. Linear Algebra and its Applications 347, 159–187 (2002)

Finding Optimal Flows Efficiently

Mehdi Mhalla[1] and Simon Perdrix[2]

[1] LIG, University of Grenoble, France
mehdi.mhalla@imag.fr
[2] Oxford University Computing Laboratory, UK
simon.perdrix@comlab.ox.ac.uk

Abstract. Among the models of quantum computation, the One-way Quantum Computer [11,12] is one of the most promising proposals of physical realisation [13], and opens new perspectives for parallelisation by taking advantage of quantum entanglement [2]. Since a One-way QC is based on quantum measurement, which is a fundamentally nondeterministic evolution, a sufficient condition of global determinism has been introduced in [4] as the existence of a *causal flow* in a graph that underlies the computation. A $O(n^3)$-algorithm has been introduced [6] for finding such a causal flow when the numbers of output and input vertices in the graph are equal, otherwise no polynomial time algorithm was known for deciding whether a graph has a causal flow or not. Our main contribution is to introduce a $O(m)$-algorithm for finding a causal flow (where m is the number of edges of the graph), if any, whatever the numbers of input and output vertices are. This answers the open question stated by Danos and Kashefi [4] and by de Beaudrap [6]. Moreover, we prove that our algorithm produces a flow of minimal depth.

Whereas the existence of a causal flow is a sufficient condition for determinism, it is not a necessary condition. A weaker version of the causal flow, called *gflow* (generalised flow) has been introduced in [3] and has been proved to be a necessary and sufficient condition for a family of deterministic computations. Moreover the depth of the quantum computation is upper bounded by the depth of the gflow. However the existence of a polynomial time algorithm that finds a gflow has been stated as an open question in [3]. In this paper we answer this positively with a polynomial time algorithm that outputs an optimal gflow of a given graph and thus finds an optimal correction strategy to the nondeterministic evolution due to measurements.

1 Introduction

A one-way quantum computation [11] consists in performing a sequence of one-qubit measurements on an initial entangled quantum state. This initial state, described by a graph, is a graph state [8], where some vertices correspond to the input qubits of the computation, others to the output qubits and the rest of the vertices correspond to auxiliary qubits measured during the computation. Since quantum measurements are nondeterministic, a one-way quantum computation requires corrections, making the basis of some measurements dependent

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 857–868, 2008.

on the classical outcome of some other measurements. This corrections induce a dependency between the measurements, affecting the depth of the computation.

Moreover, in order to be implemented, such corrections impose the structure of the graph states that can be used for deterministic computation. Indeed, for some graph states, not all the sequences of one-qubit measurements represent a unitary embedding. The measurement calculus [5] is a formal framework for one-way quantum computations, where the dependencies between measurements and corrections are precisely identified. Using this formalism, Danos and Kashefi in [4] have proved that any one-way quantum computation translated from a quantum circuit is such that its underlying graph satisfies a causal flow condition (see section 2.) As a consequence, the existence of a causal flow is a sufficient condition for determinism.

In [7] a polynomial time algorithm in the size of the graph has been proposed for finding a causal flow when the numbers of outputs and inputs are equal, whereas the existence of a polynomial time algorithm in the general case, has been stated as an open question. We propose in this paper a faster and more general algorithm for finding a causal flow, whatever the numbers of inputs and outputs are.

It turns out that the existence of a causal flow is not a necessary condition for determinism. Indeed, a weaker flow condition, the gflow condition has been introduced as a necessary and sufficient condition for a class of uniformly deterministic computations (see [3] for a formal definition.) Here, we introduce a polynomial time algorithm for finding a gflow and thus checking whether a graph allows a uniformly deterministic computation, which gives substantially more relevance to the notion of gflow introduced in [3].

We also prove that the algorithms introduced in this paper produce minimal depth flows. This implies that the gflow algorithm gives a lower bound on the complexity of a correction strategy in a measurement-based setting for quantum computation.

2 Definitions

A graph with input and output vertices is called an open graph, and is defined as follows:

Definition 1 (Open Graph). *An open graph is a triplet (G, I, O), where $G = (V, E)$ is a undirected graph, and $I, O \subseteq V$ are respectively called input and output vertices.*

During a one-way quantum computation all non output qubits (represented as non output vertices in the corresponding open graph) are measured. Since quantum measurements are nondeterministic, for each qubit measurement, a correction consists in acting on some unmeasured non input qubits, depending on the classical outcome of the measurement, in order to make the computation deterministic. Thus, a correction strategy induces a sequential dependency between measurements. As a consequence, the depth of the quantum computation depends on the correction strategy.

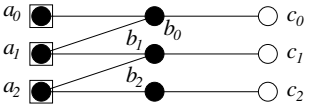


Fig. 1. Example of open graph – squared vertices represent inputs, white vertices represent outputs – which has a causal flow $(g, \prec)$, where $g(a_i) = b_i$, $g(b_i) = c_i$ and $a_0 \prec a_1 \prec a_2 \prec \{b_0, b_1, b_2\} \prec \{c_0, c_1, c_2\}$

Correction strategies are characterised by flows in open graphs. A flow $(g, \prec)$ consists in a partial order $\prec$ over the vertices ($i \prec j$ if i is measured before j) and a function g that associates with each vertex the vertices used for correcting its measurement (all non output qubits are measured.) Input qubits cannot be used for correction (see [5].)

Given an open graph, two kinds of flows are considered: the causal flow and the gflow (generalised flow.) The former has been introduced by Danos and Kashefi [4] and corresponds to the computation strategy that consists in correcting each qubit measurement by acting on a single neighbour of the measured qubit. For a given open graph, a causal flow is characterised by a function g which associates with each non output vertex a non input vertex used for the correction of its measurement. More formally:

Definition 2 (causal flow). *$(g, \prec)$ is a causal flow of (G, I, O), where $g : V(G) \setminus O \to V(G) \setminus I$ and $\prec$ is a strict partial order over $V(G)$, if and only if*

1. *$i \prec g(i)$*
2. *if $j \in N(g(i))$ then $j = i$ or $i \prec j$, where $N(v)$ is the neighbourhood of v*
3. *$i \in N(g(i))$.*

An example of causal flow is given in Figure 1. Notice that if the numbers of input and output vertices are the same, a causal flow can be reduced to a path cover and then to a standard network flow [6]. This reduction was used to define an $O(n^3)$-algorithm for finding a causal flow in the case where the cardinalities of input and output qubits are the same [6].

The second type of flow considered, the generalised flow, *gflow*, has been introduced in [3] and corresponds to a more general correction strategy that associates with each non output vertex a set of vertices used for the corresponding correction (instead of a single vertex.) This generalisation not only leads to a reduction of the computational depth, but also provides a correction strategy to some open graphs which have no causal flow. Moreover, notice that the existence of a gflow is a necessary and sufficient condition for uniform, strong and stepwise deterministic computations [3].

For a given open graph, a gflow $(g, \prec)$ is characterised by a function g which associates with each non output vertex, a set of non input vertices used for its correction, and a strict partial order $\prec$:

Definition 3 (gflow). *$(g, \prec)$ is a gflow of (G, I, O), where $g : V(G) \setminus O \to \wp(V(G) \setminus I)$ and $\prec$ is a strict partial order over $V(G)$, if and only if*

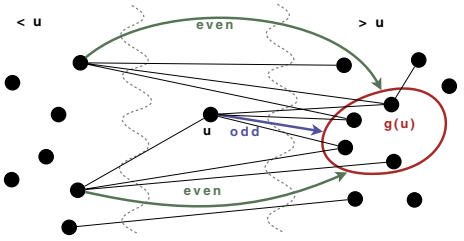


Fig. 2. Graphical interpretation of a gflow $(g, \prec)$: for a given vertex u all the vertices larger than u represent qubits that will be measured after the qubit u. The set $g(u)$ has to be composed of qubits measured after u, and such that the following parity conditions are satisfied: there is an odd number of edges between $g(u)$ and u and there is an even number of edges between $g(u)$ and any vertex which is not larger u.

1. *if* $j \in g(i)$ *then* $i \prec j$
2. *if* $j \in Odd(g(i))$ *then* $j = i$ *or* $i \prec j$
3. $i \in Odd(g(i))$

Where $Odd(K) = \{u\,, |N(u) \cap K| = 1 \mod 2\}$ *is the odd neighbourhood of* K*, i.e. the set of vertices which have an odd number of neighbours in* K*.*

A graphical interpretation of the generalised flow is given in Figure 2.

A flow $(g, \prec)$ of (G, I, O) induces a partition of the vertices of the open graph:

Definition 4. *For a given open graph* (G, I, O) *and a given flow* $(g, \prec)$ *of* (G, I, O)*, let*

$$V_k^{\prec} = \begin{cases} \mathsf{max}_{\prec}(V(G)) & \text{if } k = 0 \\ \mathsf{max}_{\prec}(V(G) \setminus (\cup_{i<k} V_i^{\prec})) & \text{if } k > 0 \end{cases}$$

where $\mathsf{max}_{\prec}(X) = \{u \in X \text{ s.t. } \forall v \in X, \neg(u \prec v)\}$ *is the set of the maximal elements of* X*. The* depth $d^{\prec}$ *of the flow is the smallest* d *such that* $V_{d+1}^{\prec} = \emptyset$*.* $(V_k^{\prec})_{k=0\ldots d^{\prec}}$ *is a partition of* $V(G)$ *into* $d^{\prec} + 1$ *layers.*

A causal flow or a gflow $(g, \prec)$ of (G, I, O) leads to a correction strategy for the corresponding one-way quantum computation, which consists in measuring the non output qubits of each layer in parallel, from the layer $V_{d^{\prec}}^{\prec}$ to the layer $V_1^{\prec}$. After the measurement of a layer $V_k^{\prec}$, with $k > 0$, corrections are realised according to the function g by acting on qubits in $\cup_{i<k} V_i^{\prec}$ (see [3] for details.) The depth of such a one-way quantum computation is $d^{\prec}$.

Definition 5. *For a given open graph* (G, I, O) *and two given causal flows (resp. gflows)* $(g, \prec)$ *and* $(g', \prec')$ *of* (G, I, O)*,* $(g, \prec)$ *is more delayed than* $(g', \prec')$ *if* $\forall k$*,* $|\cup_{i=0\ldots k} V_k^{\prec}| \geq |\cup_{i=0\ldots k} V_k^{\prec'}|$ *and there exists a* k *s.t. the inequality is strict. A causal flow (resp. gflow)* $(g, \prec)$ *is* maximally delayed *if there exists no causal flow (resp. gflow) of the same open graph that is more delayed.*

For instance, the flow $(g, \prec)$ described in Figure 1 is a maximally delayed causal flow. However, $(g, \prec)$ is not a maximally delayed gflow since $(g', \prec')$ is a more

delayed gflow, where $g'(a_0) = \{b_0, b_1, b_2\}, g'(a_1) = \{b_1, b_2\}, g'(a_2) = \{b_2\}$, $g'(b_0) = \{c_0\}, g'(b_1) = \{c_1\}, g'(b_2) = \{b_2\}$, and $\{a_0, a_1, a_2\} \prec' \{b_0, b_1, b_2\} \prec' \{c_0, c_1, c_2\}$. One can prove that $(g', \prec')$ is a maximally delayed gflow.

The following two lemmas are proved for both kinds of flows.

Lemma 1. *If $(g, \prec)$ is a maximally delayed causal flow (resp. gflow) of (G, I, O) then $V_0^{\prec} = O$.*

Proof. Let $(g, \prec)$ be a maximally delayed causal flow (resp. gflow) of (G, I, O). Elements of $V_0^{\prec}$ have no image under g because of condition 1 in both definitions thus $V_0^{\prec} \subseteq O$. Moreover, by contradiction, if $O \setminus V_0^{\prec} \neq \emptyset$, let $\prec' = \prec \setminus (O \setminus V_0^{\prec}) \times V(G)$. $(g, \prec')$ is a causal flow (resp. gflow) of (G, I, O): condition 1 of both definition is satisfied by $\prec'$, because the domain of g does not intersect O, so for any i in the domain of g, $i \prec' j$ iff $i \prec j$; conditions 2 and 3 of both definitions are satisfied in a same way. Thus, $(g, \prec')$ is a causal flow (resp. gflow) of (G, I, O). Moreover, for any k, $\cup_{i=0\ldots k} V_k^{\prec} \subseteq \cup_{i=0\ldots k} V_k^{\prec'}$, and $|V_0^{\prec}| < |V_0^{\prec'}|$ thus $(g, \prec')$ is more delayed than $(g, \prec)$ which leads to a contradiction. □

Lemma 2. *If $(g, \prec)$ is a maximally delayed causal flow (resp. gflow) of (G, I, O) then $(\tilde{g}, \tilde{\prec})$ is a maximally delayed causal flow (resp. gflow) of $(G, I, O \cup V_1^{\prec})$ where $\tilde{g}$ is the restriction of g to $V(G) \setminus (V_0^{\prec} \cup V_1^{\prec})$ and $\tilde{\prec} = \prec \setminus V_1^{\prec} \times V_0^{\prec}$.*

Proof. First, one can prove that $(\tilde{g}, \tilde{\prec})$ is a causal flow (resp. gflow) of $(G, I, O \cup V_1^{\prec})$. Moreover, by contradiction, if there exists a causal flow (resp. gflow) $(g', \prec')$ that is more delayed than $(\tilde{g}, \tilde{\prec})$ then it could be extended to $(g'', \prec'')$ where $g''(u) = g'(u)$ if $u \in V \setminus (V_0^{\prec} \cup V_1^{\prec})$, $g''(u) = g(u)$ if $u \in V_1^{\prec}$ and $\prec'' = \prec' \cup \{(u, v), u \in V_1^{\prec} \wedge u \prec v\}$. $(g'', \prec'')$ is then a more delayed causal flow (resp. gflow) of (G, I, O) than $(g, \prec)$, which leads to a contradiction. □

Lemma 3. *If $(g, \prec)$ is a maximally delayed gflow, then $V_1^{\prec} = \{u \in V \setminus O,$ $\exists K \subseteq O,\ Odd(K) \cap (V \setminus O) = \{u\}\}$.*

Proof. First, notice that if $(g, \prec)$ is a maximally delayed gflow, then for any $u \in V_1^{\prec}$, $g(u) \subseteq O$ since $u \prec v$ if $v \in g(u)$ (condition 1 of definition 3.) Furthermore, by definition of $V_1^{\prec}$, if $u \prec v$ then $v \in O$ thus conditions 2 and 3 of definition 3 imply that $Odd(g(u)) \cap (V \setminus O) = \{u\}$.

To prove that any $u \in V \setminus O$ such that $\exists K \subseteq O, Odd(K) \cap V \setminus O = \{u\}, u \in V_1^{\prec}$, we proceed by contradiction. We prove that delaying the measurement of a vertex not in $V_1^{\prec}$ satisfying the condition permits to create a more delayed gflow. Indeed, let $(g, \prec)$ be a maximally delayed flow of (G, I, O) and let $u_1 \in V \setminus V_0^{\prec}$ be such that $\exists K \subseteq O, Odd(K) \cap V \setminus O = \{u_1\}$. Let $g'(u) = K$ if $u = u_1$ and $g'(u) = g(u)$ otherwise. Let $\prec'$ be the strict partial order defined by $u \prec' v$ if $u \neq u_1$ and $u \prec v$ or if $u = u_1$ and $v \in K$. It leads to a contradiction since $(g', \prec')$ is a more delayed gflow of (G, I, O) than $(g, \prec)$. □

In a similar way, one can prove that:

Lemma 4. *If $(g, \prec)$ is a maximally delayed causal flow, then $V_1^{\prec} = \{u \in V \setminus O,$ $\exists v \in O,\ N(v) \cap V \setminus O = \{u\}\}$.*

Lemmas 3 and 4 show that in a maximally delayed flow, all the elements that can be corrected at the last step are in the maximal layer of $V \setminus O$ (i.e. in $V_1^{\prec}$.) Combined with the recursive structure of maximally delayed flow (lemma 2), this shows that the layers $V_k^{\prec}$ of a maximally delayed flow can be iteratively constructed by finding elements that can be corrected starting from the output qubits. This gives rise to the polynomial time algorithms of the next sections.

3 Causal Flow Algorithm

The relation between causal flow and determinism is presented in [4]. The best known algorithm for finding a causal flow has been proposed in [6], and works only if the numbers of inputs and outputs are the same. The complexity of the algorithm is in $O(nm)$ where n is the number of vertices and m the number of edges (more precisely $O(km)$ where k is the number of inputs (outputs) [7].) We present here a more general and faster algorithm.

Theorem 1. *For a given open graph (G, I, O), finding a causal flow can be done in $O(m)$ operations where $m = |E(G)|$ is the number of edges of the graph G.*

In order to prove Theorem 1, we introduce the algorithm 1 which decides whether a given open graph has a causal flow, and outputs a maximally delayed causal flow if one exists. This recursive algorithm is based on the recursive structure, pointed out in the previous section, of the maximally delayed causal flows. The algorithm recursively finds the layers $(V_k^{\prec})_{k=0\ldots d^{\prec}}$: at the k^{th} call to `Flowaux`, the algorithm finds the set $V_k^{\prec} = Out'$ (see algorithm 1 and figure 3.) To improve the complexity of the algorithm, a set C of potential correctors is maintained, we also maintain the number of non output neighbours of every output vertex that is not an input. The partial order $\prec$ of the flow produced by the algorithm is defined via a labeling l which associates with each vertex, the index of its layer. As a consequence, for any two vertices u and v, $u \prec v$ iff $l(u) > l(v)$.

Proof of Theorem 1: The precondition for the call of `Flowaux` is: $\{v \in Out \setminus In, |N(v) \cap (V \setminus Out)| = 1\} \subseteq C$ and $\forall v \in Out \setminus In$, $Past(v) = |N(v) \cap (V \setminus Out)|$.

By induction on the number of non output vertices, we prove that if the given open graph has a causal flow then the algorithm outputs a maximally delayed one. Assume that the given open graph has a causal flow. First, if there is no non output vertex, then no correction is needed: the empty flow $(g, \emptyset)$ (where g is a function with an empty domain) is a maximally delayed gflow. Now suppose that there exist some non output vertices, according to Lemma 4 the elements of $V_1^{\prec}$ have a neighbour in C and thus they are considered in the loop line 13. The test line 14 ensures that they are considered only once: if a vertex can be corrected by two vertices in C then only the first is considered. After the loop (line 27), $Out := Out \cup V_1^{\prec}$. At each modification of the output set (line 15) the preconditions are maintained. Indeed, the number of non output neighbours of this vertex is computed (line 17), and the number of non output neighbours of the potential correctors (vertices in $In \setminus Out$) is updated (line 23). Notice that

```
   input  : An open graph
   output: A causal flow
 1 Flow ((V,N),In,Out) =
 2 begin
 3 |  C:=∅ ; Past:= 0^n ; l := 0^n ;
 4 |  for all v ∈ Out \ In do
 5 |  |  Past(v):=|N(v) ∩ (V \ Out)| ;
 6 |  |  if Past(v) = 1 then C := C ∪ {v} ;
 7 |  end
 8 |  return Flowaux ((V,N),In,Out,C,Past,1);
 9 end

10 Flowaux ((V,N),In,Out,C,Past,k) = begin
11 |  C′:=∅;
12 |  for all v ∈ C do
13 |  |  if |N(v) ∩ (V \ Out)| = 1 then
14 |  |  |  {u} := N(v) ∩ (V \ Out) ; g(u) := v ; l(u) := k ; Out := Out ∪ {u} ;
15 |  |  |  if u ∉ In then
16 |  |  |  |  Past(u) := |N(u) ∩ (V \ Out)| ;
17 |  |  |  |  if Past(u) = 1 then C′:=C′ ∪ {u} ;
18 |  |  |  end
19 |  |  |  for all w ∈ N(u) do
20 |  |  |  |  if Past(w) > 0 then
21 |  |  |  |  |  Past(w) := Past(w) − 1 ;
22 |  |  |  |  |  if Past(w) = 1 then C′ := C′ ∪ {w} ;
23 |  |  |  |  end
24 |  |  |  end
25 |  |  end
26 |  end
27 |  if C′ = ∅ then
28 |  |  return (Out = V, (g, l)) ;
29 |  else
30 |  |  return Flowaux ((V,N),In,Out,C′,Past,k + 1) ;
31 |  end
32 end
```

Algorithm 1. Causal flow

after the insertion of u in Out, the vertices that may become a potential corrector for the next recursive call are u and some of the neighbours of u (loop line 20). Thus, C' satisfies the precondition $\{v \in Out \setminus In, |N(v) \cap (V \setminus Out)| = 1\} \subseteq C'$ for the recursive call (line 31).

Since the existence of a causal flow is assumed, $V_1^{\prec}$ cannot be empty (every output vertex has to be corrected), thus the algorithm is called recursively. Lemma 2 ensures the existence of a causal flow in $(G, I, O \cup V_1^{\prec})$.The induction hypothesis ensures that the recursive calls output a maximally delayed causal

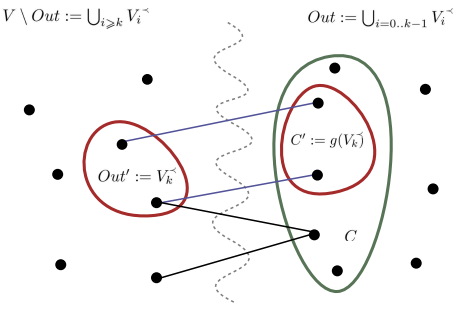


Fig. 3. Causal flow algorithm: At the k^{th} recursive call, the algorithm finds out the set $V_k^{\prec}$ composed of the qubits that will be measured at the $d^{\prec} - k + 1$ step of the one-way quantum computation, where $d^{\prec}$ is the depth of the computation. At that step, all the qubits in $Out := \bigcup_{i=0..k-1} V_k^{\prec}$ are not measured, whereas the qubits in $\bigcup_{i>k} V_k^{\prec}$ are already measured. The correctors of the elements of $V_k^{\prec}$ are in a set $C \subseteq Out \setminus In$ of candidates composed of vertices u not already assigned to the correction of some future measurements ($C = \{v, Past(v) \geq 1\}$). Since at each step, a maximum number of vertices are added to $V_k^{\prec}$, the causal flow, if it exists, produced by this algorithm is maximally delayed.

flow in $(G, I, O \cup V_1^{\prec})$ and thus the produced causal flow $(g, \prec)$ is a maximally delayed causal flow of (G, I, O).

The termination of the algorithm is ensured by the fact that the set of output vertices strictly increases at each recursive call.

For a given open graph, if the algorithm outputs $(true, (g, \prec))$, then $(g, \prec)$ is a valid causal flow since every output vertex has an image under g, moreover for any vertex i, $i \prec g(i)$, and finally if $j \in N(g(i))$ then $j = i$ or $i \prec j$. Thus, if the given open graph has no flow, the algorithm returns false.

For the analysis of the complexity of the algorithm, we consider that testing whether a vertex is an input (resp. ouput) can be done in constant time. This can be achieved with an additional cost of $n + |In|$ (resp. $2n + |Out|$) by building a boolean array of size n. The additional n for the outputs comes from the fact that the output set has to be maintained as it changes during the algorithm. Each vertex v is inserted at most once in a set C. The cost associated with the vertex v consists in

- Finding the predecessor by g: $\{u\} = N(v) \cap (V \setminus Out)$ which costs the degree $\delta(v)$ of v assuming that the graph is given as an adjacency list ;
- Computing $Past(v) = |N(v) \cap (V \setminus Out)|$ when inserting one of its neighbours in C which also costs $\delta(v)$;
- Adding it to Out and C which has a constant cost ;
- Decreasing $Past(v)$ and testing if $Past(v) = 1$ which occurs at most $\delta(v)$ times.

Thus, the total cost of the algorithm is upper bounded by $O(\sum \delta(v)) = O(m)$.□

This result improves the algorithm in [6] that decides, under the precondition $|I| = |O|$, whether an open graph (G, I, O) has a causal flow in $O(km)$ operations,

where $k = |O|$. In [10], Pei and de Beaudrap have proved that an open graph which has a causal flow has at most $(n-1)k - \binom{k}{2}$ edges. According to this result, the algorithm in [6] can be transformed (see [10]) into a $O(k^2 n)$-algorithm, whereas our algorithm becomes a $O(\min(m, kn))$-algorithm.

4 A Polynomial Algorithm for Gflow

In this section, we prove that the ideas of the algorithm 1 can be extended to derive a polynomial time algorithm in the more general case of the gflow, where each measurement is corrected by a set of qubits, instead of a single qubit. Since the existence of such a corrective strategy is sufficient and necessary for a large family of derterministic computations, the following algorithm decides whether a given one-way quantum computation is a member of such family of deterministic computation.

Theorem 2. *There exists a polynomial time algorithm that decides whether a given open graph has a gflow, and which outputs a gflow if it exists.*

Proof. In order to prove Theorem 2, we introduce a polynomial time algorithm which decides whether a given open graph has a gflow. Moreover, if a gflow exists, the algorithm outputs a maximally delayed gflow.

Let (G, I, O) be an open graph. The algorithm `gFlow`(Γ, I, O) (Algorithm 2), where Γ is the adjacency matrix of G, finds a maximally delayed gflow $(g, \prec)$ and returns $(true, (g, \prec))$ if one exists and returns *false* otherwise. Given a set Y and a subset $X \subseteq Y$, $\mathbb{I}_X$ stands for a $|Y|$-dimensional vector defined by $\mathbb{I}_X(i) = 1$ if $i \in X$ and $\mathbb{I}_X(i) = 0$ otherwise.

At the k^{th} recursive call, the set C found by the algorithm at the end of the loop at line 16 corresponds to the layer $V_k^{\prec}$ of the partition induced by the returned strict partial order. At line 12, the columns of the sub-matrix $\Gamma_{V \setminus Out, Out \setminus In}$ correspond to the vertices that can be used for correction (vertices in $\cup_{i<k} V_k^{\prec} \setminus In$) and the rows to the candidates for the set $V_k^{\prec}$. A solution X_0 in $\mathbb{F}_2$ to $\Gamma_{V \setminus Out, Out \setminus In} \mathbb{I}_X = \mathbb{I}_{\{u\}}$ is a subset of $\cup_{i<k} V_k^{\prec} \setminus In$ that has only u as odd neighbourhood in $\cup_{i \geq k} V_i^{\prec}$, thus $g(u) := X_0$ satisfies conditions 2 and 3 required by the definition of gflow (see Definition 3). Furthemore, line 11 of the algorithm ensures condition 1, thus if the algorithm returns a flow then it satisfies the definition of gflows.

Now suppose that the graph admits a gflow $(g, \prec)$, then it also admits a maximally delayed gflow $(g', \prec')$. The algorithm finds the set $V_1^{\prec'}$ (in the loop at line 11), and by induction (similarly to the induction in the proof of Theorem 1) it also finds a maximally delayed gflow in $(G, I, O \cup V_1^{\prec})$ with the recursive call. Thus the algorithm finds a maximally delayed gflow. □

In order to analyse the complexity, notice that lines 11 to 16 consists in solving a system $Ax = b_i$ for $n - \ell$ different b_is where $n = |V|$, $\ell = |Out|$ and A is a $(n - \ell) \times \ell$ matrix. In order to solve these $n - \ell$ systems, the $(n - \ell) \times n$-matrix $M = [A|b_1 \ldots b_{n-\ell}]$ is transformed into an upper triangular form within

```
   input  : An open graph
   output: A generalised flow
 1 gFlow (V, Γ, In, Out) =
 2 begin
 3 |  for all v ∈ Out do
 4 |  |  l(v) := 0 ;
 5 |  end
 6 |  return gFlowaux (V, Γ, In, Out, 1) ;
 7 end

 8 gFlowaux (V, Γ, In, Out, k) =
 9 begin
10 |  C := ∅ ;
11 |  for all u ∈ V \ Out do
12 |  |  Solve in F_2 : Γ_{V\Out, Out\In} I_X = I_{{u}} ;
13 |  |  if there is a solution X_0 then
14 |  |  |  C := C ∪ {u} ; g(u) := X_0 ; l(u) := k ;
15 |  |  end
16 |  end
17 |  if C = ∅ then
18 |  |  return (Out = V, (g, l)) ;
19 |  else
20 |  |  return gFlowaux (V, Γ, In, Out ∪ C, k + 1) ;
21 |  end
22 end
```

Algorithm 2. Generalised flow

$O(n^3)$ operations using gaussian eliminations for instance, then for each b_i a back substitution within $O(n^2)$ operations is used to find x_i, if it exists, such that $Ax_i = b_i$ (see [1]). The back substitutions cost $O(n^3)$ operations at each call of the function. Since there are at most n recursive calls, the overall complexity is $O(n^4)$.

5 Depth Optimality

We consider in this section the depth of the flows produced by the algorithms. The depth of a given flow is nothing but the depth of any one-way quantum computation based on the correction strategy described by this flow, even if the preparation of the initial graph state is taken into account since any graph state can be prepared in a constant depth [9].

Theorem 3. *A maximally delayed causal flow (resp. gflow) has minimum depth.*

Proof. Let $(g, \prec)$ be a minimum depth causal flow (resp. gflow) of a given open graph. If $(g, \prec)$ is a maximally delayed causal flow (resp. gflow), then let $(g', \prec')$ $= (g, \prec)$. Otherwise, let $(g', \prec')$ be a maximally delayed causal flow (resp. gflow)

which is more delayed than $(g, \prec)$. $(g', \prec')$ and $(g, \prec)$ have the same depth. Indeed $|\cup_{i=0\ldots d^{\prec}} V_k^{\prec'}| \geq |\cup_{i=0\ldots d^{\prec}} V_k^{\prec}| = |V|$, thus $\forall k > d^{\prec}, V_k^{\prec} = \emptyset$, so $d^{\prec} \geq d^{\prec'}$. Since $(g, \prec)$ has minimum depth $d^{\prec} \leq d^{\prec'}$, so $d^{\prec} = d^{\prec'}$. As a consequence $(g', \prec')$ is a minimum-depth maximally delayed causal flow (resp. gflow). Moreover, even if a maximally delayed causal flow (resp. gflow) of a given open graph is not unique, one can prove, using Lemmas 2, 3, and 4, that all the maximally delayed causal flows (resp. gflows) of a given open graph induce the same partition of the vertices, and as a consequence, have the same depth. Thus, a maximally delayed causal flow (resp. gflow) has the same depth as $(g', \prec')$ which is a minimum depth flow. □

Notice that the algorithms 1 and 2 produce maximally delayed flows, thus:

Corollary 1. *The previous algorithms find an optimal depth flow.*

The depth optimality of the flows found by the previous algorithms have several decisive implications in one-way quantum computation. First, the depth (optimal or not) of a flow is an upper bound on the depth of the corresponding deterministic one-way quantum computation. Moreover, if the one-way quantum computation is uniformly, stepwise and strongly deterministic (which mainly means that if the measurements are applied with an error in the angle which characterises the measurement, then the computation is still deterministic), then the correction strategy must be described by a gflow [3]. As a consequence the algorithm 2 produces the optimal correction strategy, and the depth of the gflow produced by the algorithm is a lower bound on the depth of a uniformly, stepwise and strongly deterministic one-way quantum computation.

6 Conclusion

Starting from quantum computational problems (determinism in one-way quanum computation), interesting graph problems have arisen as the property that the depth of correcting strategies for measurement-based quantum computation depends on flows in graphs. We have defined in this paper two algorithms for finding optimal causal flow and gflow. The key points are: the simplification of the structure of the gflows considering only the maximally delayed flows which have a nice recursive structure; a backward analysis (start from the outputs) which allows to take advantage of this structure and avoids backtracking.

From a complexity point of view, an important open question is: given a graph state and a fixed set of measurements (we relax the uniformity condition) what would be the depth of an optimal correction strategy. One direction to answer that question would be to define a weaker flow that is still polynomially computable. One can also consider the characterisation and the depth of computation in more generalised measurement-based models where other planes of measurements are allowed. Finally, these results open up new perspectives of depth optimisation in the more traditional model of quantum circuits: any circuit can be represented, in the one-way model, as an open graph that has a causal flow; moreover, the application on this open graph of the gflow algorithm

introduced in this paper produces a gflow of minimal depth. Investigating how such a one-way quantum computation of minimal depth can be translated back to a quantum circuit which has a smaller depth than the original circuit (but probably more ancillary qubits), should lead to a novel approach to reducing the depth complexity of quantum circuits.

Acknowledgements

The authors would like to thank Elham Kashefi, Philippe Jorrand and Thierry Boy de la Tour for fruitful discussions.

References

1. Bard, G.V.: Achieving a log(n) Speed Up for Boolean Matrix Operations and Calculating the Complexity of the Dense Linear Algebra step of Algebraic Stream Cipher Attacks and of Integer Factorization Methods. Cryptology ePrint Archive, Report 2006/163 (2006)
2. Broadbent, A., Kashefi, E.: Parallelizing Quantum Circuits. arXiv, quant-ph 0704.1806 (2007)
3. Browne, D., Kashefi, E., Mhalla, M., Perdrix, S.: Generalized flow and determinism in measurement-based quantum computation. NJP 9, 250 (2007)
4. Danos, V., Kashefi, E.: Determinism in the one-way model PRA, 74 (2006)
5. Danos, V., Kashefi, E., Panangaden, P.: The measurement calculus. J. ACM 54(2) (2007)
6. de Beaudrap, N.: Finding flows in the one-way measurement model. Phys. Rev. A 77, 022328 (2008)
7. de Beaudrap, N.: Complete algorithm to find flows in the one-way measurement model (2006) arXiv, quant-ph 0603072
8. Hein, M., Dür, W., Eisert, J., Raussendorf, R., Van den Nest, M., Briegel, H.J.: Entanglement in graph states and its applications. In: Proc. of the Int. School of Physics Enrico Fermi on Quantum Computers, Algorithms and Chaos (July 2005) quant-ph/0602096
9. Høyer, P., Mhalla, M., Perdrix, S.: Resources required for preparing graph states. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288. Springer, Heidelberg (2006)
10. Pei, M., de Beaudrap, N.: An extremal result for geometries in the one-way measurement model. Quantum Information and Computation 8(5) (2008)
11. Raussendorf, R., Briegel, H.: A one-way quantum computer. PRL 86 (2001)
12. Raussendorf, R., Briegel, H.: Computational model underlying the one-way quantum computer. Quantum Information and Computation 2(6) (2002)
13. Walther, P., Resch, K., Rudolph, T., Schenck, E., Weinfurter, H., Vedral, V., Aspelmeyer, M., Zeilinger, A.: Experimental one-way quantum computing. Nature 434 (2005) (quant-ph/0503126)

Optimal Quantum Adversary Lower Bounds for Ordered Search

Andrew M. Childs[1] and Troy Lee[2]

[1] Department of Combinatorics & Optimization and
Institute for Quantum Computing, University of Waterloo
[2] Department of Computer Science, Rutgers University

Abstract. The goal of the ordered search problem is to find a particular item in an ordered list of n items. Using the adversary method, Høyer, Neerbek, and Shi proved a quantum lower bound for this problem of $\frac{1}{\pi}\ln n + \Theta(1)$. Here, we find the exact value of the best possible quantum adversary lower bound for a symmetrized version of ordered search (whose query complexity differs from that of the original problem by at most 1). Thus we show that the best lower bound for ordered search that can be proved by the adversary method is $\frac{1}{\pi}\ln n + O(1)$. Furthermore, we show that this remains true for the generalized adversary method allowing negative weights.

1 Introduction

Search is a fundamental computational task. In a general search problem, one is looking for a distinguished item in a set, which may or may not have some structure. At one extreme, in the *unstructured search problem*, we assume the set has no additional structure whatsoever. In this setting, a classical search requires $\Omega(n)$ queries to find the distinguished item. Grover's well-known search algorithm shows that a quantum computer can find the distinguished item with high probability in only $O(\sqrt{n})$ queries [15]. A lower bound based on a precursor to the adversary method shows this is optimal up to a constant factor [6].

At the other extreme of search problems, in the *ordered search problem*, we assume our set comes equipped with a total order, and we are able to make comparison queries, i.e., queries of the form '$w \leq z$?'. Classically, we can apply binary search to find the desired item in $\lceil \log_2 n \rceil$ queries, and an information theoretic argument shows this is tight.

Quantum computers can speed up ordered search by a constant multiplicative factor. Farhi, Goldstone, Gutmann, and Sipser developed a class of translation-invariant ordered search algorithms and showed that one such algorithm, applied recursively, gives an exact ordered search algorithm using $3\log_{52} n \approx 0.526 \log_2 n$ quantum queries [13]. Brookes, Jacokes, and Landahl used a gradient descent search to find an improved translation-invariant algorithm, giving an upper bound of $4\log_{550} n \approx 0.439 \log_2 N$ queries [8]. Childs, Landahl, and Parrilo used numerical semidefinite optimization to push this approach still further, improving the upper bound to $4\log_{605} n \approx 0.433 \log_2 n$ [10]. Ben-Or and Hassidim gave

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 869–880, 2008.

an algorithm based on adaptive learning that performs ordered search with error probability $o(1)$ using only about $0.32 \log_2 n$ queries [7].

In fact, the quantum speedup for ordered search is not more than a constant multiplicative factor. Using the quantum adversary method [2], Høyer, Neerbek, and Shi showed a lower bound of $\frac{1}{\pi}(\ln n - 1) \approx 0.221 \log_2 n$ queries [17], improving on several previous results [1,9,12]. However, the exact value of the best possible speedup factor, a fundamental piece of information about the power of quantum computers, remains undetermined.

In this paper, we give some evidence that the asymptotic quantum query complexity of ordered search is $\frac{1}{\pi} \ln n + O(1)$. Specifically, we show that the best lower bound given by the adversary method, one of the most powerful techniques available for showing lower bounds on quantum query complexity, is $\frac{1}{\pi} \ln n + O(1)$. We show this both for the standard adversary method [2] and the recent strengthening of this method to allow negative weights [16]. In particular:

Theorem 1. *Let* $\mathrm{ADV}(f)$ *be the optimal bound given by the adversary method for the function* f*, let* $\mathrm{ADV}^{\pm}(f)$ *be the optimal value of the adversary bound with negative weights, and let* OSP_n *the ordered search problem on* n *items (symmetrized as discussed in Section 4). Then*

$$\mathrm{ADV}(\mathrm{OSP}_{2m}) = 2 \sum_{i=0}^{m-1} \left(\frac{\binom{2i}{i}}{4^i} \right)^2$$

$$\mathrm{ADV}(\mathrm{OSP}_{2m+1}) = 2 \sum_{i=0}^{m-1} \left(\frac{\binom{2i}{i}}{4^i} \right)^2 + \left(\frac{\binom{2m}{m}}{4^m} \right)^2.$$

Furthermore, $\mathrm{ADV}^{\pm}(\mathrm{OSP}_n) \leq \mathrm{ADV}(\mathrm{OSP}_n) + O(1)$.

These bounds are asymptotically $\frac{2}{\pi} \ln n + O(1)$, but are always strictly larger than the Høyer-Neerbek-Shi bound. Understanding the best possible adversary bound for small n could be useful, since the best exact algorithms for ordered search have been found by discovering a good algorithm for small values of n and using this algorithm recursively. Furthermore, since the adversary quantity can be viewed as a simplification of the quantum query complexity, we hope that our analytic understanding of optimal adversary bounds will provide tools that are helpful for determining the quantum query complexity of ordered search.

2 Adversary Bound

The adversary method, along with the polynomial method [5], is one of the two main techniques for proving lower bounds on quantum query complexity. The adversary method was originally developed by Ambainis [2], with roots in the hybrid method of [6]. It has proven to be a versatile technique, with formulations given by various authors in terms of spectral norms of matrices [4], weight schemes [3,21], and Kolmogorov complexity [18]. Špalek and Szegedy showed

that all these versions of the adversary method are in fact equivalent [20]. Recently, Høyer, Lee, and Špalek developed a new version of the adversary method using negative weights which is always at least as powerful as the standard adversary method, and can sometimes give better lower bounds [16].

We will use the spectral formulation of the adversary bound, as this version best expresses the similarity between the standard and negative adversary methods. In this formulation, the value of the adversary method for a function f is given by

$$\mathrm{ADV}(f) := \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma \circ D_i\|},$$

where Γ is a square matrix with rows and columns indexed by the possible inputs $x \in S \subseteq \{0,1\}^n$, constrained to satisfy $\Gamma[x,y] = 0$ if $f(x) = f(y)$; D_i is a zero/one matrix with $D_i[x,y] = 1$ if $x_i \neq y_i$ and 0 otherwise; $A \circ B$ denotes the Hadamard (i.e., entrywise) product of matrices A and B; and $\Gamma \geq 0$ means that the matrix Γ is entrywise non-negative. Note that the set S of possible inputs need not be the entire set $\{0,1\}^n$ of all n-bit strings—in other words, f might be a partial function, as is the case for ordered search.

The negative adversary method is of the same form, but removes the restriction to non-negative matrices in the maximization. Thus the value of the negative adversary method for a function f is given by

$$\mathrm{ADV}^{\pm}(f) := \max_{\Gamma \neq 0} \frac{\|\Gamma\|}{\max_i \|\Gamma \circ D_i\|}.$$

The relation of these adversary values to quantum query complexity is given by the following theorem. Let $Q_\epsilon(f)$ denote the minimum number of quantum queries to f needed to compute that function with error at most ϵ. Then

Theorem 2 ([2,16]). *Let $S \subseteq \{0,1\}^n$, and let Σ be a finite set. Then for any function $f : S \to \Sigma$,*

$$Q_\epsilon(f) \geq \frac{1 - 2\sqrt{\epsilon(1-\epsilon)}}{2}\,\mathrm{ADV}(f) \text{ and } Q_\epsilon(f) \geq \frac{1 - 2\sqrt{\epsilon(1-\epsilon)} - 2\epsilon}{2}\,\mathrm{ADV}^{\pm}(f).$$

In particular, $Q_0(f) \geq \frac{1}{2}\,\mathrm{ADV}^{\pm}(f) \geq \frac{1}{2}\,\mathrm{ADV}(f)$.

3 Ordered Search Problem

In the ordered search problem, we are looking for a marked element w in a set $Z = \{z_1, z_2, \ldots, z_n\}$ equipped with a total order, such that $z_1 < z_2 < \cdots < z_n$. We are able to make queries of the form '$w \leq z$?' for any $z \in Z$. If $w = z_i$, then the answer to this query will be 'no' for $z = z_j$ with $j < i$, and will be 'yes' otherwise. We can model this problem as finding the first occurrence of a '1' in a string $x \in \{0,1\}^n$ where $x_j = 0$ for $j < i$ and $x_j = 1$ otherwise. Thus we have transformed the input into a binary string, such that the queries are to the bits

of the input. The goal is to determine which input we have—in other words, the function takes a different value on each input.

In general, when trying to determine the query complexity of a function f, it is helpful to consider its symmetries, as expressed by its *automorphism group*. We say that $\pi \in S_n$, a permutation of the n bits of the input, is an automorphism of the function f provided it maps inputs to inputs, and $f(x) = f(y) \Leftrightarrow f(\pi(x)) = f(\pi(y))$. The set of automorphisms of any function on n-bit inputs is a subgroup of S_n, called the automorphism group of that function.

The ordered search problem as formulated above has a trivial automorphism group, because any nontrivial permutation maps some input to a non-input. However, we can obtain a more symmetric function, with only a small change to the query complexity, by putting the input on a circle [13]. Now let the inputs have $2n$ bits, and consist of those strings obtained by cyclically permuting the string of n 1's followed by n 0's. Again, we try to identify the input, so the function OSP_n takes a different value on each of the $2n$ inputs. The automorphism group of OSP_n is isomorphic to $\mathbb{Z}_{2n}$, a fact that we will exploit in our analysis.

The query complexity of this extended function is closely related to that of the original function. Given an n-bit input x, we can simulate a $2n$-bit input by simply querying x for the first n bits, and the complement of x for the second n bits. In the other direction, to simulate an n-bit input using a $2n$-bit input, first query the nth bit of the $2n$-bit input. If it is 1, then we use the first half of the $2n$-bit input; otherwise we use the second half (or, equivalently, the complement of the first half). Thus the query complexity of the extended function is at least that of the original function, and at most one more than that of the original function, a difference that is asymptotically negligible.

4 Adversary Bounds for Ordered Search

Finding the value of the adversary method is an optimization problem. We can simplify this problem using the symmetry of OSP_n. The same simplification applies to both the standard and negative adversary bounds, so we treat the two cases simultaneously. Specifically, we use the following result:

Theorem 3 (Automorphism principle [16]). *Let G be the automorphism group of f. Then there is an optimal adversary matrix Γ satisfying $\Gamma[x,y] = \Gamma[\pi(x),\pi(y)]$ for all $\pi \in G$ and all pairs of inputs x, y. Furthermore, if G acts transitively on the inputs (i.e., if for every x, y there is an automorphism taking x to y), then the uniform vector (i.e., the vector with each component equal to 1) is a principal eigenvector of Γ.*

The automorphism group for OSP_n is generated by the element $(1\,2\,3\ldots2n)$ that cyclically permutes the list. This group acts transitively on the inputs, so by the automorphism principle, the uniform vector is a principal eigenvector of the adversary matrix. In addition, any input pairs (x, y) and (x', y') that have the same Hamming distance are related by an automorphism. Thus we may assume that the adversary matrix has at most n distinct entries, and that the (x, y)

entry depends only on the Hamming distance between x and y. As all strings have the same Hamming weight, the Hamming distance between any pair is even. We let $\Gamma[x,y] = \gamma_i$ when x, y have Hamming distance $2i$. Since all rows have the same sum, the uniform vector is indeed an eigenvector, corresponding to the eigenvalue $\gamma_n + 2\sum_{i=1}^{n-1} \gamma_i$.

Transitivity of the automorphism group also implies that all matrices $\Gamma \circ D_i$ have the same norm, so it is sufficient to consider $\Gamma \circ D_{2n}$. This matrix consists of two disjoint blocks, where each block is an $n \times n$ symmetric Toeplitz matrix with first row equal to $(\gamma_n, \gamma_{n-1}, \ldots, \gamma_1)$, denoted $\text{Toeplitz}(\gamma_n, \gamma_{n-1}, \ldots, \gamma_1)$. Thus we have reduced the adversary bound to the semidefinite program (SDP)

$$\max\ \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\,\text{Toeplitz}(\gamma_n, \gamma_{n-1}, \ldots, \gamma_1)\| \le 1,\ \gamma_i \ge 0 \quad \text{(P)}$$

in the case of non-negative weights, and

$$\max\ \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\,\text{Toeplitz}(\gamma_n, \gamma_{n-1}, \ldots, \gamma_1)\| \le 1 \qquad (\text{P}^{\pm})$$

in the case of the negative adversary method. We emphasize that the automorphism principle ensures there is no loss of generality in considering adversary matrices of this form—this program has the same optimal value as the best possible adversary bound.

We will also use the duals of these SDPs to give upper bounds on the adversary methods. Straightforward dualization shows that the dual of (P) is

$$\min \text{Tr}(P) \quad \text{subject to} \quad P \succeq 0,\ \ \text{Tr}_i(P) \ge 1 \text{ for } i = 0, \ldots, n-1 \qquad \text{(D)}$$

(where $\text{Tr}_i(P) := \sum_{j=1}^{n-i} P[j, i+j]$), and that the dual of ($\text{P}^{\pm}$) is

$$\min \text{Tr}(P{+}Q) \quad \text{subject to} \quad P, Q \succeq 0,\ \ \text{Tr}_i(P{-}Q) = 1 \text{ for } i = 0, \ldots, n{-}1 \quad (\text{D}^{\pm})$$

where $P \succeq 0$ means that the matrix P is positive semidefinite.

In general, by a *solution* of an SDP, we mean a choice of the variables that satisfies the constraints, but that does not necessarily extremize the objective function. If a solution achieves the optimal value of the objective function, we refer to it as an *optimal solution*.

5 Non-negative Adversary

5.1 Høyer, Neerbek, Shi Construction

Within the framework described above, the lower bound of [17] can be given very simply. Set $\gamma_i = 0$ if $i > \lfloor n/2 \rfloor$ and $\gamma_i = 1/(\pi i)$ otherwise. This gives an objective function of

$$\frac{2}{\pi} \sum_{i=1}^{\lfloor n/2 \rfloor} \frac{1}{i} \sim \frac{2}{\pi} \ln n.$$

Under this choice of weights, the matrix $\Gamma \circ D_{2n}$ consists of four disjoint nonzero blocks (and two additional 2×2 zero blocks in the case of n odd), so its spectral norm is equal to the largest spectral norm of these blocks. Each nonzero block is equivalent up to permutation to $1/\pi$ times $Z_{\lfloor n/2 \rfloor}$, where Z_m is the *half Hilbert matrix* of size $m \times m$, the Hankel matrix with entries $Z_m[i,j] = 1/(i+j-1)$ for $i+j-1 \leq m$, and $Z_m[i,j] = 0$ otherwise. This may be compared with the usual Hilbert matrix, whose (i,j) entry is $1/(i+j-1)$. The spectral norm of any finite Hilbert matrix is at most π, so as the half Hilbert matrix is non-negative and entrywise less than the Hilbert matrix, its spectral norm is also at most π. (See the delightful article of Choi for this and other interesting facts about the Hilbert matrix [11].) This shows that the spectral norm of each matrix $\Gamma \circ D_i$ is at most 1, giving a bound on the zero-error quantum query complexity of ordered search of approximately $\frac{1}{\pi} \ln n$.

5.2 Optimal Non-negative Construction

It turns out that one can do slightly better than the Hilbert weight scheme described above. Here we construct the optimal solution to the adversary bound for OSP_n with non-negative weights.

A key role in our construction will be played by the sequence $\{\xi_i\}$, where

$$\xi_i := \frac{\binom{2i}{i}}{4^i}. \tag{1}$$

This sequence has many interesting properties. First, it is monotonically decreasing. Consider the ratio

$$\frac{\xi_{i+1}}{\xi_i} = \frac{\binom{2(i+1)}{i+1} 4^i}{\binom{2i}{i} 4^{i+1}} = \frac{2(i+1)(2i+1)}{4(i+1)^2} = \frac{i+1/2}{i+1} < 1.$$

Indeed, this shows that $\{\xi_i\}$ is a hypergeometric sequence with the generating function

$$g(z) := \sum_{i=0}^{\infty} \xi_i z^i = {}_1F_0(\tfrac{1}{2}; z) = \frac{1}{\sqrt{1-z}}.$$

These observations lead us to the next interesting property of our sequence.

Proposition 4. *For any j, $\sum_{i=0}^{j} \xi_i \xi_{j-i} = 1$.*

Proof. The product $g(z)^2$ is the generating function for the convolution appearing on the left hand side. But $g(z)^2 = (1-z)^{-1}$, which has all coefficients equal to 1, as claimed. (For an alternative proof, using the fact that $\xi_i = (-1)^i \binom{-1/2}{i}$, see [14, p. 187].) □

This proposition shows that the sequence $\{\xi_i\}$ behaves nicely under convolution. We will also consider the behavior of $\{\xi_i\}$ under correlation. Define

$$A_m(j) := \sum_{i=0}^{m-j-1} \xi_i \xi_{i+j}.$$

As $\{\xi_i\}$ is a monotonically decreasing sequence, it follows that $A_m(j)$ is a monotonically decreasing function of j. With these definitions in hand, we are now ready to construct our adversary matrix.

Proof (Theorem 1, lower bound on non-negative adversary). We first consider the case where $n = 2m$ is even. In (P), let

$$\gamma_i = A_m(i-1) - A_m(i).$$

As $A_m(j)$ is a monotonically decreasing function of j, we have $\gamma_i \geq 0$. Also note that $A_m(i) = 0$ for $i \geq m$, so $\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)$ is bipartite.

The objective function is a telescoping series, so the value of the SDP is

$$2A_m(0) = 2\sum_{i=0}^{m-1} \xi_i^2,$$

as claimed. Thus it suffices to show that $\|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1$.

We will show that, in fact, $\|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| = 1$. We do this by exhibiting an eigenvector u with eigenvalue 1, and with strictly positive entries. This will finish the proof by the following argument: As $\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)$ is a non-negative, symmetric matrix, its spectral norm is equal to its largest eigenvalue. By the Perron-Frobenius theorem, it has a principal eigenvector with non-negative entries. As the eigenvectors of a symmetric matrix corresponding to distinct eigenvalues are orthogonal, and no non-negative vector can be orthogonal to u, we conclude that the largest eigenvalue must agree with the eigenvalue of u, and so is 1.

The relevant eigenvector of $\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)$ is

$$u := (\xi_0, \xi_1, \ldots, \xi_{m-1}, \xi_{m-1}, \ldots, \xi_1, \xi_0). \tag{2}$$

Computing $\text{Toeplitz}(g_n, \ldots, g_1)u$, we see that u is an eigenvector with eigenvalue 1 provided

$$\sum_{i=0}^{m-j-1} \big(A_m(i+j) - A_m(i+j+1)\big)\xi_i = \xi_j \tag{3}$$

for each $j = 0, 1, \ldots, m-1$.

We prove (3) using generating functions. Define a complementary function to $g(z)$, namely the polynomial $h(z) := \xi_{m-1} + \xi_{m-2}z + \ldots + \xi_0 z^{m-1}$, and consider the product $g(z)h(z)$. For $i = 0, \ldots, m-1$, the coefficient of z^i in this series is $A_m(m-i-1)$, so the coefficient of z^i in $(1-z)g(z)h(z)$ is $A_m(m-i-1) - A_m(m-i)$. Thus the coefficient of z^{m-j-1} in $(1-z)g(z)h(z)g(z) = h(z)$ is the left hand side of (3). But the coefficient of z^{m-j-1} in $h(z)$ is the coefficient of z^j in $g(z)$, which is simply ξ_j, the right hand side of (3).

For $n = 2m+1$ odd, let

$$\gamma_i = \frac{1}{2}\big(A_{m+1}(i-1) - A_{m+1}(i) + A_m(i-1) - A_m(i)\big).$$

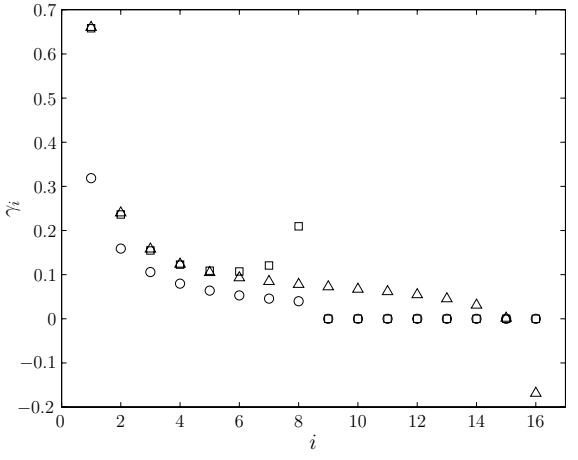


Fig. 1. Comparison of the weights γ_i with $n = 16$ for various adversary bounds: the bound of Høyer, Neerbek, and Shi (circles), the optimal non-negative adversary (squares), and the optimal negative adversary (triangles)

Then the objective function is

$$A_{m+1}(0) + A_m(0) = 2 \sum_{i=0}^{m-1} \xi_i^2 + \xi_m^2$$

as claimed. Now it suffices to show that

$$u := (\xi_0, \xi_1, \ldots, \xi_{m-1}, \xi_m, \xi_{m-1}, \ldots, \xi_1, \xi_0) \tag{4}$$

is an eigenvector of Toeplitz$(\gamma_n, \ldots, \gamma_1)$ with eigenvalue 1. (Note that for n odd, Toeplitz$(\gamma_n, \ldots, \gamma_1)$ is irreducible, so u is actually the unique principal eigenvector.) For all but the middle component of the vector Toeplitz$(\gamma_n, \ldots, \gamma_1)u$, the required condition is simply the average of (3) and the same equation with m replaced by $m + 1$. For the middle component, we require $A_{m+1}(m)\xi_0 = \xi_m$, which holds because $A_{m+1}(m) = \xi_0\xi_m$ and $\xi_0 = 1$. □

In the bound of Høyer, Neerbeck, and Shi, the weight given to a pair (x, y) is inversely proportional to the Hamming distance between x and y. This follows the intuition that pairs which are easier for an adversary to distinguish should be given less weight. It is interesting to note that the optimal weight scheme does *not* have this property—indeed, at large Hamming distances the weights actually increase with increasing Hamming distance, as shown in Figure 1.

5.3 Dual

We now show that this bound is optimal by giving a matching solution to the dual SDP (D).

Proof (Theorem 7, upper bound on non-negative adversary). Fix n, and let u be the vector of length n defined by (2) if n is even, or by (4) if n is odd. Notice that in either case, $u_i = u_{n-i+1}$. Let $P = uu^T$, a rank one matrix. This matrix is positive semidefinite, and its trace is $\|u\|^2$, which matches the value of our solution to the primal problem in the previous section. Thus it suffices to verify that $\mathrm{Tr}_i(P) \geq 1$. We have

$$\mathrm{Tr}_i(P) = \sum_{j=1}^{n-i} P[j, i+j] = \sum_{j=1}^{n-i} u_j u_{i+j} = \sum_{j=1}^{n-i} u_j u_{n-i-j+1}.$$

Since $\{\xi_i\}$ is monotonically decreasing in i, we have $u_j \geq \xi_{j-1}$, with equality holding when $j \leq \lceil n/2 \rceil$. Thus

$$\mathrm{Tr}_i(P) \geq \sum_{j=1}^{n-i} \xi_{j-1}\xi_{n-i-j} = 1$$

by Proposition 4. When $i > \lfloor n/2 \rfloor$, this inequality holds with equality. □

Having established the optimal adversary bound for OSP_n, let us examine its asymptotic behavior. Observing that the generating function for $\{\mathrm{ADV}(\mathrm{OSP}_{2m})\}$ is $\frac{4}{\pi}K(z)/(1-z)$, where $K(z) = \frac{\pi}{2}{}_2F_1(\frac{1}{2}, \frac{1}{2}; 1, z)$ is the complete elliptic integral of the first kind, and applying Darboux's method to estimate the asymptotic behavior of its coefficients, one can show

Corollary 5. $\mathrm{ADV}(\mathrm{OSP}_n) = \frac{2}{\pi}(\ln n + \gamma + \ln 8) + O(1/n)$, *where* $\gamma \approx 0.577$ *is the Euler-Mascheroni constant.*

For comparison, the bound of Høyer, Neerbek, and Shi for OSP_n is $\frac{2}{\pi}H_{\lfloor n/2 \rfloor} = \frac{2}{\pi}(\ln n + \gamma - \ln 2) + O(1/n)$, where $H_n := \sum_{i=1}^{n} \frac{1}{i}$ is the nth harmonic number. (Note that for the original, unsymmetrized ordered search problem treated in [17], the bound is $\frac{2}{\pi}(H_n - 1)$.) Indeed, the optimal value of the non-negative adversary is considerably better for small values of n, as shown in Figure 2.

6 Negative Adversary

We now turn to the negative adversary method, and give an upper bound on $\mathrm{ADV}^{\pm}(\mathrm{OSP}_n)$ by exhibiting a solution to ($\mathrm{D}^{\pm}$).

Notice that if we find a symmetric matrix R such that $\mathrm{Tr}_i(R) = 1$ for $i = 0, \ldots, n-1$, we can translate this into a solution to ($\mathrm{D}^{\pm}$) by decomposing $R = P - Q$ as the difference of two positive semidefinite matrices with disjoint support, letting P be the projection of R onto its positive eigenspace and letting Q be the projection of R onto its negative eigenspace. In this case, $\mathrm{Tr}(P+Q)$ is simply $\|R\|_{\mathrm{Tr}}$, the sum of the absolute values of the eigenvalues of R.

In looking for a matrix R satisfying $\mathrm{Tr}_i(R) = 1$ for all i, a natural starting point is our solution to the non-negative dual (D). Recall that in this construction, for

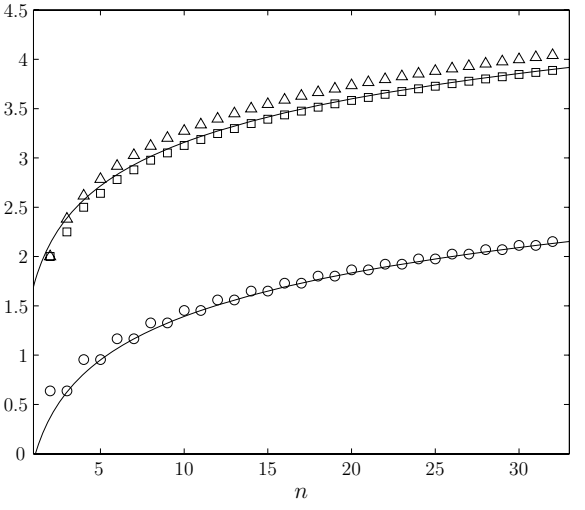


Fig. 2. Comparison of adversary lower bounds for ordered search: the bound of Høyer, Neerbek, and Shi (circles), the optimal non-negative adversary (squares), and the optimal negative adversary (triangles). The lower curve shows the asymptotic approximation $\frac{2}{\pi}(\ln n + \gamma - \ln 2)$ of the Høyer-Neerbek-Shi bound, and the upper curve shows the asymptotic approximation $\frac{2}{\pi}(\ln n + \gamma + \ln 8)$ of the non-negative adversary.

$i > \lceil n/2 \rceil$, the condition $\mathrm{Tr}_i(P) = 1$ held with equality. We imitate that construction by letting

$$R[i,j] = \begin{cases} \xi_i \xi_{n-j+1} & i \le j \\ \xi_{n-i+1} \xi_j & i > j. \end{cases}$$

Above the diagonal, R looks like a rank one matrix, but it is symmetrized below the diagonal. By the convolution property of the ξ_i's we see that $\mathrm{Tr}_i(R) = 1$ for $i = 0, \ldots, n-1$.

To upper bound the trace norm of R, the following lemma will be helpful:

Lemma 6. *Let M be an $n \times n$ matrix with entries*

$$M[i,j] = \begin{cases} v_i w_j & i \le j \\ v_j w_i & i > j \end{cases} \tag{5}$$

where the vectors $v, w \in \mathbb{R}^n$ have positive components, and satisfy $\frac{v_i}{v_{i+1}} > \frac{w_i}{w_{i+1}}$ for $i = 1, \ldots, n-1$. Then M has one positive eigenvalue and $n-1$ negative eigenvalues, and its trace norm satisfies

$$2\|v\|\|w\| - v \cdot w \le \|M\|_{\mathrm{Tr}} \le 2\|v\|\|w\| + v \cdot w.$$

We omit the proof, which is an application of Sylvester's law of inertia.

Now we are ready to finish the proof of Theorem 1.

Proof (Theorem 7, upper bound on negative adversary). The matrix R defined above is of the form (5) with $v = (\xi_0, \xi_1, \ldots, \xi_{n-1})$ and $w = (\xi_{n-1}, \xi_{n-2}, \ldots, \xi_0)$, the reversal of v. By Proposition 4, $\mathrm{Tr}_i(R) = 1$ for $i = 0, \ldots, n-1$, so R is a solution of ($\mathrm{D}^{\pm}$). Since v is monotonically increasing and w is monotonically decreasing, the conditions of Lemma 6 are satisfied, and thus $\|R\|_{\mathrm{Tr}} \leq 2\|v\|^2 + 1 = \mathrm{ADV}(\mathrm{OSP}_{2n}) + 1$.

Finally, from Corollary 5 we have $\mathrm{ADV}(\mathrm{OSP}_{2n}) - \mathrm{ADV}(\mathrm{OSP}_n) \leq \frac{2}{\pi} \ln 2 + O(1/n)$, so $\mathrm{ADV}^{\pm}(\mathrm{OSP}_n) \leq \mathrm{ADV}(\mathrm{OSP}_n) + 1 + \frac{2}{\pi} \ln 2 + O(1/n)$. □

Note that the solution of ($\mathrm{D}^{\pm}$) given above is not the optimal one. For fixed n, we can find the optimal solution using a numerical SDP solver. Figure 1 shows the optimal weights for $n = 16$, and Figure 2 shows the value of the optimal negative adversary bound for $n = 2$ through 32.

7 Conclusion

We have given upper bounds on the lower bounds provable by the quantum adversary method for the ordered search problem, showing that both the standard and negative adversary values are $\frac{2}{\pi} \ln n + \Theta(1)$. In particular, we have shown that establishing the quantum query complexity of ordered search will either require a lower bound proved by a different technique, or an improved upper bound. On the lower bound side, one could investigate the recently developed multiplicative adversary technique of Špalek [19]. However, we feel that it is more likely that the $\frac{1}{\pi} \ln n$ lower bound is in fact tight, and that further improvement will come from algorithms. As the current best upper bounds are ad hoc, based on numerical searches, they can almost certainly be improved.

The disagreeable reader may argue that upper bounds on lower bounds are only meta-interesting. We counter this objection as follows. Barnum, Saks, and Szegedy have exactly characterized quantum query complexity in terms of an SDP [4]. The adversary method can be viewed as a relaxation of this program, removing some constraints and focusing only on the output condition. Thus, our results can be viewed as solving a simplification of the quantum query complexity SDP, which might provide insight into the solution of the full program. Indeed, we hope that the results presented here will be a useful step toward determining the quantum query complexity of ordered search.

Acknowledgments

We thank Peter Høyer for stimulating discussions, and for suggesting an alternative proof of (3). This work was done in part while AMC was at the Caltech IQI, where he received support from NSF Grant PHY-0456720 and ARO Grant W911NF-05-1-0294; and while TL was at the LRI, Université Paris-Sud, where he was was supported by a Rubicon grant from the NWO and by the European Commission under the QAP Project, funded by the IST directorate as contract no. 015848.

References

[1] Ambainis, A.: A better lower bound for quantum algorithms searching an ordered list. In: Proc. 40th FOCS, pp. 352–357 (1999)

[2] Ambainis, A.: Quantum lower bounds by quantum arguments. Journal of Computer and System Sciences 64(4), 750–767 (2002); Preliminary version in STOC 2000

[3] Ambainis, A.: Polynomial degree vs. quantum query complexity. Journal of Computer and System Sciences 72(2), 220–238 (2006); Preliminary version in FOCS 2003

[4] Barnum, H., Saks, M., Szegedy, M.: Quantum query complexity and semidefinite programming. In: Proc. 18th CCC, pp. 179–193 (2003)

[5] Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. Journal of the ACM 48(4), 778–797 (2001); Preliminary version in FOCS 1998

[6] Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM Journal on Computing 26, 1510–1523 (1997)

[7] Ben-Or, M., Hassidim, A.: Quantum search in an ordered list via adaptive learning, quant-ph/0703231

[8] Brookes, E.M., Jacokes, M.B., Landahl, A.J.: An improved quantum algorithm for searching an ordered list (2004)

[9] Buhrman, H., de Wolf, R.: A lower bound for quantum search of an ordered list. Information Processing Letters 70(5), 205–209 (1999)

[10] Childs, A.M., Landahl, A.J., Parrilo, P.A.: Improved quantum algorithms for the ordered search problem via semidefinite programming. Physical Review A 75(3), 032335 (2007)

[11] Choi, M.-D.: Tricks or treats with the Hilbert matrix. American Mathematical Monthly 90(5), 301–312 (1983)

[12] Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: A limit on the speed of quantum computation for insertion into an ordered list, quant-ph/9812057

[13] Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Invariant quantum algorithms for insertion into an ordered list, quant-ph/9901059

[14] Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics. Addison-Wesley, Reading (1989)

[15] Grover, L.K.: Quantum mechanics helps in searching for a needle in a haystack. Physical Review Letters 79, 325–328 (1997); Preliminary version in STOC 1996

[16] Høyer, P., Lee, T., Špalek, R.: Negative weights make adversaries stronger. In: Proc. 39th STOC, pp. 526–535 (2007)

[17] Høyer, P., Neerbek, J., Shi, Y.: Quantum complexities of ordered searching, sorting, and element distinctness. Algorithmica 34(4), 429–448 (2002); Preliminary version in ICALP 2001

[18] Laplante, S., Magniez, F.: Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. In: Proc. 19th CCC, pp. 294–304 (2004)

[19] Špalek, R.: The multiplicative quantum adversary. In: Proc. 23rd CCC (to appear, 2008), available at quant-ph/0703237

[20] Špalek, R., Szegedy, M.: All quantum adversary methods are equivalent. Theory of Computing 2(1), 1-18 (2006); Preliminary version In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1–18. Springer, Heidelberg (2005)

[21] Zhang, S.: On the power of Ambainis's lower bounds. Theoretical Computer Science 339(2-3), 241–256 (2005); Preliminary version In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 241–256. Springer, Heidelberg (2004)

Quantum SAT for a Qutrit-Cinquit Pair Is QMA_1-Complete

Lior Eldar and Oded Regev*

School of Computer Science
Tel-Aviv University

Abstract. We show that the quantum SAT problem is QMA_1-complete when restricted to interactions between a three-dimensional particle and a five-dimensional particle. The best previously known result is for particles of dimensions 4 and 9. The main novel ingredient of our proof is a certain Hamiltonian construction named the *Triangle Hamiltonian*. It allows to verify the application of a 2-qubit CNOT gate without generating explicitly interactions between pairs of workspace qubits. We believe this construction may contribute to progress in other Hamiltonian-related problems as well as in adiabatic computation.

1 Introduction

In the k-SAT problem we are given a formula on n binary variables $x_1, \ldots, x_n$ of the form

$$\Phi_1 \wedge \Phi_2 \wedge \cdots \wedge \Phi_l$$

where each clause $\Phi_i : \{0,1\}^k \to \{T, F\}$ for $1 \leq i \leq l$ is a function on k of the variables.[1] We are then asked whether there exists an assignment to these variables that satisfies simultaneously all clauses. This problem is known to be NP-complete for $k \geq 3$ and solvable in polynomial time for $k \leq 2$. For instance, the following is a satisfiable 2-SAT instance on binary variables x_1, x_2, x_3,

$$(x_1 \vee \neg x_2) \wedge (x_1 \neq x_3). \tag{1}$$

One may investigate the behavior of such formulas when the variables are not necessarily binary. For instance, the 2-SAT problem with ternary variables is NP-complete as it includes the NP-complete 3-coloring problem as a special case (see, e.g., [1]). We denote this problem by $(3,3)$-SAT. Moreover, the 2-SAT problem, with each clause containing one binary variable and one ternary variable, which we denote by $(2,3)$-SAT, is still NP-complete. This can be shown, for instance, by a simple reduction from 3-coloring.

* Supported by the Binational Science Foundation, by the Israel Science Foundation, and by the European Commission under the Integrated Project QAP funded by the IST directorate as Contract Number 015848.

[1] Strictly speaking, this problem is known as the k-Constraint Satisfaction Problem (k-CSP) whereas k-SAT is the restriction of k-CSP to formulas in conjunctive normal form. We choose to keep this notation for consistency with previous results.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 881–892, 2008.

These results provide a complete understanding of the 2-SAT problem: for any $a, b \geq 2$ the (a, b)-SAT problem is in P if $a = b = 2$ and NP-complete otherwise. Regarding the 3-SAT problem, it is known to be NP-complete even if all variables are binary, thereby giving, again, a complete understanding of the problem, i.e., for all $a, b, c \geq 2$, (a, b, c)-SAT is NP-complete.[2]

A closely related problem is MAX-k-SAT in which we are asked for the maximum number of simultaneously satisfiable clauses in a given formula. Clearly, MAX-k-SAT is at least as hard as k-SAT and, in fact, is known to be NP-complete for all $k \geq 2$ (see, e.g., [1]).

An extensive study has been carried out in recent years to understand the behavior of the quantum analogs of MAX-k-SAT and k-SAT, known as the *k-local Hamiltonian* problem and the *quantum k-SAT* problem, respectively.

The k-local Hamiltonian problem. In the k-local Hamiltonian problem, we are given a Hermitian operator H, often referred to as a Hamiltonian, that acts on n qubits and can be written as a sum of Hermitian operators, each acting nontrivially on at most k qubits. We are asked whether H has an eigenvalue less than a, or that all of H's eigenvalues are greater than b, where a and b are two constants.

Notice that MAX-k-SAT is essentially a special case of the k-local Hamiltonian problem: any MAX-k-SAT formula can be written as a Hamiltonian whose lowest eigenvalue is exactly the smallest possible number of unsatisfied clauses in any assignment, and its corresponding eigenvector is the binary assignment that achieves this number. For instance, the formula in (1) can be written as the following sum of projectors,

$$H = |01\rangle\langle 01|_{x_1,x_2} + (|00\rangle\langle 00| + |11\rangle\langle 11|)_{x_1,x_3}.$$

Kitaev has shown [2, Chapter 14] that the 5-local Hamiltonian problem is complete for a complexity class called QMA. This class consists of all promise problems whose membership can be verified efficiently by a quantum circuit when given a quantum witness state. QMA is often regarded as the quantum analog of NP, although it is more precisely described as the quantum analog of the classical class MA.

Kitaev's result has later been improved by Kempe and Regev [3] to the 3-local Hamiltonian problem, and finally by Kitaev, Kempe, and Regev who showed [4] that even the 2-local Hamiltonian problem is QMA-complete. Thus, the k-local Hamiltonian problem is QMA-complete for all $k \geq 2$, in analogy to the behavior of the MAX-k-SAT problem in the classical world. These results provide a complete understanding of the complexity of the k-local Hamiltonian problem.

The quantum k-SAT problem. In contrast to our knowledge regarding the k-local Hamiltonian problem, our understanding of the quantum analog of the k-SAT

[2] We remark that important open questions remain regarding the complexity of other constraint satisfaction problems, most notably the *dichotomy conjecture* formulated by Feder and Vardi in 1993.

problem is far from satisfactory. In the quantum k-SAT problem, defined by Bravyi [5], we are given a set of projectors on a system of n qubits, each acting nontrivially on at most k qubits, and asked whether they have a common groundstate or whether the lowest eigenvalue of their sum is at least $1/\mathrm{poly}(n)$. This problem is essentially equivalent to the k-local Hamiltonian problem, except that in 'yes' instances there must exist a common groundstate of all Hamiltonians (the restriction to projectors is for convenience and is essentially without loss of generality).

Bravyi proved two fundamental results regarding the quantum k-SAT problem. First he showed that the problem can be solved in polynomial time for $k = 2$. His second result is a completeness result for all $k \geq 4$. More precisely, he showed that for all $k \geq 4$, quantum k-SAT is complete for a class known as QMA_1, which is defined similarly to QMA except for the extra restriction that the verifier can be made to accept positive instances with probability 1 (the classical analog of this class, MA_1, is known to be equal to MA). Whether or not the quantum 3-SAT problem, known to be NP-hard, is also QMA_1-complete remains an important and challenging open question.

As in the classical world, it is interesting to analyze the complexity of these problems in the non-binary setting. For instance, one might ask what are the lowest values of a and b such that the quantum (a, b)-SAT problem is still QMA_1-complete, and a similar question regarding 3-particle interactions, i.e., what are the minimal values of a, b, c such that the quantum (a, b, c)-SAT problem is QMA_1-complete. Some progress in this direction has recently been made by Nagaj and Mozes [6] who have shown that quantum (a, b, c)-SAT is QMA_1-complete for all $a \geq 3$, $b \geq 2$ and $c \geq 2$. Additionally, they have claimed that quantum (a, b)-SAT is still QMA_1-complete for $a = 4, b = 9$.

Our Result. In this paper we show that the quantum (a, b)-SAT problem is QMA_1-complete even when $a = 3$ and $b = 5$. We state this as the main theorem of this paper:

Theorem 1. *Quantum* $(3, 5)$*-SAT is* QMA_1*-complete.*

We summarize the known results in Table 1.

Table 1. Best known results for quantum (a, b)-SAT (NP-H stands for NP-hard, and Q1-C stands for QMA_1-complete). Prior to our result, the highlighted entries were only known to be NP-hard.

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	P	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H
3		NP-H	NP-H	**Q1-C**	**Q1-C**	**Q1-C**	**Q1-C**	**Q1-C**	**Q1-C**
4			NP-H	**Q1-C**	**Q1-C**	**Q1-C**	**Q1-C**	Q1-C	Q1-C
5				**Q1-C**	**Q1-C**	**Q1-C**	**Q1-C**	Q1-C	Q1-C
6					**Q1-C**	**Q1-C**	**Q1-C**	Q1-C	Q1-C
7						**Q1-C**	**Q1-C**	Q1-C	Q1-C
8							**Q1-C**	Q1-C	Q1-C
9								Q1-C	Q1-C

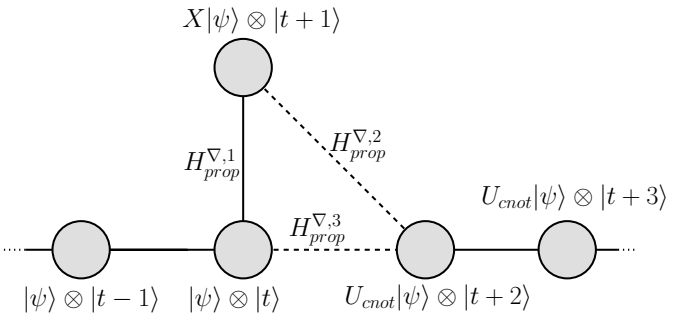


Fig. 1. The Triangle Hamiltonian Construction

The Triangle Construction. The main novel ingredient of our proof is a certain Hamiltonian construction named the *Triangle Hamiltonian*. It allows to verify the application of a 2-qubit CNOT (controlled-NOT) gate without generating explicit interactions between workspace qubits. The only other construction we are aware of that is capable of achieving this is due to Kempe et al. [4]. Unlike their construction, however, our construction is direct and is not based on perturbation techniques, which seem inadequate for proving QMA_1-completeness results.

The Triangle Hamiltonian verifies the application of a CNOT gate by examining separately correct application of the gate on the subspace in which the control qubit is $|0\rangle$ and on the subspace in which the control qubit is $|1\rangle$. Thus, this construction can be easily extended to any arbitrary 2-qubit controlled gate. In Figure 1 we illustrate the main idea of this construction.

For those familiar with previous Hamiltonian constructions, we shall briefly present its essential components. Assuming orthogonal clock states $|t\rangle, |t+1\rangle$, $|t+2\rangle$ the sum of the following Hamiltonians verifies that if $|t\rangle$ is in tensor with an arbitrary workspace state $|\psi\rangle$ then the workspace state in tensor with $|t+2\rangle$ is $U_{cnot}|\psi\rangle$:

$$H^{\nabla,1}_{prop} = |t\rangle\langle t| + |t+1\rangle\langle t+1| - X_{tgt} \otimes |t+1\rangle\langle t| - X_{tgt} \otimes |t\rangle\langle t+1|$$
$$H^{\nabla,2}_{prop} = |1\rangle\langle 1|_{ctl} \otimes (|t+1\rangle - |t+2\rangle)(\langle t+1| - \langle t+2|)$$
$$H^{\nabla,3}_{prop} = |0\rangle\langle 0|_{ctl} \otimes (|t\rangle - |t+2\rangle)(\langle t| - \langle t+2|)$$

where *ctl* is the control qubit and *tgt* is the target qubit. The first Hamiltonian $H^{\nabla,1}_{prop}$ verifies the application of an X gate on the target qubit in the transition from time $|t\rangle$ to time $|t+1\rangle$. Thus, if $|t\rangle$ is in tensor with a workspace state $|\psi\rangle$, then $|t+1\rangle$ is in tensor with $X_{tgt}|\psi\rangle$. The two other Hamiltonians verify that in the subspace in which the control qubit is $|1\rangle$, the state at time $|t+2\rangle$ agrees with that at time $|t+1\rangle$ (i.e. $X_{tgt}|\psi\rangle$) whereas in the subspace in which the control qubit is $|0\rangle$, the state at time $|t+2\rangle$ agrees with that at time $|t\rangle$ (i.e. $|\psi\rangle$).

Applications of the Triangle Construction. We believe that the triangle construction presented in this paper may contribute to progress in similar Hamiltonian-oriented problems, as it decouples in a simple fashion the dependency between computationally-coupled qubits. Indeed, we were recently notified that this construction is being used by Nagaj and Love [7] to prove the completeness of local Hamiltonian problems involving Hamiltonians of a very simple form, namely those composed of only X and Z. Furthermore, we conjecture that the triangle construction may lead to improvements in adiabatic computation (see, e.g., [8]), especially in terms of running time.

Open Questions. Several important open questions remain. It would be interesting to classify the complexity of quantum (a, b)-SAT for all cases marked as NP-hard in Table 1. These problems are known to be NP-hard, but it is unknown whether they are also QMA_1-complete, or whether they lie in some intermediate complexity class. Note in this respect, that the quantum $(4, 2)$-SAT problem is essentially a special case of the quantum 3-SAT problem (i.e., quantum $(2, 2, 2)$-SAT), which is also an open problem.

It would seem natural that the Triangle construction could help resolve the open question regarding the quantum 3-SAT ($(2, 2, 2)$-SAT) problem. Indeed, we have tried to apply it to prove that quantum 3-SAT is QMA_1-complete. Unfortunately, it seems very difficult to obtain a sufficiently strong clock encoding. Nagaj and Mozes [6], who had faced similar difficulties, have even suggested that the quantum 3-SAT problem might not be QMA_1-complete after all.

2 Preliminaries

The classes QMA *and* QMA_1. The class QMA is a class of promise problems. A promise problem is a pair of disjoint sets of strings (L_{yes}, L_{no}) corresponding to the positive and negative instances of the problem. For a given string $x \in L_{yes} \cup L_{no}$ we are asked whether $x \in L_{yes}$ or $x \in L_{no}$.

Definition 1. *A promise problem L is in* QMA *if there exists a polynomial p, and a classical poly-time algorithm A, that for every $x \in \{0,1\}^*$ returns a quantum circuit $V_x = A(x)$ that accepts $p(|x|)$ qubits as input and is allowed to use $p(|x|)$ ancilla qubits, such that*

$$\text{if } x \in L_{yes}, \text{ then } \exists |\eta\rangle \in (\mathbb{C}^2)^{\otimes p(|x|)} \text{ s.t. } Prob(V_x|\eta\rangle) \geq 2/3, \text{ and}$$
$$\text{if } x \in L_{no}, \text{ then } \forall |\eta\rangle \in (\mathbb{C}^2)^{\otimes p(|x|)} \text{ we have } Prob(V_x|\eta\rangle) \leq 1/3$$

where $Prob(V_x|\eta\rangle)$ is the probability that measuring the output qubit after the computation will result in state $|1\rangle$.

It is known that one can replace the constants $2/3, 1/3$ by $1 - \epsilon, \epsilon$ where $\epsilon = \epsilon(|x|) = 2^{-\mathrm{poly}(|x|)}$ without affecting the resulting class.

Definition 2. *A promise problem L is in QMA_1 if there exists a polynomial p, and a classical poly-time algorithm A, that for every $x \in \{0,1\}^*$ returns a*

quantum circuit $V_x = A(x)$ that accepts $p(|x|)$ qubits as input and is allowed to use $p(|x|)$ ancilla qubits, such that

$$\text{if } x \in L_{yes}, \text{ then } \exists |\eta\rangle \in (\mathbb{C}^2)^{\otimes p(|x|)} \text{ s.t. } Prob(V_x|\eta\rangle) = 1, \text{ and}$$
$$\text{if } x \in L_{no}, \text{ then } \forall |\eta\rangle \in (\mathbb{C}^2)^{\otimes p(|x|)} \text{ we have } Prob(V_x|\eta\rangle) \leq 1/3$$

where $Prob(V_x|\eta\rangle)$ is the probability that measuring the output qubit after the computation will result in state $|1\rangle$.

It is known that one can replace the constant $1/3$ by any $2^{-\text{poly}(|x|)}$ without affecting the resulting result. Note that $\mathsf{NP} \subseteq \mathsf{QMA}_1 \subseteq \mathsf{QMA}$.

The Local Hamiltonian and the Quantum SAT problems. We define the k-local Hamiltonian problem, the quantum analog of MAX-k-SAT as follows:

Definition 3. *In the k-local Hamiltonian problem we are given as input $m = \text{poly}(n)$ Hamiltonians $H_1, \ldots, H_m$ on a system of n qubits, each acting nontrivially on at most k qubits and satisfying $0 \leq H_j \leq I$, as well as two numbers a, b such that $b - a = 1/\text{poly}(n)$. We are then asked whether the lowest eigenvalue of $H = \sum_{j=1}^{m} H_j$ is at most a ('yes' instance) or at least b ('no' instance).*

The k-local Hamiltonian problem is known to be in QMA for any constant k [2]. We proceed to define the quantum k-SAT problem, the quantum analog of k-SAT.

Definition 4. *In the quantum k-SAT problem we are given $m = \text{poly}(n)$ projectors $\Pi_1, \ldots, \Pi_m$ on a system of n qubits, each acting nontrivially on at most k qubits. We are asked whether there exists a common groundstate of all projectors, i.e., a state $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$ such that for all $j, \Pi_j|\psi\rangle = 0$ ('yes' instance), or whether for all $|\psi\rangle, \sum_j \langle\psi|\Pi_j|\psi\rangle \geq 1/\text{poly}(n)$ ('no' instance).*

The quantum k-SAT problem is known to be in QMA_1 for any constant k [5].

The particle system. The Hamiltonians in this paper, as in previous papers, operate on a system of particles comprised of a *workspace subsystem* and a *clock subsystem*,

$$\mathcal{H}_{total} = \mathcal{H}_{work} \otimes \mathcal{H}_{clock}.$$

Throughout this paper we use the tensor symbol $\otimes$ to separate the work subsystem (left) from the clock subsystem (right).

3 QMA-Hardness of the Local Hamiltonian Problem

In this section we briefly recall the main components in Kitaev's proof that the local Hamiltonian problem is QMA-complete [2]. Our goal is to take any language in QMA and reduce it to the local Hamiltonian problem. In order to prove this, we will show how to convert any verifying circuit into a local Hamiltonian in a

way that the maximum acceptance probability of the verifying circuit (over all possible witnesses) corresponds to the lowest eigenvalue of the Hamiltonian.

In more detail, let $V = V_x = V_T \cdots V_1$ be a T-step quantum verifying circuit that receives as input an n_w-qubit witness, and uses n_a ancilla qubits, where $n_a + n_w = n = \text{poly}(|x|)$ and $T = \text{poly}(n)$. Kitaev defines three Hamiltonians that operate on a clock register with states $|1\rangle, \ldots, |T+1\rangle$, and a workspace register of n qubits, as follows.

1. A Hamiltonian that checks correct initialization,
$$H_{in}^K = \sum_{k \in ancilla} |1\rangle\langle 1|_k \otimes |1\rangle\langle 1|. \tag{2}$$
2. A Hamiltonian that verifies acceptance of the witness by the circuit,
$$H_{out}^K = |0\rangle\langle 0|_{out} \otimes |T+1\rangle\langle T+1|. \tag{3}$$
3. A Hamiltonian that verifies correct propagation according to the circuit,
$$H_{prop}^K = \sum_{j=1}^{T} H_{prop}^{(K,j)}$$
where
$$\begin{aligned} H_{prop}^{(K,j)} &= \frac{1}{2}(I \otimes |j\rangle - V_j \otimes |j+1\rangle)(I \otimes \langle j| - V_j^\dagger \otimes \langle j+1|) \\ &= \frac{1}{2}(I \otimes |j\rangle\langle j| - V_j \otimes |j+1\rangle\langle j| - V_j^\dagger |j\rangle\langle j+1| + I \otimes |j+1\rangle\langle j+1|). \end{aligned} \tag{4}$$

Kitaev then proved the following for the sum of the three Hamiltonians, $H_{total}^K = H_{in}^K + H_{out}^K + H_{prop}^K$.

Lemma 1. *There are a, b with $b - a \geq 1/\text{poly}(n)$, such that, if V accepts with probability greater than $1 - \epsilon$ for $\epsilon = 2^{-\text{poly}(n)}$ on some input $|\eta, 0\rangle$, then the Hamiltonian H_{total}^K has an eigenvalue smaller than a. Conversely, if the circuit accepts with probability less than ϵ on all inputs $|\eta, 0\rangle$, then all eigenvalues of H_{total}^K are greater than b.*

If we represent the clock register with $\lceil \log(T+1) \rceil$ qubits and assume without loss of generality that V is composed of two-qubit gates, then we obtain that H_{total}^K is $(\lceil \log(T+1) \rceil + 2)$-local. By using a unary representation of the clock register and slightly modifying the Hamiltonians above, Kitaev was able to turn H_{total}^K into a 5-local Hamiltonian, thereby proving that the k-local Hamiltonian problem is QMA-complete for all $k \geq 5$.

4 Proof of Theorem 1

4.1 General

Proof structure. It is easy to see that the quantum $(3,5)$-SAT problem is in QMA_1. For example, one can reduce quantum $(3,5)$-SAT to quantum 5-SAT,

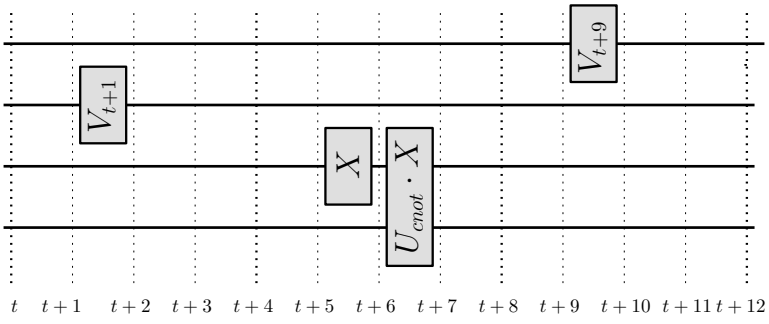


Fig. 2. Three consecutive clusters $j, j+1, j+2$, and their corresponding time points $t = 4(j-1)+1, \ldots, 4(j-1)+4, 4j+1, \ldots, 4j+4, 4(j+1)+1, \ldots, t+12 = 4(j+2)+1$

essentially by replacing each 5-dimensional particle by 3 qubits and each 3-dimensional particle by 2 qubits. Thus, we focus on proving that $(3,5)$-SAT is QMA_1-hard. We prove this along the lines of Kitaev [2], by showing how to reduce any verifying circuit to a Hamiltonian sum, in which each term is a projector acting on two particles of dimensions at most 3 and 5. We begin by describing the behavior of the clock register, and then build upon it to verify correct computation according to the circuit. We conclude by proving that for a 'yes' instance the projectors have a common groundstate, while for a 'no' instance, the lowest eigenvalue of the sum of the projectors is $\Omega(1/\mathrm{poly}(n))$.

The verifying circuit. Let $V = V_x$ denote the verifier circuit for some $x \in \{0,1\}^*$ that acts on an n_w-qubit witness state, and uses n_a ancilla qubits, where $n_a + n_w = n = \mathrm{poly}(|x|)$. We assume without loss of generality that $V = V_{4T} \cdots V_1$ consists of $4T$ gates, arranged in T 'clusters' of size 4 each, where $T = \mathrm{poly}(n)$. For each $j \in \{1, \ldots, T\}$, we assume that the jth cluster (i.e., $V_{4(j-1)+4} \cdots V_{4(j-1)+1}$) is either of the form $I \cdot (U_{cnot} \cdot X) \cdot X \cdot I$, where the X gates act on the target qubit of the U_{cnot} gate, or of the form $I \cdot I \cdot U \cdot I$, where U is some one-qubit gate (see Figure 2). Let $T_0 \subseteq \{1, \ldots, T\}$ denote the set of clusters of the former type, i.e., clusters whose action is a CNOT gate. By definition, for any 'yes' instance there exists a witness that makes V accept with probability 1. Moreover, as mentioned earlier, we can assume that for any 'no' instance and any witness, V accepts with exponentially small probability $\epsilon = 2^{-\mathrm{poly}(|x|)}$.

4.2 Clock Encoding

The clock register is comprised of 5-state particles called *cinquits* and 3-state particles called *qutrits*. For a circuit of T gates, the clock register includes $T+1$ qutrits and T cinquits that appear in an interleaved fashion $q_1, c_1, q_2, c_2, \ldots, q_T, c_T, q_{T+1}$. The possible states for the qutrits are u (the 'unborn' state), a (the 'active' state), and d (the 'dead' state); the possible states for the cinquits are $\overline{u}$ (the 'unborn' state), $\overline{a}_1, \overline{a}_2, \overline{a}_3$, (the 'active' states), and $\overline{d}$ (the 'dead' state).

We define the subspace of legal clock encodings, S_{legal}, of dimension $4T+1$, by defining an orthonormal basis B_{legal} that spans S_{legal}. We use the following notation:

$$
\begin{aligned}
|1\rangle &= |a\ \overline{u}\ u\ \overline{u} \dots u\ \overline{u}\ u\ \overline{u}\ u\ \overline{u}\ u\ \overline{u}\ \dots u\ \overline{u}\ u\rangle \\
&\vdots \\
|4(j-1)+1\rangle &= |d\ \overline{d}\ d\ \overline{d} \dots d\ \overline{d}\ a\ \overline{u}\ \ u\ \overline{u}\ u\ \overline{u}\ \dots u\ \overline{u}\ u\rangle \\
|4(j-1)+2\rangle &= |d\ \overline{d}\ d\ \overline{d} \dots d\ \overline{d}\ d\ \overline{a}_1\ u\ \overline{u}\ u\ \overline{u}\ \dots u\ \overline{u}\ u\rangle \\
|4(j-1)+3\rangle &= |d\ \overline{d}\ d\ \overline{d} \dots d\ \overline{d}\ d\ \overline{a}_2\ u\ \overline{u}\ u\ \overline{u}\ \dots u\ \overline{u}\ u\rangle \\
|4(j-1)+4\rangle &= |\underbrace{d\ \overline{d}\ d\ \overline{d} \dots d\ \overline{d}}_{2(j-1)} d\ \overline{a}_3\ \underbrace{u\ \overline{u}\ u\ \overline{u}\ \dots u\ \overline{u}\ u}_{2(T-j)+1}\rangle \\
&\vdots \\
|4T+1\rangle &= |d\ \overline{d}\ d\ \overline{d} \dots d\ \overline{d}\ d\ \overline{d}\ d\ \overline{d}\ d\ \overline{d}\ \dots \quad d\ \overline{d}\ a\rangle
\end{aligned}
$$

We define B_{legal} as $|1\rangle, \dots, |4T+1\rangle$. In other words, B_{legal} contains all clock registers of the form $d \dots d\ a\ u \dots u$, i.e., a sequence of zero or more dead states, followed by an active state, which is then followed by zero or more unborn states. We set $S_{legal} = \mathrm{span}(B_{legal})$.

For cluster indices $j \in T_0$, the computation takes place during the transition between the clock states $|4(j-1)+2\rangle, |4(j-1)+3\rangle, |4(j-1)+4\rangle$. For cluster indices $j \notin T_0$, the computation takes place during the transition from $|4(j-1)+2\rangle$ to $|4(j-1)+3\rangle$. For both cluster types, the state $|4(j-1)+1\rangle$ is just an intermediate step that transfers the active site between neighboring cinquits, using the intermediate qutrit particle.

We now define a Hamiltonian H_{clock} that verifies a correct clock encoding. First, for each $1 \le k \le T$ we define a Hamiltonian acting on the qutrit-cinquit pair (q_k, c_k),

$$
H^{(k)}_{clock,1} = \left((I - |d\rangle\langle d|) \otimes (I - |\overline{u}\rangle\langle \overline{u}|) + |d\rangle\langle d| \otimes |\overline{u}\rangle\langle \overline{u}|\right)_{q_k, c_k},
$$

and a Hamiltonian acting on the cinquit-qutrit pair (c_k, q_{k+1}),

$$
H^{(k)}_{clock,2} = \left((I - |\overline{d}\rangle\langle \overline{d}|) \otimes (I - |u\rangle\langle u|) + |\overline{d}\rangle\langle \overline{d}| \otimes |u\rangle\langle u|\right)_{c_k, q_{k+1}}.
$$

Moreover, we define $H_{clock,1} = \sum_{k=1}^{T} H^{(k)}_{clock,1}$ and $H_{clock,2} = \sum_{k=1}^{T} H^{(k)}_{clock,2}$. One can verify that the term $H_{clock,1} + H_{clock,2}$ restricts any computational-basis clock state according to the following rules:

1. An unborn or active particle may only be followed by an unborn particle;
2. A dead particle may not be followed by an unborn particle.

We also define the Hamiltonian

$$
H_{clock,3} = \left(|u\rangle\langle u|\right)_{q_1} + \left(|d\rangle\langle d|\right)_{q_{T+1}},
$$

which checks that the first qutrit is not unborn, and that the last qutrit is not dead. This serves to rule out the all-unborn state $uu\ldots u$ and the all-dead state $dd\ldots d$. Finally, we define $H_{clock} = H_{clock,1} + H_{clock,2} + H_{clock,3}$. The proof of the following easy claim is omitted.

Claim. H_{clock} is a positive semidefinite operator whose zero groundspace is S_{legal}.

4.3 Verifying Input, Output and Propagation

Using the definitions above we write Hamiltonians that verify correct propagation of the computation steps of V.

1. For $j \in T_0$,
   (a) $H_{prop}^{(4(j-1)+2)} = \frac{1}{2}(I \otimes |\overline{a}_1\rangle_{c_j} - X_{tgt(j)} \otimes |\overline{a}_2\rangle_{c_j})(I \otimes \langle\overline{a}_1|_{c_j} - X_{tgt(j)} \otimes \langle\overline{a}_2|_{c_j})$
   (b) $H_{prop}^{(4(j-1)+3)} = \frac{1}{2}(|1\rangle\langle 1|_{ctl(j)} \otimes (|\overline{a}_2\rangle - |\overline{a}_3\rangle)(\langle\overline{a}_2| - \langle\overline{a}_3|)_{c_j} + |0\rangle\langle 0|_{ctl(j)} \otimes (|\overline{a}_1\rangle - |\overline{a}_3\rangle)(\langle\overline{a}_1| - \langle\overline{a}_3|)_{c_j})$
   where $ctl(j)$ and $tgt(j)$ are the control and target qubits of the CNOT gate of the j-th cluster. These two Hamiltonians correspond to $H_{prop}^{\nabla,1}$ and $H_{prop}^{\nabla,2} + H_{prop}^{\nabla,3}$ in the notation of Figure 1.
2. For $j \notin T_0$
   (a) $H_{prop}^{(4(j-1)+2)} = \frac{1}{2}(I \otimes |\overline{a}_1\rangle_{c_j} - V_{4(j-1)+2} \otimes |\overline{a}_2\rangle_{c_j})(I \otimes \langle\overline{a}_1|_{c_j} - V_{4(j-1)+2}^{\dagger} \otimes \langle\overline{a}_2|_{c_j})$
   (b) $H_{prop}^{(4(j-1)+3)} = \frac{1}{2}(|\overline{a}_2\rangle - |\overline{a}_3\rangle)(\langle\overline{a}_2| - \langle\overline{a}_3|)_{c_j}$
   where $V_{4(j-1)+2}$ is the one-qubit gate of cluster j.
3. For $1 \leq j \leq T$
   (a) $H_{prop}^{(4(j-1)+1)} = \frac{1}{2}((|a,\overline{u}\rangle - |d,\overline{a}_1\rangle)(\langle a,\overline{u}| - \langle d,\overline{a}_1|)_{q_j,c_j})$
   (b) $H_{prop}^{(4(j-1)+4)} = \frac{1}{2}((|\overline{a}_3,u\rangle - |\overline{d},a\rangle)(\langle\overline{a}_3,u| - \langle\overline{d},a|)_{c_j,q_{j+1}})$

We thus define the complete propagation Hamiltonian as the sum

$$H_{prop} = \sum_{t=1}^{4T} H_{prop}^{(t)}.$$

In addition to the Hamiltonians H_{clock} and H_{prop} defined above we define two Hamiltonians that check correct ancilla initialization and final acceptance by C:

1. $H_{in} = \sum_{k\in ancilla} |1\rangle\langle 1|_k \otimes |a\rangle\langle a|_{q_1}$
2. $H_{out} = |0\rangle\langle 0|_{out} \otimes |a\rangle\langle a|_{q_{T+1}}$.

Finally we define

$$H_{total} = H_{clock} + c \cdot H_{prop} + H_{in} + H_{out}$$

as the input to the quantum 2-SAT problem, where c is some large enough positive integer constant.[3] Note that H_{total} can be written as a sum of projectors, each acting on at most two particles, whose dimensions are at most three and five. Also, it can be verified that none of the Hamiltonians in the sum above induces transitions from legal to illegal clock states, and therefore the subspace S_{legal} is invariant under $c \cdot H_{prop}, H_{in}, H_{out}$.

4.4 Proof of Completeness

Our goal is to show that if there is a witness state $|\psi\rangle$ that is accepted by the circuit V with probability 1, then H_{total} has an eigenvector with eigenvalue zero. We prove this by considering the computational history of V,

$$|\eta\rangle = \frac{1}{\sqrt{4T+1}} \sum_{t=1}^{4T+1} |\eta_t\rangle \otimes |t\rangle,$$

where $|\eta_t\rangle = V_{t-1} \cdots V_1 |\psi\rangle \otimes |0\rangle^{\otimes n_a}$. It is straightforward to verify that $|\eta\rangle$ is a zero eigenvector of H_{clock}, H_{prop} and H_{in}. According to the definition of QMA_1, the circuit V accepts the witness $|\psi\rangle$ with probability 1, and therefore the state $|\eta_{4T+1}\rangle$, which is equal to $V_{4T} \cdots V_1 |\psi\rangle \otimes |0\rangle^{\otimes n_a}$ has $|1\rangle$ at its output qubit. Therefore the state $|\eta_{4T+1}\rangle \otimes |4T+1\rangle$ is a zero eigenvector of H_{out}, and we conclude that $|\eta\rangle$ is a zero eigenvector of H_{total} as required.

4.5 Proof of Soundness

In this section we show that if no witness is accepted by the circuit V with probability greater than ϵ, for $\epsilon = 2^{-\mathrm{poly}(|x|)}$, then the smallest eigenvalue of H_{total} is at least $1/\mathrm{poly}(n)$.

Since S_{legal} is invariant under H_{total}, and any state in the illegal clock subspace $S_{legal}^{\perp}$ has energy at least 1 due to H_{clock}, we can disregard $S_{legal}^{\perp}$ altogether and restrict the analysis to S_{legal}. Thus, our goal is to prove that inside S_{legal} the lowest eigenvalue of $c \cdot H_{prop} + H_{in} + H_{out}$ is at least $1/\mathrm{poly}(n)$. The restriction of these Hamiltonians to S_{legal} is given by the following. For all $t = 4(j-1)+3$ where $j \in T_0$,

$$H_{prop}^{(t)}|_{S_{legal}} = \frac{1}{2}(|1\rangle\langle 1|_{ctl(j)} \otimes (|t\rangle - |t+1\rangle)(\langle t| - \langle t+1|) + |0\rangle\langle 0|_{ctl(j)} \otimes (|t-1\rangle - |t+1\rangle)(\langle t-1| - \langle t+1|)). \quad (5)$$

For all other values of $t \in \{1, \ldots, 4T+1\}$,

$$H_{prop}^{(t)}|_{S_{legal}} = \frac{1}{2}(I \otimes |t\rangle - V_t \otimes |t+1\rangle)(I \otimes \langle t| - V_t^{\dagger} \otimes \langle t+1|) \quad (6)$$

[3] Actually, the construction also works with $c = 1$, but our choice slightly simplifies the argument.

and finally,

$$H_{in}|_{S_{legal}} = \sum_{k \in ancilla} |1\rangle\langle 1|_k \otimes |1\rangle\langle 1| \tag{7}$$

$$H_{out}|_{S_{legal}} = |0\rangle\langle 0|_{out} \otimes |4T+1\rangle\langle 4T+1|. \tag{8}$$

By slight abuse of notation, we shall identify from now on each Hamiltonian with its restriction to S_{legal}.

Let $H^K_{in}, H^K_{out}, H^K_{prop}$ be Kitaev's Hamiltonians that correspond to the verifier $V = V_{4T} \cdots V_1$ (see Section 3). By comparing equations (2)-(3) to (7)-(8), it can be verified that both H_{in} and H_{out} are identical to Kitaev's H^K_{in} and H^K_{out}, respectively.

Additionally, since a 'no' instance is accepted with probability at most $2^{-\mathrm{poly}(n)}$, we obtain by Lemma 1 that the lowest eigenvalue of $H^K_{prop}+H^K_{in}+H^K_{out}$ is $1/\mathrm{poly}(n)$. Thus, a sufficient condition for proving the lower bound on the smallest eigenvalue of $c \cdot H_{prop} + H_{in} + H_{out}$ is that $c \cdot H_{prop} \geq H^K_{prop}$ for some large enough positive constant c. We show that this condition indeed holds in the following lemma, which concludes the proof of soundness. The proof is omitted due to lack of space.

Lemma 2. *There exists a positive constant c such that $c \cdot H_{prop} \geq H^K_{prop}$.*

Acknowledgments. We thank Daniel Nagaj for useful discussions. We also thank the anonymous referees for useful comments. Part of this work was done while the second author was attending the Lorentz Center workshop on "Computational Complexity of Quantum Hamiltonian Systems" and he would like to thank the organizers for inviting him.

References

1. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Longman, Amsterdam (1995)
2. Kitaev, A.Y., Shen, A.H., Vyalyi, M.N.: Classical and quantum computation. Graduate Studies in Mathematics, vol. 47. AMS, Providence (2002)
3. Kempe, J., Regev, O.: 3-local Hamiltonian is QMA-complete. Quantum Information & Computation 3(3), 258–264 (2003)
4. Kempe, J., Kitaev, A., Regev, O.: The complexity of the local Hamiltonian problem. SIAM Journal of Computing 35(5), 1070–1097 (2006)
5. Bravyi, S.: Efficient algorithm for a quantum analogue of 2-SAT. ArXiv Quantum Physics e-prints (2006)
6. Nagaj, D., Mozes, S.: New construction for a QMA complete three-local Hamiltonian. Journal of Mathematical Physics 48, 2104 (2007)
7. Nagaj, D., Love, P.: (in preparation, 2007)
8. Aharonov, D., van Dam, W., Kempe, J., Landau, Z., Lloyd, S., Regev, O.: Adiabatic quantum computation is equivalent to standard quantum computation. SIAM Journal on Computing 37(1), 166–194 (2007)

Author Index