Volker Diekert
Bruno Durand (Eds.)

LNCS 3404

STACS 2005

22nd Annual Symposium
on Theoretical Aspects of Computer Science
Stuttgart, Germany, February 2005, Proceedings

Lecture Notes in Computer Science 3404

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

Editorial Board

Volker Diekert Bruno Durand (Eds.)

STACS 2005

22nd Annual Symposium
on Theoretical Aspects of Computer Science
Stuttgart, Germany, February 24-26, 2005
Proceedings

Springer

Volume Editors

Volker Diekert
Universität Stuttgart, FMI
Universitätsstr. 38, 70569 Stuttgart, Germany
E-mail: diekert@informatik.uni-stuttgart.de

Bruno Durand
LIF - CMI
39, rue Joliot Curie, 13453 Marseille Cedex 13, France
E-mail: bruno.durand@lif.univ-mrs.fr

Library of Congress Control Number: 2005920531

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

ISSN 0302-9743
ISBN 3-540-24998-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11394556 06/3142 5 4 3 2 1 0

Preface

The Symposium on Theoretical Aspects of Computer Science (STACS) is alternately held in France and in Germany. The conference of February 24–26, 2005 in Stuttgart was the twenty-second in this series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), and Montpellier (2004).

The interest in STACS has been increasing continuously during recent years and has turned it into a leading conference in theoretical computer science. The call for papers for STACS 2005 led to 217 submissions from all over the world.

The 17 members of the Program Committee met in Stuttgart for two days at the end of October 2004 to select the contributions to be presented at the conference. Thanks are due to the committee as well as to all the external referees for their valuable work put into the reviewing process. Just 54 papers (i.e., 25% of the submissions) could be accepted, thus guaranteeing the very high scientific quality of the conference. Moreover, this strict selection enabled us to keep the conference in its standard format with only two parallel sessions.

We would like to thank the three invited speakers Manindra Agrawal (Singapore), Mireille Bousquet-Mélou (Bordeaux), and Uwe Schöning (Ulm) for presenting lectures and for their contributions to the proceedings.

Finally we thank the local organizing committee and all members of the Institute of Formal Methods in Computer Science for their help. In particular we thank Holger Austinat, Heike Photien, and Horst Prote for their great efforts. Special thanks are due to Andrei Voronkov for his PC-expert software *EasyChair* and for continuously supporting us.

Financial support for the conference was provided by the German research foundation DFG and by the University of Stuttgart. The Informatik Forum Stuttgart (infos e.V.) helped in the management of the conference.

February 2005

Volker Diekert
Bruno Durand

The Program Committee at Work

(October 29/30, 2004)

Brigitte Vallée Rolf Klein
Claude Kirchner

Jop Sibeyn Matthew Hennessy
Friedhelm Meyer auf der Heide

Gilles Schaeffer Dietrich Kuske Giuseppe Persiano
Pascal Weil Monika Henzinger

Gerhard Woeginger Eric Allender
Hendrik Jan Hoogeboom

Peter Høyer Volker Diekert
Bruno Durand

Organization

STACS 2005 was organized by the Institute of Formal Methods in Computer Science at the University of Stuttgart and by infos e.V. under the auspices of the Special Interest Group for Theoretical Computer Science of the Gesellschaft für Informatik (GI).

Program Committee

Eric Allender (Rutgers State University, New Brunswick)
Volker Diekert (Universität Stuttgart), Co-chair
Bruno Durand (Université de Provence, Marseille), Co-chair
Matthew Hennessy (University of Sussex)
Monika Henzinger (École Polytechnique, Lausanne)
Hendrik Jan Hoogeboom (Universiteit Leiden)
Peter Høyer (University of Calgary)
Claude Kirchner (LORIA, Nancy)
Rolf Klein (Universität Bonn)
Dietrich Kuske (Universität Dresden)
Friedhelm Meyer auf der Heide (Universität Paderborn)
Giuseppe Persiano (Università di Salerno)
Gilles Schaeffer (École Polytechnique, Palaiseau)
Jop Sibeyn (Universität Halle)
Brigitte Vallée (Université de Caen)
Pascal Weil (Université Bordeaux 1)
Gerhard Woeginger (Universiteit Eindhoven)

Organizing Committee

Volker Claus	Javier Esparza	Michael Matthiesen
Volker Diekert	Ulrich Hertrampf	Holger Petersen

Software for the Program Committee Work

EasyChair by Andrei Voronkov (University of Manchester)
http://www.easychair.org

Referees

Rudolf Ahlswede	Adrianna Alexander	Andris Ambainis
Ali Akhavi	Noga Alon	Klaus Ambos-Spies

Pasquale Ambrosio
Matthew Andrews
Elliot Anshelevich
Aaron Archer
Carlos Areces
André Arnold
Eugène Asarin
Vincenzo Auletta
Nikhil Bansal
Patrick Bas
Ulrike Baumann
Marie-Pierre Béal
Danièle Beauquier
Luca Becchetti
Emmanuel Beffara
Rene Beier
Andras A. Benczur
Michael Ben-Or
Anne Bergeron
Jean Berstel
Marcin Bienkowski
Jean-Camille Birget
Bruno Blanchet
Norbert Blum
Achim Blumensath
Carlo Blundo
Luc Boasson
Hans L. Bodlaender
Mikołaj Bojańczyk
Paola Bonizzoni
Olaf Bonorden
Henning Bordihn
Alan Borodin
Alin Bostan
Jérémie Bourdon
Olivier Bournez
Patricia Bouyer
Julian Bradfield
Robert Brijder
Hervé Brönnimann
Anne Brüggemann-Klein
Véronique Bruyère
Harry Buhrman
Andrei Bulatov
Peter Bürgisser
Julien Cassaigne
Julien Cervelle
Rohit Chada
Krishna Chatterjee
Jingchao Chen
Victor Chepoi
Yannick Chevalier
Andrew Childs
Janka Chlebikova
Bogdan Chlebus
Christian Choffrut
Ferdinando Cicalese
Julien Clément
Loek Cleophas
Johanne Cohen
Loïc Colson
Robert Cori
Fabien Coulon
Jean-Michel Couvreur
Claude Crépeau
Artur Czumaj
Paolo D'Arco
Guillaume Dabosville
Elias Dahlhaus
Benoit Daireaux
Valentina Damerow
Philippe Darondeau
Jürgen Dassow
Anuj Dawar
Frank Dehne
Marianne Delorme
François Denis
Roberto De Prisco
Ronald de Wolf
Walter Didimo
Martin Dietzfelbinger
Clare Dixon
Gilles Dowek
Rod Downey
Jean-Christophe Dubacq
Devdatt Dubhashi
Guillaume Dufay
Arnaud Durand
Jérôme Durand-Lose
Thomas Eiter
Joost Engelfriet
David Eppstein
Leah Epstein
Thomas Erlebach
Zoltán Ésik
Rolf Fagerberg
Hongbing Fan
Lene Monrad Favrholdt
Henning Fernau
Alessandro Ferrante
Michael J. Fischer
Melvin Fitting
Philippe Flajolet
Michele Flammini
Rudolf Fleischer
Fedor Fomin
Enrico Formenti
Lance Fortnow
Dimitris Fotakis
Gereon Frahling
Pierluigi Frisco
David Gamarnik
Leszek A. Gasieniec
Paul Gastin
Cyril Gavoille
Sukumar Ghosh
Dora Giammarresi
Raffaele Giancarlo
Christian Glaßer
Emmanuel Godard
Ashish Goel
Joel Goossens
Aline Gouget
Erich Grädel
Vladimir Grebinski
Frederic Green
Martin Grohe
Clemens Gröpl
Roberto Grossi
Jozef Gruska
Jens Gustedt
Torben Hagerup
Nir Halman
Horst W. Hamacher
Tero Harju
Prahladh Harsha

Pinar Heggernes
Tom Henzinger
Miki Hermann
Ulrich Hertrampf
Andreas Herzig
William Hesse
Lisa Higham
Denis Hirshfeldt
John Hitchcock
Thomas Hofmeister
Markus Holzer
Juraj Hromkovič
Falk Hüffner
Cor Hurkens
Samuel Hym
David Ilcinkas
Lucian Ilie
Neil Immerman
Sandy Irani
Yuval Ishai
Gabor Ivanyos
Bill Jackson
Riko Jacob
David Pokrass Jacobs
David Janin
Klaus Jansen
Emmanuel Jeandel
Emmanuel Jeannot
Mark Jerrum
Mohamed Kaaniche
Frank Kammer
Juhani Karhumäki
Jarkko Kari
Marek Karpinski
Jonathan Katz
Jordan Kerenidis
Lutz Kettner
Bakhadyr Khoussainov
Samir Khuller
Andreas Klappenecker
Jetty Kleijn
Bettina Klinz
Adam Klivans
Daniel Kobler
Roman Kolpakov
Walter A. Kosters
Daniel Král'
Dieter Kratsch
Danny Krizanc
Andrei Krokhin
Piotr Krysta
Gregory Kucherov
Manfred Kufleitner
Michal Kunc
Martin Kutrib
Gregory Lafitte
Martin Lange
Elmar Langetepe
Sophie Laplante
François Laroussinie
Salvatore La Torre
Michel Latteux
Reinhard Laue
Jean-Marie Le Bars
Grégoire Lecerf
Martin Leucker
Christos Levcopoulos
Asaf Levin
Loick Lhote
Christian Liebchen
Yuri Lifshits
Luigi Liquori
Martin Löhnertz
Sylvain Lombardy
Alexandro López-Ortiz
Tamas Lukovszki
Olivier Ly
Frederic Magniez
Meena Mahajan
Anirban Mahanti
Jean Mairesse
Oded Maler
Sebastian Maneth
Yannis Mannoussakis
Stuart Margolis
Jean-Yves Marion
Charles Martel
Conrado Martínez
Barbara Masucci
Jacques Mazoyer
Catherine McCartin
Ross McConnell
Pierre McKenzie
Kurt Mehlhorn
Stephan Merz
Yves Métivier
Ulrich Meyer
Filippo Mignosi
Peter Bro Miltersen
Manuela Montangero
Cris Moore
François Morain
Malika More
Mohamed Mosbah
Anca Muscholl
Muthu Muthukrishnan
N.S. Narayanaswamy
Ashwin Nayak
Ilan Newman
Pierre Nicodème
David Nicol
Rolf Niedermeier
Noam Nisan
Ryan O'Donnell
Nicolas Ollinger
Harold Ollivier
Ayoub Otmani
Martin Otto
Rasmus Pagh
Christophe Papazian
Domenico Parente
Mimmo Parente
Gennaro Parlato
Tomi A. Pasanen
Hélène Paugam-Moisy
Andrzej Pelc
David Peleg
Rudi Pendavingh
Paolo Penna
Holger Petersen
Jordi Petit
Ion Petre
Jean-Eric Pin
Dennis Pixton
Andrei Popescu

Natacha Portier
Dominique Poulalhon
Guido Proietti
Mathieu Raffinot
Silvio Ranise
Julian Rathke
Oded Regev
Mireille Régnier
Horst Reichel
Wolfgang Reisig
Bernhard Reus
Yossi Richter
Michel Rigo
John Michael Robson
Manuel Rode
Andrei Romashchenko
Adi Rosen
Andrei Rumyantsev
Michael Rusinowitch
Jörg R. Sack
Jacques Sakarovitch
Kai Salomaa
Miklos Santha
Luigi Santocanale
Martin Sauerhoff
Carla D. Savage
Christian Scheideler
Christian Schindelhauer
Werner Schindler
Karl Schlechta
Markus Schmidt
Henning Schnoor
Uwe Schöning
Eric Schost
Lutz Schröder
Stefan Schwoon
Detlef Seese
Raimund Seidel
Victor Selivanov
Pranab Sen
Géraud Sénizergues
Maria Serna
Paul Sernine
Wendelin Serwe
Jiří Sgall
Alexander Shen
Amir Shpilka
Sandeep K. Shukla
Riccardo Silvestri
Martin Skutella
Christian Sohler
Greg Sorkin
Robert Spalek
Jeremy Sproston
Ludwig Staiger
Frank Stephan
Howard Straubing
Martin Strauss
Madhu Sudan
Hung-Min Sun
Firas Swidan
Mario Szegedy
Alain Tapp
Jan Arne Telle
Véronique Terrier
Guillaume Theyssier
P.S. Thiagarajan
Robin Thomas
Siegbert Tiga
Jean-Pierre Tillich
Cesare Tinelli
Hans Raj Tiwary
Andrea Torsello
Luca Trevisan
Philippas Tsigas
Vinodchandran Variyam
Yann Vaxès
Vlatko Vedral
Ivan Vegner
Carmine Ventre
Nikolai Vereshchagin
Ivan Visconti
Paul Vitányi
Berthold Vöcking
Heribert Vollmer
Sergei Vorobyov
Timothy Walsh
Igor Walukiewicz
Rolf Wanka
Ingo Wegener
Matthias Westermann
Tom Wexler
Avi Wigderson
Thomas Wilke
Erik Winfree
Guohua Wu
Hiroaki Yamamoto
Ching-Nung Yang
Shaofa Yang
Nobuko Yoshida
Mariette Yvinec
Martin Zachariasen
Norbert Zeh
Marc Zeitoun
Xizhong Zheng
Martin Ziegler
Alexandre Zvonkine

Sponsoring Institutions

Deutsche Forschungsgemeinschaft
DFG

infos e. V.
Informatik-Forum Stuttgart

Universität Stuttgart

Table of Contents

Invited Talks

Session 1A

Session 1B

Session 2A

Session 2B

Session 3A

Session 3B

Session 4A

Session 4B

Session 5A

Session 5B

Session 6A

Session 6B

Session 7A

Session 7B

Session 8A

Session 8B

Session 9A

Session 9B

Session 10A

Session 10B

Automorphisms of Finite Rings and Applications to Complexity of Problems

Manindra Agrawal and Nitin Saxena

National University of Singapore**
{agarwal, nitinsax}@comp.nus.edu.sg

1 Introduction

In mathematics, automorphisms of algebraic structures play an important role. Automorphisms capture the symmetries inherent in the structures and many important results have been proved by analyzing the automorphism group of the structure. For example, Galois characterized degree five univariate polynomials f over rationals whose roots can be expressed using *radicals* (using addition, subtraction, multiplication, division and taking roots) via the structure of automorphism group of the splitting field of f. In computer science too, automorphisms have played a useful role in our understanding of the complexity of many algebraic problems. From a computer science perspective, perhaps the most important structure is that of finite rings. This is because a number of algebraic problems efficiently reduce to questions about automorphisms and isomorphisms of finite rings. In this paper, we collect several examples of this from the literature as well as providing some new and interesting connections.

As discussed in section 2, finite rings can be represented in several ways. We will be primarily interested in the *basis* representation where the ring is specified by its basis under addition. For this representation, the complexity of deciding most of the questions about the automorphisms and isomorphisms is in $\mathrm{FP}^{\mathrm{AM}\cap\mathrm{coAM}}$ [KS04]. For example, *finding ring automorphism* (find a non-trivial automorphism of a ring), *automorphism counting problem* (count the number of automorphisms of a ring), *ring isomorphism problem* (decide if two rings are isomorphic), *finding ring isomorphism* (find an isomorphism between two rings). Also, *ring automorphism problem* (decide if a ring has a non-trivial automorphism) is in P [KS04]. In addition, a number of problems can be reduced to answering these questions. Some of them are:

Primality Testing. Fermat's Little Theorem states that the map $a \mapsto a^n$ is the trivial automorphism in Z_n if n is prime. Although this property is not strong enough to decide primality, the recent deterministic primality test [AKS04] generalizes this to the property that the map is an automorphism in the ring $Z_n[Y]/(Y^r - 1)$ for a suitable r iff n is prime. Further, they prove that it is

** On leave from Indian Institute of Technology, Kanpur.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 1–17, 2005.

enough to test the correctness of the map at a "few" elements to guarantee that it is indeed an automorphism.

Polynomial Factorization. Factoring univariate polynomials over finite fields uses automorphisms in a number of ways [LN86, vzGG99]. It is used to split the input polynomial into factors with each one being square-free and composed only of same degree irreducible factors. Then to transform the problem of factoring polynomial with equal degree irreducible factors to that of root finding. And finally, in finding the roots of the polynomial in the field (this step is randomized while the others are deterministic polynomial-time).

Integer Factorization. Two of the fastest known algorithms for factoring integers, *Quadratic sieve* [Pom84] and *Number Field sieve* [LLMP90], essentially aim to find a non-obvious automorphism of the ring $Z_n[Y]/(Y^2-1)$. Besides, recently [KS04] have shown that integer factorization can be reduced to (1) automorphism counting for ring $Z_n[Y]/(Y^2)$, (2) finding automorphism of the ring $Z_n[Y]/(f(Y))$ where f is a degree three polynomial, (3) finding isomorphism between rings $Z_n[Y]/(Y^2-1)$ and $Z_n[Y]/(Y^2-a^2)$ where $a \in Z_n$.

Graph Isomorphism. Again, [KS04] show this problem reduces to ring isomorphism problem for rings of the form $Z_{p^3}[Y_1,\ldots,Y_n]/\mathcal{I}$ where p is an odd prime and ideal $\mathcal{I}$ has degree two and three polynomials. Here, we improve this result to the rings with any prime characteristic. As the isomorphism problems for a number of structures reduce to Graph Isomorphism (e.g., Group Isomorphism), this shows that all these problems reduce to ring isomorphism and counting automorphisms of a ring (it can be shown easily that ring isomorphism problem reduces to counting automorphism in a ring [KS04]).

Polynomial Equivalence. Two polynomials $p(x_1,\cdots,x_n)$ and $q(x_1,\ldots,x_n)$ over field F are said to be *equivalent* if there is an invertible linear transformation T, $T(x_i)=\sum_{j=1}^{n} t_{i,j}x_j$, $t_{i,j} \in F$, such that $p(T(x_1),\ldots,T(x_n)) = q(x_1,\ldots,x_n)$.[1] This is a well studied problem: we know a lot about the structure of equivalent polynomials when both p and q are *quadratic forms* (homogeneous degree two polynomials) resulting in a polynomial time algorithm for testing their equivalence (Witt's equivalence theorem, see, e.g., [Lan93]). The structure of *cubic forms* (homogeneous degree three polynomials) is less understood though. There is also a cryptosystem based on the difficulty of deciding equivalence between a collection of degree three polynomials [Pat96]. In [Thi98], it was shown that polynomial equivalence problem is in NP $\cap$ coAM and Graph Isomorphism reduces to *polynomial isomorphism* problem where we require T to be a permutation.

Here, we show that the ring isomorphism problem over finite fields reduces to *cubic polynomial equivalence*. We prove a partial converse as well: deciding equivalence of homogeneous degree k polynomials with n variables over field F_q such that $(k, q-1) = 1$, reduces to ring isomorphism problem in time

[1] In some literature, p and q are said to be equivalent if $p = q$ for all elements in F^n.

$n^{O(k)}$. This shows that (1) equivalence for homogeneous constant degree polynomials (for certain degrees) can be efficiently reduced to equivalence for degree three polynomials, and (2) Graph Isomorphism reduces to equivalence for degree three polynomials. In fact, we show that Graph Isomorphism can even be reduced to cubic form equivalence. This explains, at least partly, why cubic form equivalence has been hard to analyze.

The organization of the remaining paper is as follows. The next section discusses the various representations of the rings and their morphisms. Sections 3 to 7 discuss applications of ring automorphisms and isomorphisms in the order outlined above. The last section lists some open questions.

2 Representations of Rings and Automorphisms

We will consider finite rings with identity. Any such ring R can be represented in multiple ways. We discuss three important representations.

Table Representation

The simplest representation is to list all the elements of the ring and their addition and multiplication tables. This representation has size $n = O(|R|^2)$ where $|R|$ is the number of elements of the ring. This is a highly redundant representation and the problem of finding automorphisms or isomorphisms can be solved in $n^{O(\log n)}$ time since any minimal set of generators for the additive group has size $O(\log n)$.

Basis Representation

This representation is specified by a set of generators of the additive group of R. Let n be the *characteristic* of the ring. Then the additive group $(R, +)$ can be expressed as the direct sum $\oplus_{i=1}^{m} Z_{n_i} b_i$ where b_1, ..., b_m are elements of R and $n_i \mid n$ for each i. The elements b_1, ..., b_m are called *basis elements* for $(R, +)$. Therefore, the ring R can be represented as $(n_1, \ldots, n_m, A_1, \ldots, A_m)$ where matrix $A_i = (a_{i,j,k})$ describes the effect of multiplication on b_i, viz., $b_i \cdot b_j = \sum_{k=1}^{m} a_{i,j,k} b_k$, $a_{i,j,k} \in Z_{n_k}$. The size of this representation is $O(m^3)$. This, in general, is exponentially smaller than the size of the ring $|R| = \prod_{i=1}^{m} n_i$. For example, the ring Z_n (it has only one basis element).

The problem of finding automorphisms or isomorphisms becomes harder for this representation. As [KS04] show, these problems belong to the complexity class $\mathrm{FP}^{\mathrm{AM} \cap \mathrm{coAM}}$ and are at least as hard as factoring integers and—in the case of finding isomorphisms—solving graph isomorphism.

Polynomial Representation

A third, and even more compact, representation of R is obtained by starting with the basis representation and then selecting the smallest set of b_is, say b_1,

..., b_m such that the remaining b_is can be expressed as polynomials in b_1, ..., b_m. The representation can be specified by the m basis elements and generators of the ideal of polynomials satisfied by these. Each polynomial is specified by an arithmetic circuit.

The ring can be written as:

$$R = Z_n[Y_1, Y_2, \ldots, Y_m]/(f_1(Y_1, \ldots, Y_m), \ldots, f_k(Y_1, \ldots, Y_m))$$

where Y_1, ..., Y_m are basis elements and $(f_1(Y_1, \ldots, Y_m), \ldots, f_k(Y_1, \ldots, Y_m))$ is the ideal generated by the polynomials f_1, ..., f_k describing all polynomials satisfied by Y_1, ..., Y_m.[2] Often, this representation is exponentially more succinct that the previous one. For example, consider the ring $Z_2[Y_1, \ldots, Y_m]/(Y_1^2, Y_2^2, \ldots, Y_m^2)$. This ring has 2^m basis elements and so the basis representation would require $\Omega(2^{3m})$ space.

The problem of finding automorphisms or isomorphisms is even harder for this representation:

Theorem 1. *Ring automorphism for polynomial representation is NP-hard and ring isomorphism problem is coNP-hard.*

Proof. To prove NP-hardness of ring automorphism problem, we reduce 3SAT to it. Let F be a 3CNF boolean formula over n variables, $F = \wedge_{i=1}^{m} c_i$. Let $\hat{F} = \prod_{i=1}^{m} \hat{c}_i$ and $\hat{c}_i = 1 - (1 - x_{i_1}) \cdot x_{i_2} \cdot (1 - x_{i_3})$ where $c_i = x_{i_1} \vee \bar{x}_{i_2} \vee x_{i_3}$. It is easy to verify that F is unsatisfiable iff $\hat{F}(x_1, \ldots, x_n) \in (x_1^2 - x_1, \ldots, x_n^2 - x_n)$. Let ring

$$R = F_2[Y_1, \ldots, Y_n]/(1 + \hat{F}(Y_1, \ldots, Y_n), \{Y_i^2 - Y_i\}_{1 \le i \le n}).$$

It follows that R is a trivial ring iff formula F is unsatisfiable. So ring $R \oplus R$ has a non-trivial automorphism iff F is satisfiable.

For hardness of ring isomorphism problem, simply note that ring R is isomorphic to trivial ring $\{0\}$ iff F is unsatisfiable.

So the table representation is too verbose while the polynomial representation is too compact. In view of this, we will restrict ourselves to the basis representation for the rings. The rings that we will consider are all commutative with a basis that has all basis elements of the same additive order. In addition, their polynomial representation is of similar size to the basis representation and so, for clarity of exposition, we will use the polynomial representation to express our rings.

Representation of Automorphisms and Isomorphisms

An *automorphism* ϕ of ring R is a one-one and onto map, $\phi : R \mapsto R$ such that for all $x, y \in R$, $\phi(x + y) = \phi(x) + \phi(y)$ and $\phi(x \cdot y) = \phi(x) \cdot \phi(y)$.

[2] Throughout the paper, we use lower case letters, e.g., x, y for free variables (as in polynomial $p(x, y) = x^2 - 2y$) and upper case letters, e.g., X, Y for bound variables (as in the ring $Z_n[X, Y]/(X^2 - 2Y, Y^2)$).

An *isomorphism* between two rings R_1 and R_2 is a one-one and onto map ϕ, $\phi : R_1 \mapsto R_2$ such that for all $x, y \in R_1$, $\phi(x + y) = \phi(x) + \phi(y)$ and $\phi(x \cdot y) = \phi(x) \cdot \phi(y)$.

Their representations will depend on the representation chosen for the rings. For basis representation, an automorphism (and isomorphism) will be represented as a linear transformation mapping basis elements. Thus, it corresponds to an invertible matrix of dimension n where n is the number of basis elements.

For polynomial representation, say $R = Z_n[Y_1, \ldots, Y_t]/\mathcal{I}$, an automorphism (or isomorphism) ϕ will be specified by a set of t polynomials $p_1, \ldots, p_t$ with $\phi(Y_i) = p_i(Y_1, \ldots, Y_t)$.

3 Application: Primality Testing

A number of primality tests use the properties of the ring Z_n where n is the number to be tested. The prominent ones are Miller-Rabin test [Mil76, Rab80], Solovay-Strassen test [SS77], Adleman-Pomerance-Rumely test [APR83] etc. There are several others that use a different algebraic structure, e.g., elliptic curve based tests [GK86].

However, even the ones based on Z_n use properties other than automorphisms of Z_n. The reason is that approaches based on automorphisms do not work. For example, when n is prime, the map $\phi(x) = x^n$ is an automorphism (in fact it is the trivial automorphism); on the other hand when n is composite then ϕ may not be an automorphism. We can use this to design a test, however, as testing if $\phi(x) = x \pmod n$ for all x's requires exponential time, we do the test for only polynomially many x's. This test *does* separate prime numbers from non-square-free composites (see Lemma 1 below), however fails for square-free composites. The reason are Carmichael numbers [Car10]: these are composite numbers for which ϕ is the trivial automorphism.

So an automorphism based property *appears* too weak to separate primes from composites. However, it is not so. The strongest known deterministic primality test [AKS04] is based on the same property of automorphisms as outlined above! What makes it work is the idea of using a polynomial ring instead of Z_n. Let $R = Z_n[Y]/(Y^r - 1)$ where r is a "small" number. As before, the map ϕ remains an automorphism of R when n is prime. It is easy to see that ϕ is an automorphism of R iff for every $g(Y) \in R$,

$$g^n(Y) = \phi(g(Y)) = g(\phi(Y)) = g(Y^n). \tag{1}$$

As above, this can be tested for polynomially many $g(Y)$'s. It was shown in [AKS04] that for a suitably chosen r, if the equation (1) holds for $\sqrt{r} \log n$ many $g(Y)$'s of the form $Y + a$ then n must be a prime power. The analysis in the paper can easily be improved to show that when a's are chosen from $[1, \sqrt{r} \log n]$ then n must be a prime: Suppose equation (1) holds for all a's in the above range. Then we know that n is a prime power. Let $n = p^k$ for some $k > 1$. Let ring $R_0 = Z_{p^2}[Y]/(Y - 1) \cong Z_{p^2}$. Clearly, equation (1) will hold in R_0 too. This implies that for all $a \leq 1 + \sqrt{r} \log n$:

$$a^{p^k} = a \pmod{p^2}.$$

The choice of r is such that $r \geq \log^2 n$ [AKS04] and therefore, the above equation holds for all $a \leq 4\log^2 p$. The following lemma, proved by Hendrik Lenstra [Len] contradicts this:

Lemma 1. *(Hendrik Lenstra) For all large enough primes p, for every $\ell > 0$ there is an $a \leq 4\log^2 p$ such that $a^{p^\ell} \neq a \pmod{p^2}$.*

Proof. Suppose there is an $\ell > 0$ such that $a^{p^\ell} = a \pmod{p^2}$ for all $a \leq 4\log^2 p$. We first prove that we can always assume ℓ to be 1. Consider the case when $\ell > 1$. Since $a^p = a \pmod{p}$, we have

$$a^p = a + p \cdot t \pmod{p^2}$$

for some t. Therefore,

$$\begin{aligned} a^{p^\ell} &= (a + p \cdot t)^{p^{\ell-1}} \pmod{p^2} \\ &= a^{p^{\ell-1}} \pmod{p^2} \end{aligned}$$

Repeating this, we get $a^p = a^{p^\ell} = a \pmod{p^2}$. Now, there are at most p solutions to the equation $a^p = a \pmod{p^2}$ in Z_{p^2}. Since all numbers up to $4\log^2 p$ are solutions to this, so will be all their products. Let $\psi(p^2, 4\log^2 p)$ denote the number of distinct numbers less than p^2 that are $4\log^2 p$-smooth (all their prime factors are at most $4\log^2 p$). Using the bound for ψ [CEG83], $\psi(x, x^{1/u}) = x \cdot u^{-u+o(1)}$ for $u = O(\frac{x}{\log x})$, we get that $\psi(p^2, 4\log^2 p) > p$ for large enough p. This is a contradiction. □

So when n is composite then for at least one of $Y + a$'s, ϕ does not satisfy equation 1 and the test works correctly.

4 Application: Factoring Polynomials

Automorphisms play a central role in efficient factoring of univariate polynomials over finite fields. We outline a randomized polynomial time factoring algorithm using automorphisms. This, and similar algorithms can be found in any text book discussing polynomials over of finite fields, e.g., [LN86, vzGG99]. Let f be a degree d polynomial over finite field F_q. Let $R = F_q[Y]/(f(Y))$ and $\phi : R \mapsto R$, $\phi(x) = x^q$. Clearly, ϕ is an automorphism of R. Notice that if f is irreducible then ϕ^d is trivial. Conversely, if ϕ^d is trivial then, letting f_0 be an irreducible factor of f, ϕ^d is trivial on the ring $F_q[Y]/(f_0(Y))$ as well. Therefore, degree of f_0 divides d. This can be generalized to show that all irreducible factors of f have degrees dividing k iff ϕ^k is trivial. Moreover, ϕ^k is trivial iff $\phi^k(Y) = Y$. An algorithm for *distinct degree square-free* factorization of f follows: for $k = 1$ to d, compute the gcd of $f(Y)$ and $\phi^k(Y) - Y$. The algorithm can also be used

to decide if f is irreducible: f is irreducible iff the smallest k with non-trivial $\gcd(f(Y), \phi^k(Y) - Y)$ is d.

For *equal degree factorization*—given f that is square-free and all irreducible factors of the same degree k—some more work is needed. Find an $t(Y) \in R = F_q[Y]/(f(Y))$ with $t(Y) \notin F_q$ and $\phi(t(Y)) = t(Y)$. Since f is reducible, such a $t(Y)$ always exists and can be found using linear algebra as ϕ is a linear map. Clearly, $t(Y) \pmod{f_i(Y)} \in F_q$ where f_i is an irreducible factor of f and so, $\gcd(t(Y) - x, f(Y)) > 1$ for some $x \in F_q$. This condition can be expressed as a polynomial in x, e.g., $\gcd(t(Y) - x, f(Y)) > 1$ iff $R(t(Y) - x, f(Y)) = 0$ where R is the *resultant* polynomial defined as determinant of a matrix over coefficients on two input polynomials. Therefore, $g(x) = R(t(Y) - x, f(Y)) \in F_q[x]$. By above discussion, a root of this polynomial will provide a factor of f.

To factor $g(x)$, we use the distinct degree factorization method. Choose a random $a \in F_q$ and let $h(x) = g(x+a)$. Then with probability at least $\frac{1}{2}$, $h(x^2)$ can be factored over F_q using the above distinct degree factorization algorithm. To see this, let $g(x) = \prod_{i=1}^{d}(x - \eta_i)$ for $\eta_i \in F_q$. Then $h(x^2) = \prod_{i=1}^{d}(x^2 - \eta_i + a)$. With probability at least $\frac{1}{2}$, there exist i and j such that $\eta_i + a$ is a quadratic residue and $\eta_j + a$ is a quadratic non-residue in F_q. The distinct degree factorization algorithm will separate these factors into two distinct polynomials $h_1(x^2)$ and $h_2(x^2)$. This gives $g(x) = h_1(x - a) \cdot h_2(x - a)$.

Algorithms for polynomial factorization over rationals also (indirectly) use automorphisms since these proceed by first factoring the given polynomial f over a finite field, then use Hensel lifting [Hen18] and LLL algorithm for short lattice vectors [LLL82] to obtain factors over rationals efficiently.

Multivariate polynomial factorization can be reduced, in polynomial time, to the problem of factoring a univariate polynomial via Hilbert irreducibility theorem and Hensel lifting [Kal89]. Therefore, this too, *very* indirectly though, makes use of automorphisms.

5 Application: Factoring Integers

Integer factorization has proven to be much harder than polynomial factorization. The fastest known algorithm is *Number Field Sieve* [LLMP90] with a conjectured time complexity of $2^{O((\log n)^{1/3}(\log\log n)^{2/3})}$. This was preceded by a number of algorithms with provable or conjectured time complexity of $2^{O((\log n)^{1/2}(\log\log n)^{1/2})}$, e.g., Elliptic Curve method [Len87], Quadratic Sieve method [Pom84].

Of these, the fastest two—Quadratic and Number Field Sieve methods—can be easily viewed as trying to find a non-obvious automorphism in a ring. Both the methods aim to find two numbers u and v in Z_n such that $u^2 = v^2$ and $u \neq \pm v$ in Z_n where n is an odd, square-free composite number to be factored. Consider the ring $R = Z_n[Y]/(Y^2 - 1)$. Apart from the trivial automorphism, the ring has another obvious automorphism specified by the map $Y \mapsto -Y$. The problem of finding u and v as above is precisely the one of finding a third automorphism of R. This can be seen as follows. Let ϕ be an automorphism of R with $\phi(Y) \neq \pm Y$. Let

$\phi(Y) = aY + b$. We then have $0 = \phi(Y^2 - 1) = (aY + b)^2 - 1 = a^2 + b^2 - 1 + 2abY$ in R. This gives $ab = 0$ and $a^2 + b^2 = 1$ in Z_n. Notice that $(a, n) = 1$ since otherwise $\phi(\frac{n}{(a,n)}Y) = \frac{n}{(a,n)}b = \phi(\frac{n}{(a,n)}b)$. Therefore, $b = 0$ and $a^2 = 1$. By assumption, $a \neq \pm 1$ and so $u = a$ and $v = 1$. Conversely, given a u and v with $u^2 = v^2$, $u \neq \pm v$ in Z_n, we get $\phi(Y) = \frac{u}{v}Y$ as an automorphism of R.

In fact, as shown in [KS04], factoring integers can be reduced to a number of questions about automorphisms and isomorphisms of rings. They show that an odd, square-free composite number n can be factored in (randomized) polynomial time if

- one can count the number of automorphisms of the ring $Z_n[Y]/(Y^2)$, or
- one can find an isomorphism between rings $Z_n[Y]/(Y^2 - a^2)$ and $Z_n[Y]/(Y^2 - 1)$ for a randomly chosen $a \in Z_n$, or
- one can find a non-trivial automorphism of the ring $Z_n[Y]/(f(Y))$ where f is a randomly chosen polynomial of degree three.

6 Application: Graph Isomorphism

In this section, we consider the application of ring isomorphisms for solving the graph isomorphism problem. It was shown in [KS04] that testing isomorphism between two graphs on n vertices can be reduced to testing the isomorphism between two rings of the form $Z_{p^3}[Y_1, \ldots, Y_n]/\mathcal{I}$ where p is any odd prime and $\mathcal{I}$ is an ideal generated by certain degree two and three polynomials. Here, borrowing ideas from [KS04] and [Thi98] we give a different, and more general, reduction.

Let $G = (V, E)$ be a simple graph on n vertices. We define polynomial p_G as:

$$p_G(x_1, \ldots, x_n) = \sum_{(i,j)\in E} x_i \cdot x_j.$$

Also, define ideal $\mathcal{I}_G$ as:

$$\mathcal{I}_G(x_1, \ldots, x_n) = (p_G(x_1, \ldots, x_n), \{x_i^2\}_{1\leq i\leq n}, \{x_i x_j x_k\}_{1\leq i,j,k\leq n}). \tag{2}$$

Then,

Theorem 2. *Simple graphs G_1 and G_2 over n vertices are isomorphic iff either $G_1 = G_2 = K_m \cup D_{n-m}$ (D_{n-m} is a collection of $n-m$ isolated vertices) or rings $R_1 = F_q[Y_1, \ldots, Y_n]/\mathcal{I}_{G_1}(Y_1, \ldots, Y_n)$ and $R_2 = F_q[Z_1, \ldots, Z_n]/\mathcal{I}_{G_2}(Z_1, \ldots, Z_n)$ are isomorphic. Here F_q is a finite field of odd characteristic.*[3]

Proof. If the graphs are isomorphic, then the map ϕ, $\phi : R_1 \mapsto R_2$, $\phi(Y_i) = Z_{\pi(i)}$, is an isomorphism between the rings where π is an isomorphism mapping G_1 to G_2. This follows since $\phi(p_{G_1}(Y_1, \ldots, Y_n)) = p_{G_2}(Z_1, \ldots, Z_n)$. Conversely,

[3] The theorem also holds for fields of characteristic two. For such fields though, we need to change the definition of the ideal $\mathcal{I}_G$. It now contains $x_{n+1} \cdot p_G$, x_i^3's and $x_i x_j x_k x_\ell$'s and the ring is defined over $n + 1$ variables. The proof is similar.

suppose that G_2 is not of the form $K_m \cup D_{n-m}$ and the two rings are isomorphic. Let ϕ, $\phi : R_1 \mapsto R_2$ be an isomorphism. Let

$$\phi(Y_i) = \alpha_i + \sum_{1 \le j \le n} \beta_{i,j} Z_j + \sum_{1 \le j < k \le n} \gamma_{i,j,k} Z_j Z_k.$$

Since $Y_i^2 = 0$ in the ring,

$$0 = \phi(Y_i^2) = \phi^2(Y_i) = \alpha_i^2 + (\text{higher degree terms}).$$

This gives $\alpha_i = 0$. Again looking at the same equation:

$$0 = \phi(Y_i^2) = \phi^2(Y_i) = 2 \sum_{1 \le j < k \le n} \beta_{i,j} \beta_{i,k} Z_j Z_k.$$

If more than one $\beta_{i,j}$ is non-zero, then we must have $\sum_{j,k \in J, j<k} \beta_{i,j}\beta_{i,k} Z_j Z_k$ divisible by $p_{G_2}(Z_1, \ldots, Z_n)$ where J is the set of non-zero indices. Since p_{G_2} is also homogeneous polynomial of degree two, it must be a constant multiple of the above expression implying that $G_2 = K_{|J|} \cup D_{n-|J|}$. This is not possible by assumption. Therefore, at most one $\beta_{i,j}$ is non-zero. Now suppose that *all* $\beta_{i,j}$'s are zero. But then $\phi(Y_i Y_\ell) = 0$ which is not possible. Hence, *exactly one* $\beta_{i,j}$ is non-zero for every i.

Define $\pi(i) = j$ where j is the index with $\beta_{i,j}$ non-zero. Suppose $\pi(i) = \pi(\ell)$ for $i \neq \ell$. Then, $\phi(Y_i Y_\ell) = 0$. Again, this is not possible. Hence π is a permutation on $[1, n]$. Now consider $\phi(p_{G_1}(Y_1, \ldots, Y_n))$. It follows that:

$$\begin{aligned} 0 &= \phi(p_{G_1}(Y_1, \ldots, Y_n)) \\ &= \sum_{(i,j) \in E_1} \phi(Y_i)\phi(Y_j) \\ &= \sum_{(i,j) \in E_1} \beta_{i,\pi(i)} \beta_{j,\pi(j)} Z_{\pi(i)} Z_{\pi(j)} \end{aligned}$$

The last expression must be divisible by p_{G_2}. This gives $\beta_{i,\pi(i)} = \beta_{\ell,\pi(\ell)}$ for all i and ℓ. This implies that the expression is a constant multiple of p_{G_2}, or equivalently, that G_1 is isomorphic to G_2. □

Notice that the rings R_1 and R_2 constructed above have lots of automorphisms. For example, $Y_i \mapsto Y_i + Y_1 Y_2$ is a non-trivial automorphism of R_1. Therefore, automorphisms of graph G_1 do not directly correspond to automorphisms of the ring R_1. In fact, each automorphism of G_1 gives rise to at least $p^{n \cdot (\binom{n}{2} - 1)}$ automorphisms of R_1 (this is the number of ways we can add quadratic terms to the automorphism map).

7 Application: Polynomial Equivalence

Thomas Thierauf [Thi98] analyzed the complexity of *polynomial isomorphism* problem where one tests if the two given polynomials, say p and q, become equal

after a permutation of variables of p. He showed that this problem is in NP $\cap$ coAM and Graph Isomorphism reduces to it. His upper bound proof can easily be generalized to polynomial equivalence. We first prove a lower bound by showing that ring isomorphism problem reduces to it.

Theorem 3. *Ring isomorphism problem for rings of prime characteristic reduces, in polynomial time, to cubic polynomial equivalence.*

Proof. For this proof, we adopt the basis representation of rings. Let R and R' be two rings with additive basis $b_1, \ldots, b_n$ and $d_1, \ldots, d_n$ respectively and characteristic p. Multiplication in R is defined as

$$(\forall)\ i, j, 1 \leq i, j \leq n : b_i \cdot b_j = \sum_{k=1}^{n} a_{i,j,k} b_k \text{ where } a_{i,j,k} \in F_p.$$

Let us define a polynomial which *captures* the relations defining ring R:

$$f_R(\bar{y}, \bar{b}) := \sum_{1 \leq i \leq j \leq n} y_{i,j} \left(b_i b_j - \sum_{1 \leq k \leq n} a_{i,j,k} b_k \right) \tag{3}$$

Similarly, we define $f_{R'}$ over variables $\bar{z}$ and $\bar{d}$.

Let us start off with an easy observation:

Claim 1. *If rings R and R' are isomorphic then f_R is equivalent to $f_{R'}$.*

Proof of Claim. Let ϕ be an isomorphism from R to R'. Note that ϕ sends each b_i to a linear combination of d's and for all i, j, $\phi(b_i)\phi(b_j) - \sum_{1 \leq k \leq n} a_{i,j,k}\phi(b_k) = 0$ in R'. This implies that there exist c's in F_p such that

$$\phi(b_i)\phi(b_j) - \sum_{1 \leq s \leq n} a_{i,j,s}\phi(b_s) = \sum_{1 \leq k \leq l \leq n} c_{i,j,k,\ell} \left(d_k d_\ell - \sum_{1 \leq s \leq n} a'_{k,\ell,s} d_s \right).$$

This immediately suggests that the linear transformation:

$$b_i \mapsto \phi(b_i)$$
$$\sum_{1 \leq i \leq j \leq n} c_{i,j,k,\ell} y_{i,j} \mapsto z_{k,\ell}$$

makes f_R equal to $f_{R'}$. □

Conversely,

Claim 2. *If f_R is equivalent to $f_{R'}$ then R and R' are isomorphic.*

Proof of Claim. Let ϕ be a linear transformation such that

$$\sum_{1 \leq i \leq j \leq n} \phi(y_{i,j}) \left(\phi(b_i)\phi(b_j) - \sum_{1 \leq k \leq n} a_{i,j,k}\phi(b_k) \right)$$

$$= \sum_{1 \le i \le j \le n} z_{i,j} \left(d_i d_j - \sum_{1 \le k \le n} a'_{i,j,k} d_k \right). \quad (4)$$

This immediately implies that

$$\sum_{1 \le i \le j \le n} \phi(y_{i,j})\phi(b_i)\phi(b_j) = \sum_{1 \le i \le j \le n} z_{i,j} d_i d_j. \quad (5)$$

We intend to show that $\phi(b_i)$ has no z's, i.e., $\phi(b_i)$ is a linear combination of only d's. We will be relying on the following property of rhs of equation (5): *let τ be an invertible linear transformation on the z's then for all $1 \le i \le j \le n$ the coefficient of $z_{i,j}$ in $\sum_{1 \le i \le j \le n} \tau(z_{i,j}) d_i d_j$ is nonzero.*

Suppose $\phi(b_1)$ has z's:

$$\phi(b_1) = \sum_i c_{1,i} d_i + \sum_{ij} c_{1,i,j} z_{i,j}$$

We can apply an invertible linear transformation τ on z's in equation (5) so that $\tau : \sum_{i,j} c_{1,i,j} z_{i,j} \mapsto z_{1,1}$ and then apply an evaluation map *val* by fixing $z_{1,1} \leftarrow -(\sum_i c_{1,i} d_i)$. So equation (5) becomes:

$$\sum_{2 \le i \le j \le n} val \circ \tau \circ \phi(y_{i,j} b_i b_j) = \sum_{1 \le i \le j \le n; i,j \ne 1,1} z_{i,j}(\text{quadratic } d\text{'s}) + (\text{cubic } d\text{'s}) \quad (6)$$

We repeat this process of applying invertible linear transformations on z's and fixing z's in equation (6) so that for all $2 \le i \le j \le n$, $val \circ \tau \circ \phi(y_{i,j} b_i b_j)$ either vanishes or is a cubic in d's. Thus, after $1 + \binom{n}{2}$ z-fixings the lhs of equation (5) is a cubic in d's while the rhs still has $\binom{n+1}{2} - \binom{n}{2} - 1 = (n-1)$ unfixed z's, which is a contradiction.

Since $\phi(b)$'s have no z's and there are no cubic d's in rhs of equation (4) we can ignore the d's in $\phi(y)$'s. Thus, now $\phi(y)$'s are linear combinations of z's and $\phi(b)$'s are linear combinations of d's. Again looking at equation (4), this means that $\left(\phi(b_i)\phi(b_j) - \sum_{1 \le s \le n} a_{i,j,s} \phi(b_s)\right)$ is a linear combination of $\left(d_k d_\ell - \sum_{1 \le s \le n} a'_{k,\ell,s} d_s\right)$ where $1 \le k, \ell \le n$. This implies that

$$\left(\phi(b_i)\phi(b_j) - \sum_{1 \le s \le n} a_{i,j,s} \phi(b_s) \right) = 0$$

in ring R'. This combined with the fact that ϕ is an invertible linear transformation on $\bar{b}$ means that ϕ induces an isomorphism from ring R to R'. □

The above two claims complete the proof. □

In the case of Graph Isomorphism, we can reduce the problem to cubic form equivalence.

Theorem 4. *Graph Isomorphism reduces in polynomial time to cubic form equivalence.*

Proof. Suppose we are given two graphs G_1 and G_2 and we have rings R_1 and R_2 as in the proof of Theorem 2. To simplify matters suppose $(i_0, j_0) \in E(G_1), E(G_2)$. We fix an additive basis $\{1, b_1, \ldots, b_m\}$ of the ring R_1 over F_p such that

$$b_1 = Y_1, \ldots, b_n = Y_n, \{b_{n+1}, \ldots, b_m\} = \{Y_i Y_j\}_{1 \le i < j \le n} \setminus \{Y_{i_0} Y_{j_0}\}. \tag{7}$$

Note that $m = \binom{n+1}{2} - 1$ and that $\{b_1, \ldots, b_m\}$ is an additive basis of the maximal ideal $\mathcal{M}$ ($\mathcal{M}'$) of *local* ring R_1 (R_2). Also, $b_i b_j = 0$ except for $\binom{n}{2}$ unordered tuples (i, j).

As local rings are isomorphic iff their maximal ideals are isomorphic [McD74], we focus on $\mathcal{M}$ and $\mathcal{M}'$. So let us construct homogeneous cubic polynomials *capturing* the relations in $\mathcal{M}, \mathcal{M}'$. These polynomials are similar to the ones seen in the proof of Theorem 3:

$$f_{\mathcal{M}}(u, \bar{y}, \bar{b}) = \sum_{1 \le i \le j \le m} y_{i,j} \left(b_i b_j - u \sum_{1 \le k \le m} a_{i,j,k} b_k \right) + u^3$$

$$f_{\mathcal{M}'}(v, \bar{z}, \bar{d}) = \sum_{1 \le i \le j \le m} z_{i,j} \left(d_i d_j - v \sum_{1 \le k \le m} a'_{i,j,k} d_k \right) + v^3$$

where, $a_{i,j,k}, a'_{i,j,k} \in \{-1, 0, 1\}$ are given by the definition of ideal $\mathcal{I}_G$ and b's in equations (2) and (7).

Let us start off with the easier side:

Claim 3. *If G_1 is isomorphic to G_2 then $f_{\mathcal{M}}$ is equivalent to $f_{\mathcal{M}'}$.*

Proof of Claim. If G_1 is isomorphic to G_2 then by Theorem 2, R_1 is isomorphic to R_2 which means $\mathcal{M}$ is isomorphic to $\mathcal{M}'$. Now by sending $u \mapsto v$ and following the proof of claim 1, we deduce $f_{\mathcal{M}}$ is equivalent to $f_{\mathcal{M}'}$. □

Conversely,

Claim 4. *If $f_{\mathcal{M}}$ is equivalent to $f_{\mathcal{M}'}$ then G_1 is isomorphic to G_2.*

Proof of Claim. We will try to show that if $f_{\mathcal{M}}$ is equivalent to $f_{\mathcal{M}'}$ then $\mathcal{M}$ is isomorphic to $\mathcal{M}'$, which when combined with Theorem 2 means that the graphs are isomorphic.

Suppose ϕ is an invertible linear transformation on $(u, \bar{y}, \bar{b})$ such that:

$$\sum_{1 \le i \le j \le m} \phi(y_{i,j}) \left(\phi(b_i)\phi(b_j) - \phi(u) \sum_{1 \le k \le m} a_{i,j,k} \phi(b_k) \right) + \phi(u)^3$$

$$= \sum_{1 \le i \le j \le m} z_{i,j} \left(d_i d_j - v \sum_{1 \le k \le n} a'_{i,j,k} d_k \right) + v^3. \tag{8}$$

The main idea again is to show that $\phi(b_i)$ is a linear combination of d's and the proof is very similar to the one above.

Suppose $\phi(b_1)$ has z's:

$$\phi(b_1) = c_{1,v}v + \sum_i c_{1,i}d_i + \sum_{i,j} c_{1,i,j}z_{i,j}.$$

As before, We apply an invertible linear transformation τ on z's in equation (8) so that $\tau : \sum_{i,j} c_{1,i,j}z_{i,j} \mapsto z_{1,1}$ and then apply an evaluation map val by fixing $z_{1,1} \leftarrow -(c_{1,v}v + \sum_i c_{1,i}d_i)$. So equation (8) becomes:

$$\sum_{2\le i\le j\le m} val\circ\tau\circ\phi(y_{i,j}b_ib_j) - \sum_{1\le i\le j\le m} val\circ\tau\circ\phi\left(uy_{i,j}\sum_{1\le k\le m} a_{i,j,k}b_k\right) + val\circ\tau\circ\phi(u)^3$$

$$= \sum_{1\le i\le j\le m; i,j\ne 1,1} z_{i,j}\left((\text{quadratic } d\text{'s}) - v(\text{linear } d\text{'s})\right) + (\text{cubic in } v, d\text{'s}). \quad (9)$$

Note that now on the lhs of the equation (9) there are at most $\binom{m}{2}$ terms of the form $val\circ\tau\circ\phi(y_{i,j}b_ib_j)$. And since except for $\binom{n}{2}$ pairs (i,j), the product b_ib_j is zero, there are at most $\binom{n}{2}$ terms of the form $val\circ\tau\circ\phi\left(uy_{i,j}\sum_{1\le k\le m} a_{i,j,k}b_k\right)$. We repeat this process of applying invertible linear transformations on z's and fixing z's in equation (9) so that the expressions $val\circ\tau\circ\phi(y_{i,j}b_ib_j)$ for $2 \le i \le j \le m$, $val\circ\tau\circ\phi\left(uy_{i,j}\sum_{1\le k\le m} a_{i,j,k}b_k\right)$ for $1 \le i \le j \le m$, and $val\circ\tau\circ\phi(u)^3$ either vanish or are cubics in v and d's. Thus, after at most $1 + \binom{m}{2} + \binom{n}{2} + 1$ z-fixings the lhs of equation (8) is a cubic in v and d's while the rhs still has $\binom{m+1}{2} - \binom{m}{2} - \binom{n}{2} - 2 = m - \binom{n}{2} - 2 = \binom{n+1}{2} - 1 - \binom{n}{2} - 2 = n - 3 > 0$ unfixed z's, which is a contradiction.

So $\phi(b_i)$'s have no z's. Now if $\phi(u)$ has $z_{i,j}$ then there is a nonzero coefficient of $z_{i,j}^3$ on the lhs of equation (8) while $z_{i,j}^3$ does not appear on the rhs. Thus, even $\phi(u)$ has no z's. Looking at equation (8) we deduce that all the z's on the lhs occur in $\phi(y)$'s. So we can apply a suitable invertible linear transformation τ on the z's such that for all $1 \le i \le j \le m$:

$$\tau\circ\phi(y_{i,j}) = z_{i,j} + \sum_{1\le k\le m} c_{i,j,k}d_k + c_{i,j,v}v,$$

and then equation (8) simply looks like:

$$\sum_{1\le i\le j\le m} z_{i,j}\left(\phi(b_i)\phi(b_j) - \phi(u)\sum_{1\le k\le m} a_{i,j,k}\phi(b_k)\right) + (\text{cubic in } v, d\text{'s})$$

$$= \sum_{1\le i\le j\le m} z_{i,j}((\text{quadratic } d\text{'s}) - v(\text{linear } d\text{'s})) + v^3.$$

Therefore,

$$\sum_{1\le i\le j\le m} z_{i,j}\left(\phi(b_i)\phi(b_j) - \phi(u)\sum_{1\le k\le m} a_{i,j,k}\phi(b_k)\right)$$
$$= \sum_{1\le i\le j\le m} z_{i,j}((\text{quad } d\text{'s}) - v(\text{linear } d\text{'s})). \tag{10}$$

Let us compare the coefficients of $z_{i,i}$ in equation (10):

$$\phi(b_i)^2 = (\text{quadratic } d\text{'s}) - v(\text{linear } d\text{'s}).$$

This clearly rules out $\phi(b_i)$ having a nonzero coefficient of v. Thus, $\phi(b_i)$'s are linear combinations of d's. Since we have obtained equation (10) from equation (8) by applying *invertible* linear transformation on z's, there has to be a nonzero v coefficient in the rhs and hence in the lhs of equation (10). Thus, $\phi(u)$ has a nonzero v coefficient. Say, for some $c_{u,v} \neq 0$:

$$\phi(u) = c_{u,v}v + \sum_{1\le k\le m} c_{u,k}d_k.$$

For any $1 \le i \le j \le m$, by comparing coefficients of $z_{i,j}$ in equation (10) we get that there exist elements $e_{i,j,k,\ell} \in F_p$ such that:

$$\phi(b_i)\phi(b_j) - \left(c_{u,v}v + \sum_{1\le s\le m} c_{u,s}d_s\right)\sum_{1\le s\le m} a_{i,j,s}\phi(b_s)$$
$$= \sum_{1\le k\le \ell\le m} e_{i,j,k,\ell}\left(d_k d_\ell - v\sum_{1\le s\le m} a'_{k,l,s}d_s\right).$$

By fixing $v = 1$ this actually means that in the ring $\mathcal{M}'$:

$$\phi(b_i)\phi(b_j) = \left(c_{u,v} + \sum_{1\le s\le m} c_{u,s}d_s\right)\sum_{1\le s\le m} a_{i,j,s}\phi(b_s). \tag{11}$$

Notice that there is an inverse of the expression $\left(c_{u,v} + \sum_{1\le s\le m} c_{u,s}d_s\right)$ in the ring R_2 that looks like:

$$\left(c_{u,v} + \sum_{1\le s\le m} c_{u,s}d_s\right)^{-1} = \left(c_{u,v}^{-1} + \sum_{1\le s\le m} c'_{u,s}d_s\right). \tag{12}$$

Since the product of any three terms in $\mathcal{M}'$ vanishes, we get the following when we multiply both sides of equation (11) by the inverse (12) in $\mathcal{M}'$:

$$c_{u,v}^{-1}\phi(b_i)\phi(b_j) = \sum_{1\le s\le m} a_{i,j,s}\phi(b_s)$$
$$\Rightarrow \frac{\phi(b_i)}{c_{u,v}}\frac{\phi(b_j)}{c_{u,v}} = \sum_{1\le s\le m} a_{i,j,s}\frac{\phi(b_s)}{c_{u,v}}.$$

In other words, this means that $b_i \mapsto \frac{\phi(b_i)}{c_{u,v}}$ is an isomorphism from $\mathcal{M} \to \mathcal{M}'$.

□

This completes the reduction from graph isomorphism to cubic form equivalence. □

Polynomial equivalence for homogeneous constant degree polynomials efficiently reduces to ring isomorphism for certain degrees.

Theorem 5. *Polynomial equivalence for homogeneous degree d polynomials over field F_q with $(d, q-1) = 1$ reduces, in time $n^{O(d)}$, to ring isomorphism.*

Proof. Let p and q be two homogeneous degree d polynomials over field F_q with n variables. Define rings R_p and R_q as:

$$R_p = F_q[\bar{Y}]/(p(\bar{Y}), \{Y_{j_1} Y_{j_2} \cdots Y_{j_{d+1}}\}_{1 \le j_1, j_2, \ldots, j_{d+1} \le n})$$
$$R_q = F_q[\bar{Z}]/(q(\bar{Z}), \{Z_{j_1} Z_{j_2} \cdots Z_{j_{d+1}}\}_{1 \le j_1, j_2, \ldots, j_{d+1} \le n}).$$

It is easy to see that if p and q are equivalent, then R_p and R_q are isomorphic.

The converse is also not difficult. Let ϕ be an isomorphism from R_p to R_q. Let

$$\phi(Y_i) = \alpha_i + \sum_{j=1}^{n} \beta_{i,j} Z_j + (\text{higher degree terms}). \tag{13}$$

The fact $\phi^{d+1}(Y_i) = 0$ implies that $\alpha_i = 0$. Let $\psi(Y_i) = \sum_{j=1}^{n} \beta_{i,j} Z_j$, i.e., the linear component of ϕ. We show that ψ is (almost) an equivalence between p and q.

First of all, ψ is an invertible linear transformation. This is because for every j, there exists a polynomial r_j such that $\phi(r_j(\bar{Y})) = Z_j$ (using the fact that ϕ is an isomorphism). Let r_j^L be the linear part of r_j. Then, $\phi(r_j^L(\bar{Y})) = Z_j +$ (higher degree terms). It follows that $\psi(r_j^L(\bar{Y})) = Z_j$.

Now consider the polynomial p. We have

$$\phi(p(\bar{Y})) \in (q(\bar{Z}), \{Z_{j_1} Z_{j_2} \cdots Z_{j_{d+1}}\}_{1 \le j_1, j_2, \ldots, j_{d+1} \le n}).$$

Of the polynomials defining the ideal in above equation, only q is of degree d. Hence the degree d part of $\phi(p(\bar{Y}))$ must be divisible by $q(\bar{Z})$. In other words, $\psi(p(\bar{Y}))$ is divisible by $q(\bar{Z})$. Since both p and q have the same degree, this means $\psi(p(\bar{Y})) = c \cdot q(\bar{Z})$ for $c \in F_q$. Since $(d, q-1) = 1$, there exists an $e \in F_q$ with $e^d = c$. Therefore, the map $\frac{1}{e}\psi$ is an equivalence. □

The restriction on degree in the above theorem, $(d, q-1) = 1$, appears necessary. For example, consider polynomials x^2 and ax^2 over field F_q with a being a quadratic non-residue. These two polynomials are *not* equivalent while the rings defined by them, $F_q[Y]/(Y^2)$ and $F_q[Y]/(aY^2)$ are equal.

8 Open Questions

We have listed a number of useful applications of automorphisms and isomorphisms of finite rings in complexity theory. Our list is by no means exhaustive, but should convince the reader about the importance of these. We pose a few questions that we would like to see an answer of:

- It is not clear if automorphisms play a role in some important algebraic problems, e.g., *discrete log*. This problem can easily be viewed as that of finding a certain kind of automorphism in a group, however, we do not know any connections to ring automorphisms.
- Nearly all the effort in integer factoring has been concentrated towards finding automorphism in the ring $Z_n[Y]/(Y^2-1)$. Is there another ring where this problem might be "easier"? Can some of the other formulations of [KS04] be used for factoring?
- Theorems 2 and 4 together show that Graph Isomorphism reduces to equivalence of cubic forms over fields of *any* characteristic. Can the theory of cubic forms (over complex numbers) be used to find a subexponential time algorithm for Graph Isomorphism?
- It appears likely that ring isomorphism problem reduces to equivalence of cubic forms, but we have not been able to find a proof.
- It appears likely that equivalence of constant degree polynomials reduces to ring isomorphism at least when $(d, q-1) = 1$. However, we have been able to prove it only for homogeneous polynomials.

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160:1–13, 2004.

[APR83] L. M. Adleman, C. Pomerance, and R. S. Rumely. On distinguishing prime numbers from composite numbers. *Annals of Mathematics*, 117:173–206, 1983.

[Car10] R. D. Carmichael. Note on a number theory function. *Bull. Amer. Math. Soc.*, 16:232–238, 1910.

[CEG83] E. R. Canfield, P. Erdos, and A. Granville. On a problem of Oppenhein concerning "Factorisatio Numerorum". *J. Number Theory*, 17:1–28, 1983.

[GK86] S. Goldwasser and J Kilian. Almost all primes can be quickly certified. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 316–329, 1986.

[Hen18] Kurt Hensel. Eine neue Theorie der algebraischen Zahlen. *Mathenatische Zeitschift*, 2:433–452, 1918.

[Kal89] E. Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness and Computation*, pages 375–412. JAI press, 1989.

[KS04] Neeraj Kayal and Nitin Saxena. On the ring isomorphism and automorphism problems. Technical Report TR04-109, Electronic Colloquium on Computational Complexity (http://www.eccc.uni-trier.de/eccc), 2004. Available at eccc.uni-trier.de/eccc-reports/2004/TR04-109/Paper.pdf.

[Lan93] S. Lang. *Algebra*. Addison-Wesley, 1993.

[Len] H. W. Lenstra, Jr. Private communication.

[Len87] Hendrik Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[LLL82] Arjen Lenstra, Hendrik Lenstra, and L. Lovasz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[LLMP90] Arjan K. Lenstra, Hendrik W. Lenstra, M. S. Manasse, and J. M. Pollard. The number field sieve. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 564–572, 1990.

[LN86] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1986.

[McD74] B. R. McDonald. *Finite Rings with Identity*. Marcel Dekker, Inc., 1974.

[Mil76] G. L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Sys. Sci.*, 13:300–317, 1976.

[Pat96] J. Patarin. Hidden field equations (HFE) and isomorphism of polynomials (IP): Two new families of assymetric algorithms. In *EUROCRYPT'96*, pages 33–48. Springer LNCS 1070, 1996.

[Pom84] Carl Pomerance. The quadratic sieve factoring algorithm. In *EUROCRYPT 1984*, pages 169–182. Springer LNCS 209, 1984.

[Rab80] M. O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12:128–138, 1980.

[SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6:84–86, 1977.

[Thi98] Thomas Thierauf. The isomorphism problem for read-once branching programs and arithmetic circuits. *Chicago Journal of Theoretical Computer Science*, 1998, 1998.

[vzGG99] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

Algebraic Generating Functions in Enumerative Combinatorics and Context-Free Languages

Mireille Bousquet-Mélou*

CNRS, LaBRI, Université Bordeaux 1
351 cours de la Libération, 33405 Talence Cedex, France
mireille.bousquet@labri.fr

Abstract. Numerous families of simple discrete objects (words, trees, lattice walks...) are counted by a rational or algebraic generating function. Whereas it seems that objects with a rational generating function have a structure very similar to the structure of words of a regular language, objects with an algebraic generating function remain more mysterious. Some of them, of course, exhibit a clear "algebraic" structure, which recalls the structure of words of context-free languages. For many other objects, such a structure has not yet been discovered. We list several examples of this type, and discuss various methods for proving the algebraicity of a generating function.

1 Introduction

The general topic of this paper is the enumeration of simple discrete objects (words, trees, lattice walks...) and more specifically the *rational* or *algebraic* nature of the associated generating functions. Let $\mathcal{A}$ be a class of discrete objects equipped with a size:

$$\begin{aligned} size : \mathcal{A} &\to \mathbb{N} \\ A &\mapsto |A| \end{aligned}$$

Assume that for all n,

$$\mathcal{A}_n := \{A \in \mathcal{A} : |A| = n\}$$

is finite. Let $a_n = |\mathcal{A}_n|$. The (ordinary) generating function of the objects of $\mathcal{A}$, counted by their size, is the following formal power series in the indeterminate t:

$$A(t) := \sum_{n \geq 0} a_n t^n = \sum_{A \in \mathcal{A}} t^{|A|}.$$

The purpose of enumerative combinatorics is to provide tools for finding a closed formula for the numbers a_n, or an expression for the generating function

* MBM was partially supported by the European Commission's IHRP Programme, grant HPRN-CT-2001-00272, "Algebraic Combinatorics in Europe".

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 18–35, 2005.

$A(t)$. In many cases, one is happy enough to find a recurrence relation defining the sequence a_n, or a functional equation defining $A(t)$. Enumerative problems naturally arise in various fields of mathematics and computer science, such as probability theory and the average case analysis of algorithms. Numerous interesting problems also arise from models in statistical physics, the most celebrated probably being the Ising model. When dealing in problems with a computer science of physics flavour, it is often sufficient, and more informative, to obtain the asymptotic behaviour of the numbers a_n rather than an exact formula.

Before defining the main classes of generating functions we are interested in, let us examine a few simple examples.

Example 1: A Regular Language
Let $\mathcal{L} = (a + bb)^*$, and let ℓ_n be the number of words of length n in $\mathcal{L}$. Clearly, $\ell_0 = \ell_1 = 1$, and for $n \geq 2$,

$$\ell_n = \ell_{n-1} + \ell_{n-2}. \tag{1}$$

Hence ℓ_n is the sequence of Fibonacci numbers. Solving the above recurrence relation gives

$$\ell_n = \frac{1}{\sqrt{5}} \left(\mu^{n+1} - \left(-\frac{1}{\mu} \right)^{n+1} \right) \quad \text{where} \quad \mu = \frac{1+\sqrt{5}}{2}.$$

Multiplying (1) by t^n, and summing over $n \geq 2$ gives

$$L(t) := \sum_{n \geq 0} \ell_n t^n = \frac{1}{1 - t - t^2},$$

in accordance with the fact that the non-commutative generating function of $\mathcal{L}$ is

$$\frac{1}{1 - a - bb}.$$

The closed form expression of ℓ_n implies that $\ell_n \sim \mu^{n+1}/\sqrt{5}$ as $n \to \infty$.

Example 2: Ternary Trees
Let $\mathcal{T}$ be the set of plane trees in which any vertex has either three children (ordered from left to right) or no child at all. In the former case, the vertex is said to be a node, and in the latter case, it is called a leaf. Let t_n be the number of such trees – called ternary trees – having n nodes. Then $t_0 = t_1 = 1$ and for $n \geq 2$,

$$t_n = \sum_{i,j,k \geq 0, i+j+k=n-1} t_i t_j t_k.$$

This recurrence relation is simply obtained by looking at the sizes of the three subtrees attached to the root. It translates into the following algebraic equation that defines the generating function $T(t) = \sum_{n \geq 0} t_n t^n$:

$$T(t) = 1 + tT(t)^3.$$

It is possible, but not very pleasant, to solve this equation and write $T(t)$ with radicals. More interestingly, the *Lagrange inversion formula [47–p. 38]* allows one to derive from the equation the following expression of t_n:

$$t_n = \frac{1}{3n+1}\binom{3n+1}{n} = \frac{1}{n+1}\binom{3n}{n}.$$

This formula may also be obtained by a purely combinatorial argument, through the encoding of ternary trees by certain *Lukasiewicz words*. The enumeration of these words is then performed via the *cycle lemma* [47–Ch. 5].

Using Stirling's formula, one finds

$$t_n \sim \frac{\sqrt{3}}{4\sqrt{\pi}}\left(\frac{27}{4}\right)^n n^{-3/2}.$$

Note that for $n \geq 0$,

$$2(2n+3)(n+1)t_{n+1} = 3(3n+2)(3n+1)t_n,$$

with the initial condition $t_0 = 1$, and this implies that the series $T(t)$ satisfies the linear differential equation:

$$6T(t) + 6(9t-1)T'(t) + t(27t-4)T''(t) = 0.$$

Example 3: Loops in the Plane
Let $\mathcal{W}$ be the set of walks in the discrete plane, formed from North, South, East and West steps, that start and end at the origin $(0,0)$. The length of such walks is necessarily even. Let w_n be the number of such walks (called *loops*) having $2n$ steps. Alternatively, w_n is the number of words on the alphabet $\{N,S,E,W\}$ having as many N's and S's, and as many E's as W's. By projecting the walk onto the two main diagonals of the plane, one finds

$$w_n = \binom{2n}{n}^2 \sim \frac{4^{2n}}{\pi n}.$$

These numbers satisfy the recurrence relation

$$(n+1)^2 w_{n+1} = 4(2n+1)^2 w_n$$

with the initial condition $w_0 = 1$. This gives the following linear differential equation satisfied by $W(t) = \sum_{n\geq 0} w_n t^n$:

$$4W(t) + (32t-1)W'(t) + t(16t-1)W''(t) = 0.$$

As we shall see later, the term n^{-1} in the asymptotic behaviour of w_n prevents the series $W(t)$ from satisfying a (non-trivial) polynomial equation of the form $P(t, W(t)) = 0$.

In enumerative combinatorics, there is a strong interest in the *nature* of the generating function for a class of objects. Before we try to explain why, let us define the three main types of formal power series that are usually considered: rational series, algebraic series, and D-finite series.

The formal power series $A(t)$ is *rational* if it can be written in the form

$$A(t) = \frac{P(t)}{Q(t)}$$

where $P(t)$ and $Q(t)$ are polynomials in t (see [46–Ch. 4]). In particular, the generating function of Example 1 is rational.
The series $A(t)$ is *algebraic* (over $\mathbb{Q}(t)$) if it satisfies a (non-trivial) polynomial equation [47–Ch. 6]:

$$P(t, A(t)) = 0.$$

The *degree* of $A(t)$ is the smallest possible degree of P (in its second variable). The generating function of ternary trees was shown in Example 2 to be algebraic (of degree 3).

The series $A(t)$ is *D-finite* if it satisfies a (non-trivial) linear differential equation [47–Ch. 6]:

$$P_k(t)A^{(k)}(t) + \cdots + P_1(t)A'(t) + P_0(t)A(t) = 0.$$

The generating function of loops in the plane was shown in Example 3 to be D-finite.

Why do combinatorialists like these families of series? Firstly, combinatorialists obey the general mathematical temptation of classifying everything that they see. Note that the three classes of series we have defined form a hierarchy, since every rational series is algebraic and every algebraic series is D-finite. Secondly, these three classes of series are rather well-behaved:

– they have interesting closure properties: to mention only the simplest ones, each of these families is closed under the sum and product of series,

– they can be *guessed* from sufficiently many of their first coefficients (for instance using the Maple package GFUN [42]),

– they are reasonably easy to handle via computer algebra (partial fraction expansions, elimination, resultants, Gröbner bases, GFUN...)

– the asymptotic behaviour of their coefficients is rather smooth, and can in general be determined automatically: typically, for a D-finite series,

$$a(n) \sim \alpha \mu^n n^\gamma (\log n)^j$$

where α, μ and γ are algebraic over $\mathbb{Q}$ and $j \in \mathbb{N}$. For algebraic series, $j = 0$ and $\gamma \in \mathbb{Q} \setminus \{-1, -2, \ldots\}$. Moreover, for rational series, the exponent γ belongs to $\mathbb{N}$. The word "typically" means that this is not exactly true... See [29] or [30] for more details on the algebraic case. The above description is especially incomplete in the case of D-finite series: their coefficients may actually grow faster than any exponential when the differential equation they satisfy has an *irregular singular point*. However, the systematic study of the asymptotic behaviour of the coefficients remains attainable via the determination of singular expansions

of the solutions (see [35, 51] for the theory, and the Maple packages DEtools and LREtools for its implementation).

Finally, and most importantly in this paper, there are also *combinatorial reasons* why we like to be able to determine in which class of series the generating function we consider lies: there is a general intuition as to what a class of objects with a rational or algebraic generating function looks like. This is described in the next two sections[1].

2 Rational Generating Functions

The combinatorial intuition associated with rational generating functions is easy to describe, since it essentially coincides with the notion of regular (or: rational) languages [34, 41]. It is generally believed that, if a class $\mathcal{A}$ of objects has a rational generating function, then the structure of the objects is similar to the structure of the words of a regular language. In particular, they can be constructed recursively using a finite-state automaton. Informally, we may say that

"A class of objects has a rational generating function if these objects have a *linear structure*: that is, if all objects of size n can be constructed by expanding all objects of size $n-1$ in a *finite* number of ways."

Again, this is not a complete description, since it is usually necessary to introduce several families of objects, $\mathcal{A} = \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ (one per state of the automaton).

Another way to describe this intuition is to say that the objects of $\mathcal{A}$ are in bijection with certain paths on a finite directed graph, or that they can be counted using the *transfer matrix method* [46–Ch. 4]. This intuition has proved so right in the past that I do not know of any rational generating function (counting combinatorial objects) that would not be $\mathbb{N}$-rational[2]. Moreover, it is usually rather easy to realize that a class of objects has a rational generating function (a few minutes for a well-trained combinatorialist?).

A typical example is that of integer compositions. Numerous examples are presented in [46–Ch. 4]. We shall not discuss further the rather simple case of rational generating functions. However, there are many interesting questions regarding positive rational series in several variables (see I. Gessel's lecture at the 50th Séminaire Lotharingien, available from his web page [31]).

3 Algebraic Generating Functions

By analogy with the rational case, one may think that the objects of a class $\mathcal{A}$ counted by an algebraic generating function have the same structure as the words of a non-ambiguous context-free (or: algebraic) language. In more combinatorial terms, we may say, informally again, that

[1] I do not believe there currently exists such an intuition for D-finite series.

[2] A series is $\mathbb{N}$-rational if it is the generating function of a regular language.

"A class of objects has an algebraic generating function if these objects have an *algebraic structure*: that is, if they admit a recursive description based on the *concatenation* of smaller objects of the same type."

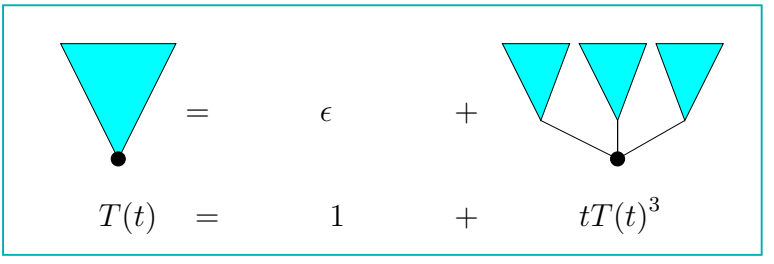


Fig. 1. Ternary trees have an algebraic generating function

A typical example is that of ternary trees, which consist of three smaller trees. This intuition was translated in the 80's into a methodology for proving the algebraicity of the generating function of some classes of objects. The principle was the following: in order to prove *in a satisfactory way* that the generating function of a class $\mathcal{A}$ of objects is algebraic, one should establish a size-preserving bijection between these objects and the words of a non-ambiguous context-free language $\mathcal{L}$. From a non-ambiguous grammar generating $\mathcal{L}$, one can then write a system of algebraic equations defining the generating function of $\mathcal{L}$, or, equivalently, of the objects of $\mathcal{A}$. This approach was called "Schützenberger's methodology" by X. G. Viennot and the Bordeaux school, and provided satisfactory explanations for the algebraicity of the generating function of numerous classes of objects [50]. In particular, it helped to clarify the algebraic nature of many families of *polyominoes* and *animals* (see Figure 2 and [23, 24, 25]). In some cases, the algebraic structure of the objects was rather clear, but in other instances, as for directed animals, it took a few years before this structure was elucidated [6, 7]. Note that, very often, the algebraic structure can be read directly from the objects: to take but a simple example, it is not necessary to encode plane trees by Dyck words

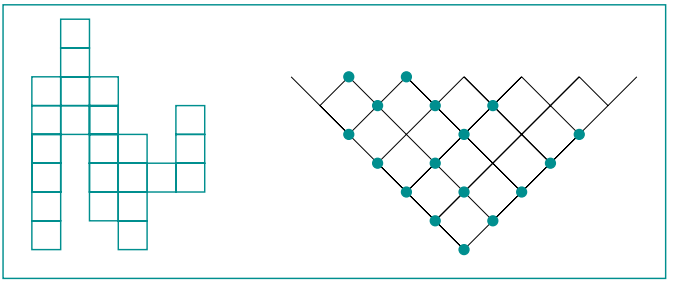


Fig. 2. Column-convex polyominoes and directed animals have an algebraic structure

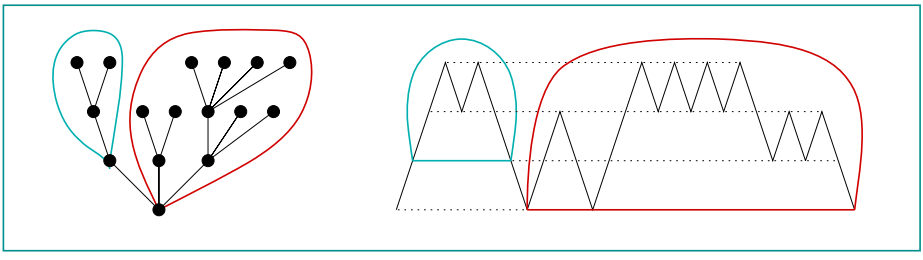


Fig. 3. The decomposition of a plane tree into two subtrees matches the decomposition of a Dyck word (or path) into two factors

to realize that their generating function is algebraic (Figure 3). This observation was formalized, many years later, into the notion of "object grammars" [26].

So, is it always true that objects with an algebraic generating function are in bijection with words of a non-ambiguous context-free language? In other words, is every algebraic generating function $\mathbb{N}$-algebraic? Well, maybe not.

Historically, one of the first combinatorial examples suggesting that things may be more tricky than in the rational case was the example of *planar maps*. A planar map is a proper embedding of a planar graph in the sphere, defined up to a continuous deformation (Figure 4). Maps are usually *rooted*, meaning that one edge is distinguished and oriented. It was proved in the early 60's by Tutte [48] that the generating function $M(t)$ of planar maps, counted by their number of edges, satisfies

$$M(t) = T(t) - tT(t)^3 \tag{2}$$

where $T(t)$ is the only formal power series in t such that

$$T(t) = 1 + 3tT(t)^2. \tag{3}$$

Tutte's result raised two questions:

– do we definitely need "minus" signs to describe algebraic generating functions arising from combinatorics? In particular, is $M(t)$ $\mathbb{N}$-algebraic?

– are there combinatorial interpretations of the above pair of equations?

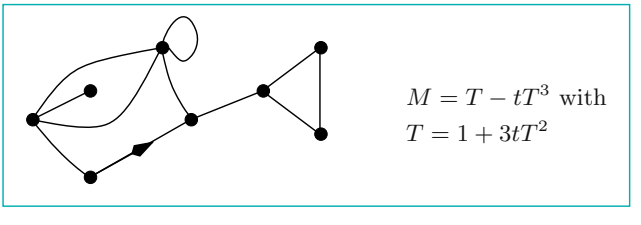


Fig. 4. Planar maps, counted by the number of edges, have an algebraic generating function

I do not know, at the moment, the answer to the first question. I would actually be happy to learn about techniques for proving that an algebraic series is, or is not, $\mathbb{N}$-algebraic. Is there a characterization of $\mathbb{N}$-algebraic series, as there is a characterization of $\mathbb{N}$-rational series [5, 45]?

The second question came from the fact that Tutte's proof did not consist of a direct combinatorial explanation of these equations. He derived them by *guessing* the solution of another functional equation that was easier to establish (see Section 5 for details). The first combinatorial explanation was given in 1980 by Cori and Vauquelin [22]. They describe a set of trees $\mathcal{T}$ naturally counted by the series $T(t)$, and a size-preserving bijection between planar maps and a subset $\mathcal{S}$ of $\mathcal{T}$. Then, they show that the trees of size n lying in $\mathcal{T} \setminus \mathcal{S}$ are in bijection with 3-tuples of trees of $\mathcal{T}$, of total size $n-1$. This explains combinatorially the system (2–3). Another, simpler explanation was given much more recently. Again, it is based on a bijection (due to Schaeffer) between planar maps and certain trees, called *balanced blossoming trees* [43, 44]. The series $T(t)$ counts all blossoming trees, and a bijective argument borrowed from [19] shows that the *unbalanced* blossoming trees are counted by $tT(t)^3$.

At this stage, we have encountered an algebraic generating function that is likely not to be $\mathbb{N}$-algebraic. This suggests that context-free languages may not encapsulate all algebraic series. Still, from a purely combinatorial point of view, this is not really annoying: the important point for combinatorialists is to be able to provide *a direct combinatorial explanation* of an algebraic system that defines the generating function.

This is, however, not the end of the story: the truth is that many classes of objects simply refuse to show their algebraic structure, even though they do have an algebraic generating function. More precisely, in the past few years, I have kept stumbling across classes of objects for which I was able to prove, with some difficulty, that the generating function is algebraic, without being able to exhibit neither a recursive construction of these objects based on concatenation, nor a bijection with other objects that were clearly algebraic. Some of the most striking examples of this type are presented in the next section.

4 Why Are These Objects "Algebraic"?

All the classes of objects listed in this section have been proved to possess an algebraic generating function by an *ad hoc* method. Unless explicitly stated, no combinatorial explanation (based on bijections and algebraic decompositions) has been given for the algebraicity of these generating functions. Several of them have, moreover, nice and simple coefficients, which are not understood combinatorially either. This raises challenging combinatorial problems.

4.1 Kreweras' Words and Walks on the Quarter Plane

Let $\mathcal{L}_0$ be the set of words u on the alphabet $\{a, b, c\}$ satisfying the following two conditions:

(i) $|u|_a = |u|_b = |u|_c$,
(ii) for every prefix v of u, $|v|_a \geq |v|_b$ and $|v|_a \geq |v|_c$.

These words encode certain walks on the plane: these walks start and end at $(0,0)$, are made of three types of steps, $a = (1,1)$, $b = (-1,0)$ and $c = (0,-1)$, and never leave the first quadrant of the plane, defined by $x, y \geq 0$. The pumping lemma [34–Theorem 4.7], applied to the word $a^n b^n c^n$, shows that the language $\mathcal{L}_0$ is not context-free. However, its generating function *is* algebraic. Denoting by $\ell_{0,0}(3n)$ the number of words of $\mathcal{L}_0$ of length $3n$, one has

$$L_0(t) = \sum_{n \geq 0} \ell_{0,0}(3n) t^{3n} = \frac{W}{2t}\left(1 - \frac{W^3}{4}\right),$$

where $W \equiv W(t)$ is the unique power series in t satisfying

$$W = t(2 + W^3).$$

Moreover, the number of such words is remarkably simple:

$$\ell_{0,0}(3n) = \frac{4^n}{(n+1)(2n+1)} \binom{3n}{n}.$$

The latter formula was proved in 1965 by Kreweras, in a fairly complicated way [36]. The algebraicity of the generating function was recognized by Gessel [32]. This rather mysterious result has attracted the attention of several combinatorialists since its publication [10, 12, 32, 38, 39].

The language $\mathcal{L}$ formed by the words satisfying condition (ii) above is not context-free either (the pumping lemma again), but it also has an algebraic generating function:

$$L(t) = 2\,\frac{(1/W - 1)\sqrt{1 - W^2}}{1 - 3t} - \frac{1}{t}.$$

More generally, let us denote by $\ell_{i,j}(n)$ the number of words u of $\mathcal{L}$ of length n such that $|u|_a - |u|_b = i$ and $|u|_a - |u|_c = j$. Define the associated three-variable generating function

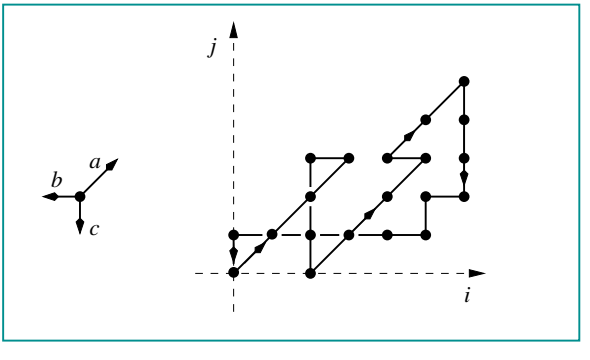


Fig. 5. Kreweras' walks in a quadrant

$$L(u,v;t) = \sum_{i,j,n} \ell_{i,j}(n) u^i v^j t^n.$$

Then

$$L(u,v;t) = \frac{(1/W - \bar{u})\sqrt{1-uW^2} + (1/W - \bar{v})\sqrt{1-vW^2}}{uv - t(u+v+u^2v^2)} - \frac{1}{uvt}$$

where $\bar{u} = 1/u$ and $\bar{v} = 1/v$.

Note that it is not true that walks in the quarter plane always have an algebraic generating function: for instance, the number of *square lattice walks* (with North, South, East and West steps) of size $2n$ that start and end at $(0,0)$ and always remain in the quarter plane is

$$\frac{1}{(2n+1)(2n+2)}\binom{2n+2}{n+1}^2 \sim \frac{4^{2n+1}}{\pi n^3},$$

and this asymptotic behaviour prevents the corresponding generating function from being algebraic. The above formula is easily proven by looking at the projections of the walk onto the horizontal and vertical axes. A bijective proof is given in [21].

4.2 Walks on the Slit Plane

Let $\mathcal{S}_0$ be the set of words u on the alphabet $\{a,b,c\}$ satisfying the following two conditions:

(i) $|u|_a = 1 + |u|_b = 1 + |u|_c$,
(ii) for every non-empty prefix v of u, if $|v|_b = |v|_c$ then $|v|_a > |v|_b$.

These words encode certain walks on the plane: these walks start at $(0,0)$, end at $(2,0)$, are made of three types of steps, $a = (2,0)$, $b = (-1,1)$ and $c = (-1,-1)$, and never hit the non-positive x-axis once they have left their starting point. We call such walks "walks on the slit plane" (Figure 6). The pumping lemma, applied to the word $b^n a^{n+1} c^n$, shows that the language $\mathcal{S}_0$ is not context-free. However, its generating function is algebraic – and even $\mathbb{N}$-algebraic. Indeed, denoting by $s_{2,0}(3n+1)$ the number of words of $\mathcal{S}_0$ of length $3n+1$, one has

$$s_{2,0}(3n+1) = \frac{4^n}{n+1}\binom{3n}{n}.$$

In other words, this number is 4^n times the number of ternary trees with n nodes, which we encountered in Example 2.

In contrast to the case of walks in the quarter plane, the algebraicity of walks on the slit plane is a robust property: that is, it is rather resistant to changes in the set of allowed steps. Take any finite set of steps $S \subset \mathbb{Z} \times \{-1,0,1\}$ (we say that these steps have *small height variations*). Let $s_{i,j}(n)$ be the number of walks of length n that start from the origin, never return to the non-positive

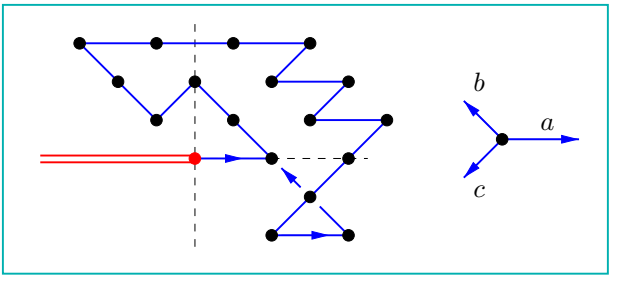


Fig. 6. A walk on the slit plane ending at $(2, 0)$

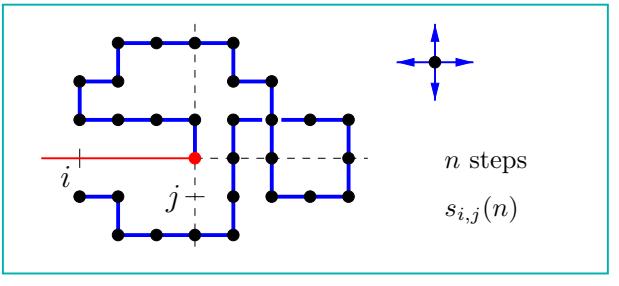


Fig. 7. A walk on the slit plane, with steps in $\{N, S, E, W\}$

horizontal axis, consist of steps of S and end at (i, j). Let $S(u, v; t)$ be the associated generating function:

$$S(u, v; t) = \sum_{i,j \in \mathbb{Z}, n \geq 0} s_{i,j}(n) u^i v^j t^n.$$

Then this series is always algebraic, as well as the series $S_{i,j}(t) := \sum_n s_{i,j}(n) t^n$ that counts walks ending at (i, j) [9, 17].

Let us consider for instance the case where S is formed of the usual square lattice steps (North, South, West and East). See Figure 7. Then

$$S(u, v; t) = \frac{\left(1 - 2t(1 + \bar{u}) + \sqrt{1 - 4t}\right)^{1/2} \left(1 + 2t(1 - \bar{u}) + \sqrt{1 + 4t}\right)^{1/2}}{1 - t(u + \bar{u} + v + \bar{v})}$$

with $\bar{u} = 1/u$ and $\bar{v} = 1/v$. Moreover, the number of walks ending at certain specific points is remarkably simple. For instance:

$$s_{1,0}(2n + 1) = C_{2n+1}, \quad s_{0,1}(2n + 1) = 4^n C_n, \quad s_{-1,1}(2n) = C_{2n}.$$

where $C_n = \binom{2n}{n}/(n + 1)$ is the celebrated nth Catalan number, which counts binary trees, Dyck words, and numerous other combinatorial objects [47–Ch. 6]. The first of these three identities has been proved combinatorially by Barcucci et al. [3]. The others still defeat our understanding.

4.3 Embedded Binary Trees

We consider here the classical binary trees, defined in much the same way as the ternary trees of Example 2: every vertex has either two ordered children (in which case it is called a node), or no child at all (in which case it is called a leaf). Let us associate with each node of a binary tree a label, equal to the difference between the number of right steps and the number of left steps one does when going from the root to the node. In other words, the label of the node is its abscissa in the natural integer embedding of the tree (Figure 8).

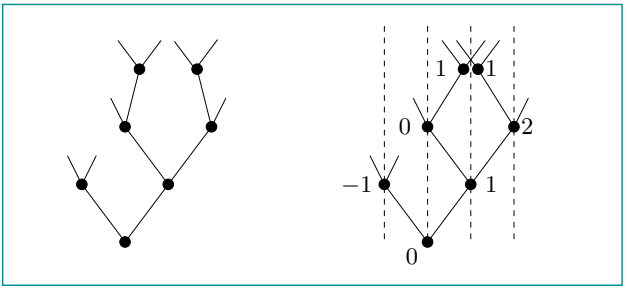


Fig. 8. The integer embedding of a binary tree

Let $S_j \equiv S_j(t,u)$ be the generating function of binary trees counted by the number of nodes (variable t) and the number of nodes at abscissa j (variable u). The standard decomposition of binary trees gives

$$\begin{aligned} S_0 &= 1 + tuS_{-1}S_1, \\ S_j &= 1 + tS_{j-1}S_{j+1} \quad \text{for } j \neq 0. \end{aligned}$$

It has be shown that for all $j \in \mathbb{Z}$, the series $S_j(t,u)$ is algebraic of degree (at most) 8 (while $S_j(t,1)$ is quadratic) [8].

Let $j \geq 1$. Setting $u = 0$ in $S_j(t,u)$, we obtain the generating function $T_{j-1}(t)$ that counts binary trees in which all nodes lie at abscissa at most $j-1$. Of course, the series T_j are algebraic too. Their degree is (at most) 2, and they admit a simple expression in terms of the series $T \equiv T(t)$ and $Z \equiv Z(t)$ defined as follows:

$$T = 1 + tT^2 \quad \text{and} \quad Z = t\,\frac{\left(1+Z^2\right)^2}{(1-Z+Z^2)}.$$

For $j \geq 0$,

$$T_j = T\,\frac{(1-Z^{j+2})(1-Z^{j+7})}{(1-Z^{j+4})(1-Z^{j+5})}.$$

Why is that so? From this, one can derive some limit results on the distribution of the number of nodes at abscissa $\lfloor \lambda n^{1/4} \rfloor$ in a random tree with n nodes [8]. These results may tell us something about the law of a "universal" random mass distribution called the integrated super-Brownian excursion [1, 37].

5 Proving the Algebraicity of a Generating Function

The first (and best) strategy for proving the algebraicity of a generating function was described and illustrated at length in Section 3. It consists of finding a recursive description of the objects based on concatenation: this gives directly a polynomial equation (or a set of polynomial equations) for their generating function. This is illustrated by Example 2 and Figure 1 (ternary trees).

A variant of this strategy consists in describing a bijection with other objects, which admit a clear "algebraic" decomposition. We have already mentioned in Section 3 the rather recent example of the enumeration of planar maps by Schaeffer via balanced blossoming trees [43, 44]. Recall that the associated decomposition involves a "minus" sign. In the past few years, this type of construction has been extended to many families of planar maps, thus providing a satisfactory explanation for the algebraicity of their generating functions [15, 16, 40].

This approach, however, has not (yet) been successful to prove any of the results stated in Section 4. Then, how did one prove these results? What can be done if one cannot discover an "algebraic structure" in the objects one is trying to count? Well, the natural strategy is to discover *any* (recursive) structure, to translate it into a functional equation for the generating function, and finally to prove that the solution of this equation *is* algebraic.

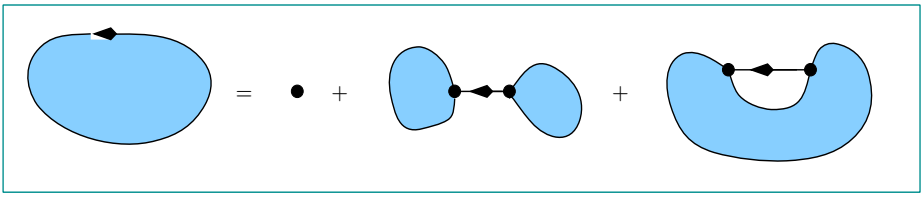


Fig. 9. Tutte's decomposition of rooted planar maps

Let us examine again the "historical" example of planar maps. Tutte established the equations of Figure 4 without giving a combinatorial explanation for them. Yet, he gave a very simple decomposition of maps (based on the deletion of the root edge, see Figure 9). But, in order to exploit this decomposition, he had to *take into account an additional parameter* in the enumeration, namely the degree of the infinite face (also called outer-degree) [48]. This forced him to introduce refined numbers $m_{n,k}$ (counting maps with n edges and outer-degree k), and a *bivariate* generating function $M(u,t) = \sum_{n,k\geq 0} m_{n,k} t^n u^k$. At this cost, it became very easy to write a functional equation defining $M(u,t)$:

$$M(u,t) = 1 + tu^2 M(u,t)^2 + t\,\frac{uM(u,t) - M(1,t)}{u-1}.$$

Tutte then proved that the series $M(1,t)$ was algebraic. This implies that $M(u,t)$ is algebraic too. He actually guessed the value of $M_1(t) := M(t,1)$, then showed

the existence of a series $M(u,t)$ that fits with $M_1(t)$ when $u=1$, and satisfies the above equation.

The above example is actually rather typical: it very often happens that it becomes much much easier to find and exploit a decomposition of the objects (even a very nave one) when taking into account one or several additional parameters. In the case of planar maps, one has to add a single parameter. For walks in the quarter plane (Section 4.1), writing an equation becomes almost trivial if one agrees to count walks ending at any point (i,j) of the quadrant, and to take into account, in the enumeration, the coordinates of this endpoint. At this cost, it becomes possible to exploit the most nave decomposition of walks one can dream of, based on the deletion of the final step. This gives, for the series $L(u,v;t)$ defined in Section 4.1:

$$L(u,v;t) = 1+tuvL(u,v;t)+\frac{t}{u}\left(L(u,v;t)-L(0,v;t)\right)+\frac{t}{v}\left(L(u,v;t)-L(u,0;t)\right) \tag{4}$$

and this equation completely defines the series $L(u,v;t)$ – but does not tell us very clearly why it is algebraic. Similarly, for the walks on the slit plane of Figure 7,

$$S(u,v;t) = 1 + t(u+v+\frac{1}{u}+\frac{1}{v})S(u,v;t) - B(1/u;t)$$

where $B(1/u;t)$ is a series in t with polynomial coefficients in $1/u$ that corresponds to the "forbidden moves", and can be described explicitly in terms of the coefficients of $S(u,v;t)$ (this description is actually not necessary to solve the equation, see [9]).

Following a terminology introduced by Zeilberger [52], we say that the variables u and v are *catalytic variables*. The above examples suggest that we have to learn which equations with catalytic variables have an algebraic solution, and how to obtain an algebraic system defining this solution. More generally, one would like to be able to *solve* such equations, whether their solution is algebraic or not...

5.1 Polynomial Equations with One Catalytic Variable

The case of one catalytic variable has now been completely clarified. Consider an equation of the form

$$P(F(u),F_1,\dots,F_k,t,u)=0 \tag{5}$$

and assume it defines uniquely a $(k+1)$-tuple $(F(u),F_1,\dots,F_k)$ of formal power series in t. Typically, $F(u)\equiv F(t,u)$ has *polynomial* coefficients in u, and $F_i \equiv F_i(t)$ is the coefficient of u^{i-1} in $F(t,u)$. In a recent work with A. Jehanne, we proved that *the solution of "every" such equation is algebraic* [18]. Moreover, a practical strategy allows one to solve specific examples (that is, to derive from (5) an algebraic equation for $F(u)$, or $F_1,\dots,F_k$). Our method extends what had been done before for linear and quadratic equations (the *kernel method [27, 2, 13]* and the *quadratic method* [20, 33–Section 2.9]). See [18] for more references and details.

These results provide a second general strategy for proving the algebraicity of the generating function of a class of objects: it suffices to establish a polynomial equation with one catalytic variable. Recent applications include the solution of a model of hard-particles on planar maps [18], and the enumeration of triangulations with high vertex degrees [4].

5.2 Linear Equations with Two Catalytic Variables

In contrast to the case of one catalytic variable, there is no hope that *all* linear equations with two catalytic variables have an algebraic solution. This is shown by the enumeration of square lattice walks constrained to stay in the first quadrant. Their three-variable generating function satisfies

$$Q(u,v;t)=1+t(u+v)Q(u,v;t)+\frac{t}{u}\left(Q(u,v;t)-Q(0,v;t)\right)+\frac{t}{v}\left(Q(u,v;t)-Q(u,0;t)\right)$$

but, as mentioned at the end of Section 4.1, the series $Q(0,0;t)$ is transcendental, which prevents the complete series $Q(u,v;t)$ from being algebraic.

To our knowledge, there is, at the moment, no way to solve systematically a linear equation with two catalytic variables. However, some principles are beginning to emerge, and have proved successful for several instances of such equations. In particular, it is shown in [10, 12] how to derive the algebraicity of the generating function of Kreweras' walks in the quarter plane from the functional equation (4). See also [10, 14] for other results on walks confined to the quarter plane, and [11] for other occurrences of such equations in the enumeration of pattern avoiding permutations. Some of the key ideas in the treatment of these equations were inspired by the book of Fayolle, Iasnogorodski and Malyshev, in which related equations are solved in a more analytic context [28].

Let us finally underline the word "linear" in the title of this subsection: we only know of *one* example of a non-linear (but polynomial) equation with two catalytic variables that has been solved. It is related to the enumeration of planar triangulations, weighted by their chromatic polynomial. It took Tutte ten years and ten papers to solve the equation he had established in 1973. The solution turned out to be *differentially algebraic*, meaning that it satisfies a polynomial differential equation

$$P(A(t), A'(t), A''(t), t) = 0.$$

See [49] for a summary of this *tour de force*.

6 Concluding Remarks and Questions

Some of the questions below may have been solved already, or be simple to solve.

Regarding $\mathbb{N}$-Algebraic Series
It is still an open question to know whether all (combinatorial) generating functions that are rational (resp. algebraic) are actually $\mathbb{N}$-rational (resp. $\mathbb{N}$-algebraic). It seems that the answer could be yes in the first case, and no in

the second. The generating function for planar maps (Figure 4) is possibly not $\mathbb{N}$-rational.

How can one decide whether an algebraic series with positive coefficients is $\mathbb{N}$-algebraic, or not? What about the singularities of $\mathbb{N}$-algebraic series? What about the asymptotic behaviour of their coefficients? Can we find context-free, non-ambiguous languages $\mathcal{L}$ such that the number of words of length n in $\mathcal{L}$ grows like $\alpha\mu^n n^\gamma$, for any $\gamma \in \mathbb{Q} \setminus \{-1, -2, \ldots\}$?

Regarding D-Finite Series
Regular languages, and context-free languages, fit well with the first two steps of our hierarchy of formal power series, namely rational series and algebraic series. Is there somewhere a well-polished class of languages that would fit with the third step of our hierarchy, namely the class of D-finite series defined in Section 1? Recall that a series $A(t) = \sum_n a_n t^n$ is D-finite if and only if its coefficients satisfy a linear recurrence relation with polynomial coefficients:

$$P_0(n)a_n + P_1(n)a_{n-1} + \cdots + P_k(n)a_{n-k} = 0$$

for n large enough.

Acknowledgements. I am very grateful to Frédérique Bassino and Mark Dukes, whose extremely valuable comments helped improving a preliminary version of this paper.

References

1. D. Aldous. Tree-based models for random distribution of mass. *J. Statist. Phys.*, 73(3-4):625–641, 1993.
2. C. Banderier, M. Bousquet-Mélou, A. Denise, P. Flajolet, D. Gardy, and D. Gouyou-Beauchamps. Generating functions for generating trees. *Discrete Math.*, 246(1-3):29–55, 2002.
3. E. Barcucci, E. Pergola, R. Pinzani, and S. Rinaldi. A bijection for some paths on the slit plane. *Adv. in Appl. Math.*, 26(2):89–96, 2001.
4. O. Bernardi. On triangulations with high vertex degrees. Submitted, 2004.
5. J. Berstel. Sur les pôles et le quotient de Hadamard de séries $\mathbb{N}$-rationnelles. *C. R. Acad. Sci. Paris Sér. A-B*, 272:A1079–A1081, 1971.
6. J. Bétréma and J.-G. Penaud. Animaux et arbres guingois. *Theoret. Comput. Sci.*, 117(1-2):67–89, 1993.
7. J. Bétréma and J.-G. Penaud. Modèles avec particules dures, animaux dirigés et séries en variables partiellement commutatives. Technical report, LaBRI, Université Bordeaux 1, 1993. arXiv:math.CO/0106210.
8. M. Bousquet-Mélou. Limit results for embedded trees. Applications to the integrated super-Brownian excursion. In preparation.
9. M. Bousquet-Mélou. Walks on the slit plane: other approaches. *Adv. in Appl. Math.*, 27(2-3):243–288, 2001.
10. M. Bousquet-Mélou. Counting walks in the quarter plane. In *Mathematics and computer science* 2*, (Versailles,* 2002*)*, Trends Math., pages 49–67. Birkhäuser, Basel, 2002.

11. M. Bousquet-Mélou. Four classes of pattern-avoiding permutations under one roof: generating trees with two labels. *Electronic J. Combinatorics*, 9(2):Research Paper 19, 2003.
12. M. Bousquet-Mélou. Walks in the quarter plane: Kreweras' algebraic model. *Ann. Appl. Proba.*, to appear.
13. M. Bousquet-Mélou and M. Petkovšek. Linear recurrences with constant coefficients: the multivariate case. *Discrete Math.*, 225(1-3):51–75, 2000.
14. M. Bousquet-Mélou and M. Petkovšek. Walks confined in a quadrant are not always D-finite. *Theoret. Comput. Sci.*, 307:257–276, 2003.
15. M. Bousquet-Mélou and G. Schaeffer. Enumeration of planar constellations. *Adv. in Appl. Math.*, 24(4):337–368, 2000.
16. M. Bousquet-Mélou and G. Schaeffer. The degree distribution of bipartite planar maps: applications to the Ising model. ArXiv math.CO/0211070, 2002.
17. M. Bousquet-Mélou and G. Schaeffer. Walks on the slit plane. *Probab. Theory Related Fields*, 124(3):305–344, 2002.
18. M. Bousquet-Mélou and A. Jehanne. Planar maps and algebraic series: a generalization of the quadratic method. In preparation.
19. J. Bouttier, P. Di Francesco, and E. Guitter. Census of planar maps: from the one-matrix model solution to a combinatorial proof. *Nuclear Phys. B*, 645(3):477–499, 2002.
20. W. G. Brown. On the existence of square roots in certain rings of power series. *Math. Ann.*, 158:82–89, 1965.
21. R. Cori, S. Dulucq, and G. Viennot. Shuffle of parenthesis systems and Baxter permutations. *J. Combin. Theory Ser. A*, 43(1):1–22, 1986.
22. R. Cori and B. Vauquelin. Planar maps are well labeled trees. *Canad. J. Math.*, 33(5):1023–1042, 1981.
23. M.-P. Delest. Generating functions for column-convex polyominoes. *J. Combin. Theory Ser. A*, 48(1):12–31, 1988.
24. M.-P. Delest, D. Gouyou-Beauchamps, and B. Vauquelin. Enumeration of parallelogram polyominoes with given bond and site perimeter. *Graphs Combin.*, 3(4):325–339, 1987.
25. M.-P. Delest and G. Viennot. Algebraic languages and polyominoes enumeration. *Theoret. Comput. Sci.*, 34(1-2):169–206, 1984.
26. I. Dutour. *Grammaires d'objets : énumération, bijections et génération aléatoire.* PhD thesis, Université Bordeaux 1, France, 1996.
27. G. Fayolle and R. Iasnogorodski. Two coupled processors: the reduction to a Riemann-Hilbert problem. *Z. Wahrsch. Verw. Gebiete*, 47(3):325–351, 1979.
28. G. Fayolle, R. Iasnogorodski, and V. Malyshev. *Random walks in the quarter-plane: Algebraic methods, boundary value problems and applications*, volume 40 of *Applications of Mathematics*. Springer-Verlag, Berlin, 1999.
29. P. Flajolet. Analytic models and ambiguity of context-free languages. *Theoret. Comput. Sci.*, 49(2-3):283–309, 1987.
30. R. Flajolet and R. Sedgewick. Analytic combinatorics: functional equations, rational, and algebraic functions. Technical Report RR4103, INRIA, 2001. A component of the book project "Analytic Combinatorics". Available at http://www.inria.fr/rrrt/rr-4103.html.
31. I. Gessel. Rational functions with nonnegative power series coefficients. 50th Séminaire Lotharingien de Combinatoire, Ottrott, France, March 2003.
32. I. M. Gessel. A probabilistic method for lattice path enumeration. *J. Statist. Plann. Inference*, 14(1):49–58, 1986.

33. I. P. Goulden and D. M. Jackson. *Combinatorial enumeration*. John Wiley & Sons Inc., New York, 1983. Wiley-Interscience Series in Discrete Mathematics.
34. J. E. Hopcroft and J. D Ullman. *Formal languages and their relation to automata*. Addison-Wesley, 1969.
35. E. L. Ince. *Ordinary Differential Equations*. Dover Publications, New York, 1944.
36. G. Kreweras. Sur une classe de problèmes liés au treillis des partitions d'entiers. *Cahiers du B.U.R.O.*, 6:5–105, 1965.
37. J.-F. Marckert and S. Janson. Convergence of discrete snakes. Technical report, Université de Versailles-Saint-Quentin, 2003.
38. H. Niederhausen. Sheffer polynomials in path enumeration. In *Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing (Humboldt State Univ., Arcata, Calif.,* 1979*)*, Congress. Numer., XXVI, pages 281–294, 1980.
39. H. Niederhausen. The ballot problem with three candidates. *European J. Combin.*, 4(2):161–167, 1983.
40. D. Poulalhon and G. Schaeffer. A bijection for triangulations of a polygon with interior points and multiple edges. *Theoret. Comput. Sci.*, 307(2):385–401, 2003.
41. A. Salomaa and M. Soittola. *Automata-theoretic aspects of formal power series*. Springer-Verlag, New York, 1978. Texts and Monographs in Computer Science.
42. B. Salvy and P. Zimmermann. Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, 20(2):163–177, 1994.
43. G. Schaeffer. Bijective census and random generation of Eulerian planar maps with prescribed vertex degrees. *Electron. J. Combin.*, 4(1):Research Paper 20, 14 pp. (electronic), 1997.
44. G. Schaeffer. *Conjugaison d'arbres et cartes combinatoires aléatoires*. PhD thesis, Université Bordeaux 1, France, 1998.
45. M. Soittola. Positive rational sequences. *Theoret. Comput. Sci.*, 2(3):317–322, 1976.
46. R. P. Stanley. *Enumerative combinatorics. Vol. 1*, volume 49 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1997.
47. R. P. Stanley. *Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999.
48. W. T. Tutte. On the enumeration of planar maps. *Bull. Amer. Math. Soc.*, 74:64–74, 1968.
49. W. T. Tutte. Chromatic sums revisited. *Aequationes Math.*, 50(1–2):95–134, 1995.
50. G. Viennot. Enumerative combinatorics and algebraic languages. In *Fundamentals of computation theory (Cottbus, 1985)*, volume 199 of *Lecture Notes in Comput. Sci.*, pages 450–464. Springer, Berlin, 1985.
51. J. Wimp and D. Zeilberger. Resurrecting the asymptotics of linear recurrences. *J. Math. Anal. Appl.*, 111(1):162–176, 1985.
52. D. Zeilberger. The umbral transfer-matrix method: I. Foundations. *J. Comb. Theory, Ser. A*, 91:451–463, 2000.

Algorithmics in Exponential Time

Uwe Schöning

Abteilung Theoretische Informatik,
Universität Ulm,
89069 Ulm, Germany
schoenin@informatik.uni-ulm.de

Abstract. Exponential algorithms, i.e. algorithms of complexity $O(c^n)$ for some $c > 1$, seem to be unavoidable in the case of NP-complete problems (unless P=NP), especially if the problem in question needs to be solved exactly and not approximately. If the constant c is close to 1 such algorithms have practical importance. Deterministic algorithms of exponential complexity usually involve some kind of backtracking. The analysis of such backtracking algorithms in terms of solving recurrence equations is quite well understood. The purpose of the current paper is to show cases in which the constant c could be significantly reduced, and to point out that there are some randomized exponential-time algorithms which use randomization in some new ways. Most of our examples refer to the 3-SAT problem, i.e. the problem of determining satisfiability of formulas in conjunctive normal form with at most 3 literals per clause.

1 Why Exponential-Time Algorithms?

Unless P=NP, exponential (or at least non-polynomial) algorithms are unavoidable for NP-complete problems such as SAT or 3-SAT. This is especially so if the problem in question needs to be solved or decided exactly, and when there is no use of any type of approximation algorithm.

Having accepted that we have to deal with exponential-time algorithms only, it makes sense to improve the relative efficiency of our algorithms by reducing the value of the base constant c in the exponential time bound $O(c^n)$. New algorithms which are able to reduce the constant c can mean an tremendous improvement. Moving from an algorithm with complexity $O(c^n)$ to another, better one, with complexity $O(d^n)$ where $d = \sqrt{c}$ means that, within the same given time limit, the input size that can be solved by the new algorithm, doubles. In concrete terms, improving the brute-force algorithm for 3-SAT of complexity $O(2^n)$ (where n is the number of Boolean variables) to another one of complexity $O(1.324^n)$ (as it was indeed the case within the last years [14, 17, 10]). This improvement allows to increase the number of tractable Boolean variables by a factor of more than 2.4 .

Another motivating example are SAT-solvers which are extremely useful general-purpose programs which are operating in worst-case exponential-time.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 36–43, 2005.

2 Backtracking

Most deterministic exponential-time algorithms use some version of backtracking. For example, one can try out all potential 2^n many assignments for a Boolean formula in n variables in a backtracking manner. In each level of the recursive backtrack tree one more variable is assigned one of the two Boolean values true or false. This direct approach leads to the recursion $T(n) \leq 2 \cdot T(n-1)$, giving the complexity $T(n) = O(2^n)$.

More sophisticated approaches can reduce the number of variables in the recursive procedure calls not only from n to $n-1$, but to $n-2$, $n-3$, etc. The 3-SAT algorithm of Monien and Speckenmeyer [11], for example, first chooses a shortest clause in the formula which, in the worst case, has 3 literals, say x, y, z. The first recursive call of the procedure assigns $x = 1$ and simplifies the formula accordingly. If this recursive call is not successful (no satisfying assignment is found), then the partial assignment ($x = 0$, $y = 1$) is tried next. If again unsuccessful, the last recursive call is done with respect to the partial assignment ($x = 0$, $y = 0$, $z = 1$). This leads to the recursion $T(n) \leq T(n-1) + T(n-2) + T(n-3)$ with the solution $T(n) = O(1.84^n)$.

This algorithm has been trimmed even more in [11]. Using the concept of autark partial assignments, the new recursive tree structure leads to the equations $T(n) \leq T'(n-1) + T'(n-2) + T'(n-3)$, $T'(n) \leq \max\{T(n-1), T'(n-1) + T'(n-2)\}$ having the solution $T(n) = O(1.619^n)$.

Consider as another example the problem of 3-coloring a given graph with n vertices. The very naive approach to solve this problem is to assign to each vertex independently one of the 3 colors and try all cases in brute-force manner. This leads to an $O(3^n)$ algorithm. The next (still) naive approach is to start in a backtracking way with a first node, assign it color 1, then backtrack through all its neighbors and try out all 2 remaining colors. Continue recursively by considering the neighbor vertices which have not been colored yet and try both remaining possibilities. This leads to a $O(2^n)$ algorithm. The next option is to notice that one of the 3 colors cannot occur more often than $\binom{n}{n/3}$ many times. Hence, we systematically assign color 1 to at most $\binom{n}{n/3}$ vertices in the graph. These are

$$\sum_{i=0}^{n/3} \binom{n}{i} \leq 2^{h(1/3)n} \leq 1.89^n$$

many posssibilities (where h is the entropy function [2].) In all those cases where there is no edge between any two vertices with color 1, the remaining graph needs to be 2-colored. Whether this is possible can be determined in polynomial time. Therefore, this procedure has complexity $O(1.89^n)$.

3 Local Search

Local Search starts with a given assignment. If this assignment is not yet a solution (i.e. a satisfying assignment in case of the SAT problem), the actual

assignment is modified. These modifications are typically very local, for example, a 1 bit flip is performed (one variable which has the actual value true is set to false, or vice versa). These modifications (or "mutations" in evolutionary algorithms terminology) are either done in a random or some deterministic, systematic fashion. The better algorithms of this kind use in some sense the (mis-)behavior of the actual assignment on the input to get some kind of "hint" what modification might be useful and will probably lead in the right direction. "Leading in the right direction" can be quantified in terms of the Hamming distance between the actual assignment and some fixed solution assignment (i.e. satisfying assignment).

In [16, 7, 8] a deterministic backtracking-like recursive procedure *search(F,a,d)* is used for solving 3-SAT. This procedure returns true, if there exists a satisfying assignment a^* for the Boolean formula F which is within Hamming distance at most d from the given initial assignment a. The Hamming distance between two bit vectors of equal length is the number of bits in which they differ. This procedure *search* works as follows. First it checks whether one of the trivial cases occurs. If for example a already satisfies F (leading to the returned value true), or else, if $d = 0$ (leading to the returned value false). If these cases do not occur, then a does not satisfy F and $d > 0$. Since a does not satisfy F there is a clause C in F which is not satisfied by a. All ≤ 3 literals in C are set to false by a. Under the satisfying assignment a^* that we are looking for, at least one of these literals must be set to true. Hence, if we flip the value of one of the variables in C, we make the Hamming distance between a and a^* smaller by 1. Therefore, we perform 3 recursive calls, in each recursive call one of the bits in the assignment a which corresponds to a variable in C is flipped. Further, the parameter d is reduced to $d - 1$. It is clear that this procedure needs time $T(d) \leq 3 \cdot T(d-1)$, hence $T(d) = O(3^d)$ (ignoring polynomial factors). Now we have to discuss the question, how the initial assignments a come about. The easiest case is that we take just two initial assignments 0^n and 1^n and set $d = n/2$. This gives us already a very simple 3-SAT algorithm of complexity $O(3^{n/2}) = O(1.74^n)$.

Next we push this further, and choose the initial assignments systematically from some precomputed list $L = \{a_1, a_2, a_3, \ldots, a_t\}$ where t is an exponential function in n, the number of variables. This list L, together with some suitable chosen Hamming distance d, should be a so called covering code [6], i.e. *every* bit vector of length n should be within Hamming distance at most d to at least one $a_i \in L$. It turns out that the optimal choice for d is $n/4$, and $t = 1.14^n$ (together with an appropriate choice of the a_i). This gives the overall complexity bound $t \cdot 3^d = 1.14^n \cdot 3^{n/4} = 1.5^n$.

In [7, 8] it is shown how the procedure *search* can be further modified such that the complexity of searching up to Hamming distance d can be estimated as $T(d) \leq 6 \cdot T(d-2) + 6 \cdot T(d-3)$ with the solution $T(d) = O(2.85^d)$. Using this bound and modifying the choices of d and t to $d = 0.26n$ and $t = 1.13^n$ gives the overall complexity bound $t \cdot 2.85^d = 1.13^n \cdot 2.85^{0.26n} = 1.481^n$. (This was recently improved to 1.473^n [5].)

4 Randomization: Reducing to a Polynomial-Time Case

Up to now, we have considered deterministic strategies which are able to reduce the exponential constant c as compared to the direct brute-force strategy (which usually means that the constant is $c = 2$). Randomization can help in various ways to improve such algorithms. Many randomized algorithms are superior to their deterministic competitors.

Often, NP-complete problems have versions which are efficiently solvable (like 2-SAT or 2-colorability). A random choice can reduce a difficult problem to an easy one if some of the parameters (values of variables) are chosen and assigned at random. Of course, by assigning some variable or problem parameter at random in a wrong way, we might loose the chance of finding any solution afterwards. Such random restrictions have to be repeated a certain number of times. The repetition number is determined by the reciprocal of the success probability of one such random restriction. Of course, this repetition number has to be taken into account within the overall complexity estimation. In other words, if we have some randomized algorithm which has complexity t and success probability p (both depending on n), then this algorithm needs to be repeated $c \cdot p^{-1}$ many times to achieve an error probability of at most $(1-p)^{c/p} \leq e^{-c}$ so that the overall complexity becomes $c \cdot t \cdot p^{-1}$.

As an example, in [4] a method for solving the 3-coloring problem is described. For each of the n vertices decide at random which one of the 3 colors should *not* be placed on this vertex. Each of these random decision can be wrong with probability 1/3, and all decisions are consistent with a potential 3-coloring with probability $p = (2/3)^n$. Given such an assignment of 2 possible colors to each of the n vertices of a graph, it is possible to decide whether such a coloring exists in polynomial time, e.g. by reducing the problem to a 2-SAT problem which can be solved in polynomial time [1]. That is, in this case t is a polynomial in n, and the overall complexity for solving 3-coloring becomes some polynomial times $(3/2)^n$.

5 Randomization: Permuting the Evaluation Order

In some cases, the complexity of an algorithm can depend very much on the order in which the variables (or the possible values of the variables) are processed. To escape the worst case input situations, it might be a good idea to choose the order in a random way. A nice example of this strategy can be found in [15]: A game tree evaluation can be considered as evaluating a quantified Boolean formula of the form

$$\exists x_1 \, \forall x_2 \, \exists x_3 \, \forall x_4 \ldots Q_n x_n \, F(x_1, x_2, \ldots, x_n)$$

Each variable x_i can take the possible values true or false. It is standard to evaluate such expressions in a backtracking manner. The recursion tree can be interpreted as an AND-OR tree. Evaluating an OR (or existential quantifier)

can turn out to be rather quickly done if the first subtree returns the value true. In this case there is no need to evaluate the second subtree; the result will be the truth value true. Obviously, we do not know beforehand which (if any) of the two subtrees might return true. Now let us decide at random which of the two arguments to evaluate first. If one of the subtrees has value true, then with probability at least 1/2 we select this subtree first and the evaluation of the OR is fast. Even in the case that both subtrees have the truth value false, there is some advantage, namely, in the AND-level (or universal quantifier) which is immediately following, we know now that the result value is false, which means that at least one argument is false. Again (by dual reasoning) this is a good situation when evaluating an AND. This leads to the following recursion for the expected number of recursive calls: $T(n) \leq S(n-1) + \frac{1}{2} \cdot T(n-1)$, $S(n) = 2 \cdot T(n-1)$, giving the complexity bounds $T(n) = O(1.69^n)$ and $S(n) = O(1.69^n)$.

Another such random permuting idea can be found in the 3-SAT algorithm of Paturi et al [13, 14]. Here the variable order is chosen at random. Now, given a particular order of the Boolean variables, one variable after the other (according to that order) is assigned a truth value. In the regular case, this is done at random (with probability 1/2 this choice might be wrong and will not lead to a satisfying assignment), or after having substituted the partial assignment so far into the formula (and simplified) it sometimes turns out that the next variable to be assigned according to the order forms a unit clause in the (remaining) formula. In this case the variable is assigned such that this unit clause becomes true. Assuming that the previous random choices were correct, this deterministic setting of the variable is also correct. It turns out, that in this (simple version of the) algorithm, on the average, taken over all $n!$ permutations of the variables, just $2n/3$ many random choices are needed. Consequently, the probability for finding a satisfying assignment this way is $2^{-2n/3}$, and the overall complexity bound is $O(2^{2n/3}) = O(1.58^n)$.

Intuitively, one might say that after substituting those partial assignments into the formula, the (resulting) formula occasionally gives a 1-bit "hint" what the next assignment should be. In the second paper [14] the authors show that it is possible to do some preprocessing with the formula F which is transformed to F', a formula having more clauses, but the same number of variables, and being equivalent to the original formula F. The advantage of using F' instead of F is that F' is likely to give more such 1-bit hints, on the average, and therefore, more random bits can be saved. The analysis in [14] shows that this is an $O(1.36^n)$ algorithm for 3-SAT.

6 Randomized Local Search

We already discussed a way of performing a deterministic local search, starting from some initial assignment a, to find out whether there is a satisfying assignment a^* within Hamming distance at most d from a. Instead of flipping each value of the 3 variables systematically in a backtracking fashion as described in Section 3, we now just choose one these variables at random and flip its value.

(This idea appears already in the randomized $O(n^2)$ 2-SAT algorithm of Papadimitriou [12]. Applying and analyzing this principle in the context of 3-SAT was done in [17, 18].) Of course, the probability of choosing the correct variable to flip can be as low as 1/3. Therefore, the probability of finding a satisfying assignment which is by Hamming distance d away from the initial assignment, is just $(1/3)^d$. But now we modify this approach by letting the flipping process last for $3d$ steps instead of just d steps. Even if we have d "mistakes" (within the $3d$ steps) by flipping variables which have already the correct value, there is still a chance that we correct these mistakes (by another d steps) and further on do the necessary corrections (by another d steps) to make the assignment equal to the satisfying assignment a^*. There are $\binom{3d}{d}$ possibilities of this situation happening, and each of these has probability $(2/3)^d \cdot (1/3)^{2d}$, resulting in a success probability of $\binom{3d}{d} \cdot (2/3)^d \cdot (1/3)^{2d} = (1/2)^d$ (up to some small polynomial factor). Compare with the $(1/3)^d$ from above. This means, if we just select a random initial assignment and then perform a random local search (as described) for $3n$ steps, we get a success probability (probability of finding a satisfying assignment - if one exists) of $E[(1/2)^D]$. Here D is the random variable that indicates the Hamming distance of the initial assignment to some fixed satisfying assignment.

If we produce the initial assignment absolutely at random – without referring to the input formula F – then this expectation becomes $E[(1/2)^{X_1+\ldots+X_n}] = \prod_{i=1}^{n} E[(1/2)^{X_i}] = (3/4)^n$. Here the X_i are 0-1-valued random variables which indicate whether there is a difference in the i-th bit. Therefore, we get the complexity bound $O((4/3)^n)$ (ignoring polynomial factors).

In the paper by Iwama and Tamaki [10] this random walk algorithm is combined with the $O(1.36^n)$-algorithm [14] from above. Both algorithms work on the same random initial assignment. This results in the fastest algorithm for 3-SAT known so far with a complexity bound of $O(1.324^n)$ (where "fastest" refers to the fact that there is a rigorous proof for the upper bound – of course there are numerous other algorithms in the SAT-solver scene using all kind of heuristics and different approaches. Doing performance test on benchmark formulas and random formulas is, of course, serious and important research, but usually leaves us without a proven worst-case upper bound.)

The same principle strategy can be applied to the 3-colorability problem. First, guess an initial coloring at random. Whenever there is an edge with its both vertices of the same color, select one of the vertices at random and change its color to one of the two remaining colors at random. Repeat this "repair" process for $3n$ steps. The probability of success for one such run turns out to be $(2/3)^n$. Therefore the complexity bound is $O(1.5^n)$ (ignoring polynomial factors), when the basis algorithm is repeated this number of times.

7 Randomization with Biased Coins

In the algorithm of the last section, we used the structure of the formula to lead the search process. But on the other hand, when producing the initial assignment we ignored the input formula completely. The question is now whether the input

formula can be used to lead the stochastic choice of the initial assignment. We present some ideas from [3, 9].

The first thing we do is to collect as many mutually variable-disjoint clauses having 3 literals of as possible, by some greedy strategy. Suppose the number m of clauses collected this way is small (say $m < 0.1469n$). Then we can cycle through all potential 7^m assignments of these variables. After plugging in these partial assignments, the rest of the formula becomes a 2-CNF formula, and its satisfiability can be determined in polynomial time. Hence, in this case, we stay below the total complexity $O(1.331^n)$.

If the number of mutually disjoint clauses is larger than $0.1469n$, then we assign each of the 3 literals of these clauses at random an initial assignment in a particular biased way: with probability $3p$ exactly one of the 3 literals is assigned the value true (where each of the 3 literals gets the same chance p), With probability $3q$ we assign exactly two of the 3 literals in the clause the value true. With probability $1 - 3p - 3q$ assign all three literals the value true. Good values for p and q will be determined later.

Supposing that the satisfying assignment a^* assigns the three literals x, y, z the values $1, 0, 0$, we get that the Hamming distance between the initial assignment a and a^*, as a random variable, can have the value 0 (with probability p), the value 1 (with probability $2q$), the value 2 (with probability $1 - p - 3q$), and the value 3 (with probability q). Hence, the expectation of the random variable $(1/2)^D$ where D is the Hamming distance on these 3 variables, is

$$1 \cdot (p) + \frac{1}{2} \cdot (2q) + \frac{1}{4} \cdot (1 - p - 3q) + \frac{1}{8} \cdot q \;=\; (1/4 + 3p/4 + 3q/8)$$

For a satisfying assignment which assigns to x, y, z the values $1, 1, 0$ or the values $1, 1, 1$ similar calculations can be done. If the number m of of mutually disjoint clauses splits into $m = m_1 + m_2 + m_3$ where m_i is the number of clauses which have exactly i literals true under assignment a^*, we get for the overall expectation of $(1/2)^{\textit{Hamming-distance}}$, and hence for the success probability, the term

$$(1/4 + 3p/4 + 3q/8)^{m_1} \cdot (1/2 - 3p/8)^{m_2} \cdot (1 - 9p/4 - 3q/2)^{m_3} \cdot (3/4)^{n-3m}$$

Now, either we determine p and q in such a way that all three exponential bases become identical (giving $p = 4/21$ and $q = 2/21$), or we compute for each of the (polynomially many!) choices for m_1, m_2, m_3 individually a good choice for p and q which minimizes the complexity. The latter strategy is somewhat better, both are close to $O(1.331^n)$.

References

1. B. Aspvall, M.F. Plass, and R.E. Tarjan: A linear time algorithm for testing the truth of certain quantified Boolean formulas, *Information Processing Letters* 8(3) (1979) 121–123.
2. R.B. Ash: *Information Theory.* Dover 1965.

3. S. Baumer and R. Schuler: Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs, In *Theory and Applications of Satisfiability Testing*, SAT 2003, Lecture Notes in Computer Science, Vol. 2919, 150–161, 2004.
4. R. Beigel and D. Eppstein: 3-coloring in time O(1:3446 n): a no-MIS algorithm. In *Proceedings of the 36nd Annual IEEE Symposium on Foundations of Computer Science*, IEEE 1995, pages 444-452.
5. T. Brueggemann and W. Kern: *An improved local search algorithm for 3-SAT*. Memorandum No. 1709, Department of Applied Mathematics, University of Twente, 2004.
6. G. Cohen, I. Honkala, and S. Litsyn, A. Lobstein: *Covering Codes*. North-Holland 1997.
7. E. Dantsin, A. Goerdt, E.A. Hirsch, and U. Schöning: Deterministic algorithms for k-SAT based on covering codes and local search. *Proc. 27th International Colloquium on Automata, Languages and Programming 2000*. Springer Lecture Notes in Computer Science, Vol. 1853, pages 236–247, 2000.
8. E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning: A deterministic $(2 - \frac{2}{k+1})^n$ algorithm for k-SAT based on local search. To appear in *Theoretical Computer Science*.
9. T. Hofmeister, U. Schöning, and R. Schuler, O. Watanabe: *A probabilistic 3-SAT algorithm further improved*. In *Proc. of the 19th Sympos. on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science, Vol. 2285, 193–202, 2002.
10. K. Iwama and S. Tamaki: Improved upper bounds for 3-SAT. *Electronic Colloquium on Computational Complexity*. Report No. 53, 2003. Also: *Proceedings of the fifteenth annual ACM-SIAM Symposium on Discrete algorithms*, ACM, 2004, pp. 328–328.
11. B. Monien, E. Speckenmeyer: Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics* 10 (1985) 287–295.
12. C.H. Papadimitriou: On selecting a satisfying truth assignment. *Proceedings of the 32nd Ann. IEEE Symp. on Foundations of Computer Science*, pp. 163–169, 1991.
13. R. Paturi, P. Pudlák, and F. Zane: Satisfiability coding lemma. *Proceedings 38th IEEE Symposium on Foundations of Computer Science* 1997, 566–574.
14. R. Paturi, P. Pudlák, M.E. Saks, and F. Zane: An improved exponential-time algorithm for k-SAT. *Proceedings 39th IEEE Symposium on Foundations of Computer Science* 1998, pp. 628–637.
15. M. Saks and A. Wigderson: Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*. IEEE Computer Science Press, 1986, pp. 29-38.
16. U. Schöning: *On The Complexity of Constraint Satisfaction Problems*. Ulmer Informatik-Berichte, Nr. 99-03. Universität Ulm, Fakultät für Informatik, 1999.
17. U. Schöning: A probabilistic algorithm for k-SAT and constraint satisfaction problems. *Proceedings 40th IEEE Symposium on Foundations of Computer Science* 1999, 410–414.
18. U. Schöning: A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica* 32,4 (2002) 615–623.

Worst-Case and Average-Case Approximations by Simple Randomized Search Heuristics

Carsten Witt*

FB Informatik, LS 2
Univ. Dortmund, 44221 Dortmund, Germany
carsten.witt@cs.uni-dortmund.de

Abstract. In recent years, probabilistic analyses of algorithms have received increasing attention. Despite results on the average-case complexity and smoothed complexity of exact deterministic algorithms, little is known about the average-case behavior of randomized search heuristics (RSHs). In this paper, two simple RSHs are studied on a simple scheduling problem. While it turns out that in the worst case, both RSHs need exponential time to create solutions being significantly better than 4/3-approximate, an average-case analysis for two input distributions reveals that one RSH is convergent to optimality in polynomial time. Moreover, it is shown that for both RSHs, parallel runs yield a PRAS.

1 Introduction

It is widely acknowledged that worst-case analyses may provide too pessimistic estimations for the runtime of practically relevant algorithms and heuristics. Therefore, in recent years, there has been a growing interest in the probabilistic analysis of algorithms. Famous examples include results on the average-case time complexity of a classical algorithm for the knapsack problem [1] and of the simplex algorithm [2]. Both papers show a polynomial runtime in the even stronger model of so-called smoothed complexity.

Approximation is another way out of this worst-case way of thinking. It is well known that many NP-hard problems allow polynomial-time approximation algorithms or even approximation schemes [3]. However, if even no approximation algorithms are available, one often resorts to heuristic approaches, which are said to provide good solutions within a tolerable span of time. Such approaches may be the only choice if there are not enough resources (time, money, experts, ...) available to design problem-specific (approximation) algorithms.

Many general-purpose heuristics such as the Metropolis algorithm or Simulated Annealing [4] rely on the powerful concept of randomization. Another popular class of randomized search heuristics (RSHs) is formed by the so-called

* The author was supported by the Deutsche Forschungsgemeinschaft (DFG) as a part of the Collaborative Research Center "Computational Intelligence" (SFB 531).

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 44–56, 2005.

Evolutionary Algorithms (EAs), see [5]. Despite having been applied successfully for more than 30 years, a theoretical foundation of the computational time complexity of EAs has started only recently, see, e. g., [6, 7, 8, 9, 10].

However, almost all results on the time complexity of RSHs are concerned with exact optimization. Moreover, these results mostly refer to worst-case instances from the class of problems considered. In contrast, the real aims of heuristics are approximation and efficient average-case behavior. Therefore, we should consider these aspects when studying general-purpose heuristics such as EAs. Positive and negative results will help to understand under what circumstances such heuristics can be efficient (approximation) algorithms and to provide guidelines for the practitioner when and how to apply them. Our approach starts by investigating RSHs on well-studied combinatorial problems. Such analyses have already been carried out in the context of exact optimization, e. g., [9, 10]. Of course, our goal is not to compare RSHs with clever problem-specific algorithms.

In this paper, we consider two simple RSHs for a well-known optimization problem, namely the optimization variant of the NP-complete PARTITION problem: Given n positive integers $w_1, \ldots, w_n$, find some subset $I \subseteq \{1, \ldots, n\}$ such that $m(I) := \max\left\{\sum_{i \in I} w_i, \sum_{i \notin I} w_i\right\}$ becomes minimal. This is one of the easiest-to-state and easiest-to-solve NP-hard problems since it even allows an FPAS [3]; from a practical point of view, it may be regarded as a simple scheduling problem. In fact, there already are some average-case analyses of classical greedy heuristics designed for this problem [11, 12]. We will relate these results to those for the general-purpose RSHs considered by us.

Since the RSHs to be defined have been designed for pseudo-Boolean optimization, we encode a solution to a PARTITION instance by the characteristic vector of I and arrive at the pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$, whose value $f(x)$ equals $m(I)$ if x encodes the set I. The following two simple RSHs are sometimes called hillclimbers. They store only one current search point and do not accept worsenings, which are, in this case, search points with some larger f-value. Both can be described by an initialization step and an infinite loop.

(1+1) EA

Initialization:	Choose $a \in \{0,1\}^n$ randomly.
Loop:	The loop consists of a mutation and a selection step.
Mutation:	For each position i, decide independently whether a_i should be flipped (replaced by $1 - a_i$), the flipping probability equals $1/n$.
Selection:	Replace a by a' iff $f(a') \leq f(a)$.

The (1+1) EA has a positive probability to create any search point from any search point and eventually optimizes each pseudo-Boolean function. This does not hold for Randomized Local Search (RLS), which flips only one bit per step.

RLS

This works like the (1+1) EA with a different mutation operator.
Mutation: Choose $i \in \{1, \ldots, n\}$ randomly and flip a_i.

We ignore the stopping criterion needed by practical implementations of these RSHs and are interested in the f-value of the current search point by some time t,

i. e., after t iterations of the infinite loop. Mainly, we try to estimate in how far this f-value approximates the optimum if t is bounded by some polynomial.

The paper is structured as follows. In Sect. 2, we provide some basic definitions and proof techniques needed to estimate the progress of the RSHs. In Sect. 3, we prove worst-case results on the approximation ratios obtainable by the RSHs within polynomial time. Moreover, we show that these results can be extended to parallel runs of the RSHs so as to design a randomized approximation scheme. In Sect. 3, we extend our techniques toward a probabilistic average-case analysis for two well-known input distributions. We finish with some conclusions.

2 Definitions and Proof Methods

Throughout the paper, we adopt the following conventions. Given an instance $w_1, \ldots, w_n$ for the optimization problem PARTITION, we assume w. l. o. g. that $w_1 \geq \cdots \geq w_n$. Moreover, we set $w := w_1 + \cdots + w_n$. We call the indices $1, \ldots, n$ *objects* and call w_i the *volume* of the i-th object. Sometimes, the objects themselves are also called $w_1, \ldots, w_n$. The optimization problem can be thought of as putting the objects in one of two bins, and a search point $x \in \{0,1\}^n$ is the characteristic vector of the set of objects put into the first bin. Then the goal function f corresponds to the total volume in the fuller bin w. r. t. x.

We will essentially exploit two proof methods in order to show bounds on the approximative quality of a solution output by the considered RSH. Both techniques only study progresses by so-called local steps, i. e., steps to search points at Hamming distance 1. Therefore, the analyses apply only until points of time where the heuristic RLS is able to get stuck in a local optimum. We are interested in sufficient conditions such that the considered RSH is able to improve the f-value by local steps. A first idea is to bound the volume of the largest object in the fuller bin from above. We do something similar but neglect the objects making the bin the fuller bin.

Definition 1 (Critical Volume). *Let $W = (w_1, \ldots, w_n)$ be an instance for the partition problem and let $\ell \geq w/2$ be a lower bound on the optimum f-value w. r. t. W. Moreover, let $x \in \{0,1\}^n$ be a characteristic vector s. t. $f(x) > \ell$. Let $w_{i_1} \geq w_{i_2} \geq \cdots \geq w_{i_k}$ be the objects in the fuller bin w. r. t. x, ranked in non-increasing order. Let $r := i_j$ for the smallest j such that $w_{i_1} + \cdots + w_{i_j} > \ell$. The volume w_r is called the* critical volume (w. r. t. W, ℓ and x).

The idea behind the critical volume is as follows. Suppose that x is the current search point of the RSH, leading to $f(x) > \ell$, and we know an upper bound v on the critical volume w. r. t. the instance, ℓ and x. Let r be the minimum i s. t. $w_i \leq v$. Due to $w_1 \geq \cdots \geq w_n$, we know that w_r is also an upper bound on the critical volume. By the definition of critical volume, there is some object $w_{r'}$, where $r' \geq r$, in the fuller bin. If we additionally know that $f(x) \geq \ell + v$ holds, together with $\ell \geq w/2$ this implies that $w_{r'}$ can be moved from the fuller bin into the emptier one, decreasing the f-value. Thus, a local step improves x.

The described sufficient condition for locally improvable search points can even be strengthened. Suppose that we have the same setting as before with the exception that now $\ell + v/2 < f(x) < \ell + v$ holds. If $w_{r'} \leq f(x) - w/2$, $w_{r'}$ can still be moved to the emptier bin. Otherwise, this step makes the fuller bin the emptier bin. Since $w_{r'} \leq w_r \leq v$, the total volume in this bin will be greater than $\ell + v/2 - w_r \geq w/2 - w_r/2$ and, therefore, the f-value less than $\ell + v/2$. Hence, the step is accepted by the RSH, too.

For RLS, a local step to a specific point at Hamming distance 1 has probability $1/n$, and for the (1+1) EA, the probability is at least $(1/n)(1-1/n)^{n-1} \geq 1/(en)$. If we know that the critical volume is always bounded by some small value, we can exploit this to show upper bounds on the f-values obtainable in expected polynomial time. The special case $w_1 \geq w/2$ can be solved exactly.

Lemma 1. *Let W and ℓ be as in Definition 1. Suppose that from some time t^* on, the critical volume w. r. t. W, ℓ and the current search point of the (1+1) EA or of RLS is at most v. Then the RSH reaches an f-value at most $\ell + v/2$ if $w_1 < w/2$ and at most w_1 otherwise in an expected number of another $O(n^2)$ steps.*

Proof. Let r be the smallest i where $w_i \leq v$. We consider the run of the RSH only from time t^* on. The proof uses a fitness-level argument [6]. Let $s := w_r + \cdots + w_n$, i. e., the sum of all volumes at most as large as w_r. Note that the conditions of the lemma and the definition of critical volume imply that $f(x) \leq \ell + s$ for all current search points x. According to $w_r, \ldots, w_n$, we partition the set of possible current search points by so-called fitness levels as follows. Let

$$L_i := \left\{ x \;\middle|\; \ell + s - \sum_{j=r}^{r+i-1} w_j \;\geq\; f(x) \;>\; \ell + s - \sum_{j=r}^{r+i} w_j \right\}$$

for $0 \leq i \leq n - r$ and $L_{n-r+1} := \{x \mid \ell = f(x)\}$. Now consider some x such that $f(x) > \ell + w_r/2$. By the definition of critical volume, there must be an object from $w_r, \ldots, w_n$ in the fuller bin whose move to the emptier bin decreases the f-value by its volume or leads to an f-value of at most $\ell + w_r/2 \leq \ell + v/2$. If $x \in L_i$, due to $w_r \geq \cdots \geq w_n$, there is even an object from $w_r, \ldots, w_{r+i}$ with this property. By the above considerations, moving this object to the emptier bin by a local step of the RSH has probability at least $1/(en)$ and, due to $w_r \geq \cdots \geq w_n$, leads to some $x' \in L_j$ such that $j > i$. The expected waiting time for such a step is at most en. After at most $n - r + 1$ sets have been left, the f-value has dropped to at most $\ell + w_r/2$. Hence, the total expected time after time t^* is $O(n^2)$.

If $w_1 \geq w/2$, we can apply the previous arguments with the special values $\ell := w_1$ and $r := 2$. The only difference is that in case that $f(x) > \ell$, there must be an object of volume at most $f(x) - \ell$ in the fuller bin. Hence, the RSH cannot be in a local optimum and is able to reach L_{n-r+1} by local steps. □

If we are satisfied with slightly larger f-values than guaranteed by Lemma 1, significantly smaller upper bounds on the expected time can be shown.

Lemma 2. *Let W and ℓ be as in Definition 1. Suppose that from some time t^* on, the critical volume w. r. t. W, ℓ and the current search point of the (1+1) EA or of RLS is at most v. Then for any $\gamma > 1$ and $0 < \delta < 1$, the (1+1) EA (RLS) reaches an f-value at most $\ell + v/2 + \delta w/2$ if $w_1 < w/2$ and at most $w_1 + \delta w/2$ otherwise in at most $\lceil en\ln(\gamma/\delta)\rceil$ ($\lceil n\ln(\gamma/\delta)\rceil$) another steps with probability at least $1 - \gamma^{-1}$. Moreover, the expected number of another steps is at most $2\lceil en\ln(2/\delta)\rceil$ ($2\lceil n\ln(2/\delta)\rceil$).*

Proof. Let r be the smallest i where $w_i \leq v$. First, we consider the run of the (1+1) EA from time t^* on. Let x be a current search point s. t. $f(x) > \ell + w_r/2$. We are interested in the contribution of the so-called small objects $w_r, \ldots, w_n$ to the f-value and want to estimate the average decrease of the f-value by a similar method as presented in [10]. Let $p(x) := \max\{f(x) - \ell - w_r/2, 0\}$ and note that due to the definition of critical volume and the conditions of the lemma, $p(x)$ is a lower bound on the contribution of small objects to $f(x)$. Moreover, as long as $p(x) > 0$, all steps moving only a small object to the emptier bin are accepted and decrease the p-value by its volume or lead to an f-value of at most $\ell + v/2$. Let p_0 be some current p-value. Since a local step of the (1+1) EA has probability at least $1/(en)$, the expected p-decrease is at least $p_0/(en)$ and the expected p-value after the step, therefore, at most $(1 - 1/(en))p_0$. Since the steps of the (1+1) EA are independent, this argumentation remains valid if p_0 is only an expected value and can be iterated until the p-value equals 0. Hence, the expected p-value p_t after t steps is at most $(1 - 1/(en))^t p_0$. For $t' := en\ln(\gamma/\delta)$, we have $p_{t'} \leq \delta p_0/\gamma \leq \delta w/(2\gamma)$. Since the p-value is non-negative, we can apply Markov's inequality, implying $p_{t'} \leq \delta w/2$ with probability at least $1 - 1/\gamma$. Since the previous arguments make no assumptions on p_0, we can repeat independent phases of length $\lceil en\ln(2/\delta)\rceil$. The expected number of phases until the p-value is at most $\delta w/2$ is at most 2, implying the lemma for the case $w_1 < w/2$.

If $w_1 \geq w/2$, we can apply the previous arguments with the special values $\ell := w_1$ and $r := 2$. The only difference is that in case that $f(x) > \ell$, there must be an object of volume at most $f(x) - \ell$ in the fuller bin. Hence, the (1+1) EA cannot be in a local optimum. Redefining $p(x) := f(x) - \ell$, the lemma follows for the (1+1) EA. The statements on RLS follow in the same way, taking into account that a local step has probability $1/n$. □

3 Worst-Case Analyses

In this section, we will study bounds on the approximation ratios obtainable by the RSHs within polynomial time regardless of the problem instance.

Theorem 1. *Let $\varepsilon > 0$ be a constant. On any instance for the partition problem, the (1+1) EA and RLS reach an f-value that is at least $(4/3+\varepsilon)$-approximate in an expected number of $O(n)$ steps and an f-value that is at least $4/3$-approximate in an expected number of $O(n^2)$ steps.*

Proof. We start by studying trivial instances with $w_1 \geq w/2$. Then even both statements follow for $\delta := 1/3$ by means of Lemma 2.

Now let $w_1 < w/2$ and $\ell := w/2$. We still have to distinguish two cases. The first case holds if $w_1+w_2 > 2w/3$. This implies $w_1 > w/3$ and, therefore, $w-w_1 < 2w/3$. Hence, if we start with w_1 and w_2 in the same bin, a step separating w_1 and w_2 by putting w_2 into the emptier bin is accepted, and these objects will remain separated afterwards. The expected time until such a separating step occurs is $O(n)$. Afterwards, the critical volume according to Definition 1 is always bounded above by w_3. Since $w_3 + \cdots + w_n < w/3$, we know that $w_i < w/3$ for $i \geq 3$. Hence, the first statement of the theorem follows for $\delta := \varepsilon$ by Lemma 2 and the second one by Lemma 1. If $w_1 + w_2 \leq 2w/3$, we have $w_i \leq w/3$ for $i \geq 2$. Since $w_1 < w/2$, this implies that the critical volume is always at most $w_2 \leq w/3$. Therefore, the theorem holds also in this case. □

The approximation ratio 4/3 that the RSHs are able to obtain within expected polynomial time is at least almost tight. Let n be even and $\varepsilon > 0$ be some arbitrarily small constant. Then the instance W^*_ε, an almost worst-case instance, contains two objects w_1 and w_2 of volume $1/3-\varepsilon/4$ each and $n-2$ objects of volume $(1/3+\varepsilon/2)/(n-2)$. Note that the total volume has been normalized to 1 and that the instance has an exponential number of perfect partitions.

Theorem 2. *Let ε be any constant s. t. $0 < \varepsilon < 1/3$. With probability $\Omega(1)$, both the (1+1) EA and RLS take $n^{\Omega(n)}$ steps to create a solution better than $(4/3-\varepsilon)$-approximate for the instance W^*_ε.*

Proof. The proof idea is to show that the RSH reaches a situation where w_1 and w_2 are in one bin and at least $k := n-2-(n-2)\varepsilon/2$ of the remaining so-called small objects are in the other one. Since $\varepsilon < 1/3$, at least k objects yield a total volume of more than $1/3+\varepsilon/4$. To leave the situation by separating w_1 and w_2, the RSH has to transfer small objects of a total volume of at least $\varepsilon/4$ from one bin to the other one in a single step. For this, $(n-2)\varepsilon/2$ small objects are not enough. Flipping $\Omega(n)$ bits in one step of the (1+1) EA has probability $n^{-\Omega(n)}$, and flipping $\Omega(n)$ bits at least once within n^{cn} steps is, therefore, still exponentially unlikely if the constant c is small enough. For RLS, the probability is even 0. Since the total volume in the fuller bin is at least $2/3-\varepsilon/2$ unless w_1 and w_2 are separated, this will imply the theorem.

To show the claim that the described situation is reached with probability $\Omega(1)$, we consider the initial search point of the RSH. With probability 1/2, it puts w_1 and w_2 into the same bin. Therefore, we estimate the probability that enough small objects are transferred from this bin to the other one in order to reach the situation, before a bit at the first two positions (denoting the large objects) flips. In a phase of length cn for any constant c, with probability $(1-2/n)^{cn} = \Omega(1)$, the latter never happens. Under this assumption, each step moving a small object into the emptier bin is accepted. By the same idea as in the proof of Lemma 2, we estimate the expected decrease of the contribution of small objects to the f-value. Reducing it to at most an $\varepsilon/2$-fraction of its initial contribution suffices to obtain at least k objects in the emptier bin. Each step leads to an expected decrease by at least a $1/(en)$-fraction. Since ε is a positive constant, $O(n)$ steps are sufficient to decrease the contribution to at

most an expected $\varepsilon/4$-fraction. By Markov's inequality, we obtain the desired fraction within $O(n)$ steps with probability at least $1/2$. Since c may be chosen appropriately, this proves the theorem. □

The worst-case example studied in Theorem 2 suggests that the RSH is likely to arrive at a bad approximation if it misplaces objects of high volume. On the other hand, it can easily be shown for the example that the RSH is able to find an optimal solution with probability $\Omega(1)$ in polynomial time if it separates the two largest objects in the beginning. We try to generalize this to arbitrary instances. In order to obtain a $(1+\varepsilon)$-approximation in polynomial time according to Lemma 1, the critical volume should be bounded above by εw. Due to the ordering $w_1 \geq \cdots \geq w_n$, all objects of index at least $s := \lceil 1/\varepsilon \rceil$ are bounded by this volume. Therefore, the crucial idea is to bound the probability that the RSH distributes the first $s-1$ objects in such a nice way that the critical volume is at most w_s. Interestingly, this is essentially the same idea as for the classical PTAS for the partition problem presented by Graham [13]. Even if the RSH does not know of this algorithmic idea, it is able to behave accordingly by chance.

Theorem 3. *Let $\varepsilon \geq 4/n$. With probability at least $2^{-(\mathrm{e}\log \mathrm{e}+\mathrm{e})\lceil 2/\varepsilon\rceil \ln(2/\varepsilon)-\lceil 2/\varepsilon\rceil}$, the (1+1) EA on any instance for the partition problem creates a $(1+\varepsilon)$-approximate solution in $\lceil \mathrm{e}n\ln(2/\varepsilon))\rceil$ steps. The same holds for RLS with $\lceil n\ln(2/\varepsilon))\rceil$ steps and a probability of even at least $2^{-(\log \mathrm{e}+1)\lceil 2/\varepsilon\rceil \ln(2/\varepsilon)-\lceil 2/\varepsilon\rceil}$.*

Proof. Let $s := \lceil 2/\varepsilon \rceil \leq n/2+1$. Since $w_1 \geq \cdots \geq w_n$, it holds that $w_i \leq \varepsilon w/2$ for $i \geq s$. If $w_1 + \cdots + w_{s-1} \leq w/2$, the critical volume w. r. t. $\ell := w/2$ is always bounded above by w_s and, therefore, by $\varepsilon w/2$. Therefore, in this case, the theorem follows for $\delta := \varepsilon$ and $\gamma := 2$ by Lemma 2.

In the following, we assume $w_1 + \cdots + w_{s-1} > w/2$. Consider all partitions of only the first $s-1$ objects. Let ℓ^* be the minimum volume of the fuller bin over all these partitions and $\ell := \max\{w/2, \ell^*\}$. Then with a probability at least 2^{-s+2}, in the beginning, neither bin receives a contribution of more than ℓ by these objects. As long as the property remains valid, we can be sure that the critical volume w. r. t. ℓ is at most $w_s \leq \varepsilon w/2$, and we can apply the arguments from the first paragraph. The probability that in a phase of $t := \lceil \mathrm{e}n\ln(2/\varepsilon)\rceil$ steps, it never happens that at least one of the first $s-1$ bits flips is bounded below by

$$\left(1-\frac{s-1}{n}\right)^{\mathrm{e}n(\ln(2/\varepsilon))+1} \geq \mathrm{e}^{-\mathrm{e}(\ln(2/\varepsilon))(s-1)}\left(1-\frac{s-1}{n}\right)^{s\mathrm{e}\ln(2/\varepsilon)},$$

which is at least $2^{-(\mathrm{e}\log \mathrm{e}+\mathrm{e})\lceil 2/\varepsilon\rceil \ln(2/\varepsilon)}$ since $s-1 \leq n/2$. Under the mentioned conditions, by Lemma 2 for $\delta := \varepsilon$ and $\gamma := 2$, the (1+1) EA reaches a $(1+\varepsilon)$-approximation within t steps with probability at least $1/2$. Altogether, the desired approximation is reached within t steps with probability at least

$$\frac{1}{2}\cdot 2^{-\lceil 2/\varepsilon\rceil+2}\cdot 2^{-(\mathrm{e}\log \mathrm{e}+\mathrm{e})\lceil 2/\varepsilon\rceil \ln(2/\varepsilon)} \geq 2^{-(\mathrm{e}\log \mathrm{e}+\mathrm{e})\lceil 2/\varepsilon\rceil \ln(2/\varepsilon)-\lceil 2/\varepsilon\rceil}.$$

The statement for RLS follows by redefining $t := \lceil n\ln(2/\varepsilon)\rceil$. □

Theorem 3 allows us to design a PRAS (polynomial-time randomized approximation scheme, see [14]) for the partition problem using multistart variants of the considered RSH. If $\ell(n)$ is a lower bound on the probability that a single run of the RSH achieves the desired approximation in $O(n \ln(1/\varepsilon))$ steps then this holds for at least one out of $\lceil 2/\ell(n) \rceil$ parallel runs with a probability of at least $1 - \mathrm{e}^{-2} > 3/4$. According to the lower bounds $\ell(n)$ given in Theorem 3, the computational effort $c(n)$ incurred by the parallel runs is bounded above by $O(n \ln(1/\varepsilon)) \cdot 2^{(\mathrm{e} \log \mathrm{e} + \mathrm{e}) \lceil 2/\varepsilon \rceil \ln(2/\varepsilon) + O(1/\varepsilon)}$. For $\varepsilon > 0$ a constant, $c(n) = O(n)$ holds, and $c(n)$ is still a polynomial for any $\varepsilon = \Omega(\log \log n / \log n)$. This is the first example where it could be shown that an RSH serves as a PRAS for an NP-hard optimization problem. Before, a characterization of an EA as a PRAS was only known for the maximum matching problem [9].

4 Average-Case Analyses

A probabilistic analysis of RSHs on random inputs must take into account two sources of randomness. Since this constitutes one of the first attempts in this respect, we concentrate on two fairly simple and well-known distributions. First, we assume the volumes w_i to be independent random variables drawn uniformly from the interval $[0, 1]$. This is called the *uniform-distribution model*. Second, we rather consider exponentially distributed random variables with parameter 1, which is called the *exponential-distribution model*.

In the last two decades, some average-case analyses of deterministic heuristics for the partition problem have been performed. The first such analyses studied the LPT rule, a greedy algorithm sorting the volumes decreasingly and putting each object from the resulting sequence into the currently emptier bin. Extending a result that stated convergence in expectation, Frenk and Rinnooy Kan [11] were able to prove that the LPT rule converges to optimality at a speed of $O(\log n/n)$ almost surely in several input models, including the uniform-distribution and exponential-distribution model. Further results on average-case analyses of more elaborate deterministic heuristics are contained in [12].

In our models, the optimum f-value is random. Therefore, for a current search point, we now consider the so-called discrepancy measure rather than an approximation ratio. The discrepancy denotes the absolute difference of the total volumes in the bins. It is easy to see that the initial discrepancy in both models is $\Omega(\sqrt{n})$ with constant probability. We start with a simple upper bound on the discrepancy after polynomially many steps in the uniform-distribution model.

Lemma 3. *The discrepancy of the (1+1) EA (RLS) in the uniform-distribution model is bounded above by* 1 *after an expected number of* $O(n^2)$ *(*$O(n \log n)$*) steps. Moreover, for any constant* $c \geq 1$*, it is bounded above by* 1 *with probability at least* $1 - O(1/n^c)$ *after* $O(n^2 \log n)$ *(*$O(n \log n)$*) steps.*

Proof. Recall the argumentation given after Definition 1. Hence, if the discrepancy is greater than 1, local steps can improve the f-value by the volume moved or lead to a discrepancy of less than 1. By a fitness-level argument like in the

proof of Lemma 1, we obtain the $O(n^2)$ bound for the (1+1) EA. This holds for any random instance. Hence, by Markov's inequality and repeating phases, the discrepancy is at most 1 with probability $1-O(1/n^c)$ after $O(n^2 \log n)$ steps. The statements for RLS follow immediately by the Coupon Collector's Theorem. □

The foregoing upper bound on the discrepancy was easy to obtain; however, for the (1+1) EA, we can show that with a high probability, the discrepancy provably becomes much lower than 1 in a polynomial number of steps. The reason is as follows. All preceding proofs considered only local steps; however, the (1+1) EA is able to leave local optima by flipping several bits in a step.

The following two theorems will use the following simple properties of order statistics (e. g., [15]). Let $X_{(1)} \geq \cdots \geq X_{(n)}$ be the order statistics of the volumes in the uniform-distribution model. Then for $1 \leq i \leq n-1$ and $0 < t < 1$, $\text{Prob}(X_{(i)} - X_{(i+1)} \geq t) = \text{Prob}(X_{(n)} \geq t) = (1-t)^n$. In the exponential-distribution model, there is a sequence of independent, parameter-1 exponentially distributed random variables $Y_1, \ldots, Y_n$ s. t. $X_{(i)} = \sum_{j=i}^{n} \frac{Y_j}{j}$ for $1 \leq i \leq n$.

Theorem 4. *Let $c \geq 1$ be an arbitrary constant. After $O(n^{c+4} \log n)$ steps, the discrepancy of the (1+1) EA in the uniform-distribution model is bounded above by $O(\log n/n)$ with probability at least $1-O(1/n^c)$. Moreover, the expected discrepancy after $O(n^5 \log n)$ steps is also bounded by $O(\log n/n)$.*

Proof. By Lemma 3, the discrepancy is at most 1 after $O(n^2 \log n)$ steps with probability at least $1-O(1/n^2)$. Since the discrepancy is always bounded by n, the failure probability contributes only an $O(1/n)$-term to the expected discrepancy after $O(n^5 \log n)$ steps. From now on, we consider the time after the first step where the discrepancy is at most 1 and concentrate on steps flipping two bits. If an accepted step moves an object of volume w' from the fuller to the emptier bin and one of volume $w'' < w'$ the other way round, the discrepancy may be decreased by $2(w'-w'')$. We look for combinations s. t. $w'-w''$ is small.

Let $X_{(1)} \geq \cdots \geq X_{(n)}$ be the order statistics of the random volumes. If for the current search point, there is some i s. t. $X_{(i)}$ is the order statistic of an object in the fuller and $X_{(i+1)}$ is in the emptier bin then a step exchanging $X_{(i)}$ and $X_{(i+1)}$ may decrease the discrepancy by $2(X_{(i)} - X_{(i+1)})$. If no such i exists, all objects in the emptier bin are larger than any object in the fuller bin. In this case, $X_{(n)}$ can be moved into the emptier bin, possibly decreasing the discrepancy by $2X_{(n)}$. Hence, we need upper bounds on $X_{(i)} - X_{(i+1)}$ and $X_{(n)}$.

Let $t^* := (c+1)(\ln n)/n$, i. e., $t^* = O(\log n/n)$ since c is a constant. We obtain $(1-t^*)^n \leq n^{-c-1}$. By the above-mentioned statement, this implies that with probability $1-O(1/n^c)$, $X_{(i)} - X_{(i+1)} \leq t^*$ holds for all i and $\text{Prob}(X_{(n)} \geq t^*) = O(1/n^{c+1})$. Now assume $X_{(i)} - X_{(i+1)} \leq t^*$ for all i and $X_{(n)} \leq t^*$. If this does not hold, we bound the expected discrepancy after $O(n^{c+4} \log n)$ steps by 1, yielding a term of $O(1/n^c) = O(1/n)$ in the total expected discrepancy. By the argumentation given after Definition 1, there is always a step flipping at most 2 bits that decreases the discrepancy as long as the discrepancy is greater than t^*.

It remains to estimate the time to decrease the discrepancy. Therefore, we need lower bounds on $X_{(i)} - X_{(i+1)}$ and X_n. Let $\ell^* := 1/n^{c+2}$. We obtain $\text{Prob}(X_{(i)} - X_{(i+1)} \geq \ell^*) \geq \mathrm{e}^{-2/n^{c+1}} \geq 1 - 2/n^{c+1}$. Hence, with probability $1 - O(1/n^c)$, $X_{(i)} - X_{(i+1)} \geq \ell^*$ for all i. Moreover, $X_{(n)} \geq \ell^*$ with probability $1 - O(1/n^{c+1})$. We assume these lower bounds to hold, introducing a failure probability of only $O(1/n^c)$, whose contribution to the expected discrepancy is negligible as above. A step flipping 1 resp. 2 specific bits has probability at least $n^{-2}(1-1/n)^{n-2} \geq 1/(\mathrm{e}n^2)$. Hence, the discrepancy is decreased by at least ℓ^* or drops below t^* with probability $\Omega(1/n^2)$ in each step. The expected time until the discrepancy becomes at most t^* is, therefore, bounded above by $O(\ell^* n^2) = O(n^{c+4})$, and, by repeating phases, the time is at most $O(n^{c+4} \log n)$ with probability $1 - O(1/n^c)$. The sum of all failure probabilities is $O(1/n^c)$. □

Theorem 5. *Let $c \geq 1$ be an arbitrary constant. With probability $1 - O(1/n^c)$, the discrepancy of the (1+1) EA in the exponential-distribution model is bounded above by $O(\log n)$ after $O(n^2 \log n)$ steps and by $O(\log n/n)$ after $O(n^{c+4} \log^2 n)$ steps. Moreover, the expected discrepancy is $O(\log n)$ after $O(n^2 \log n)$ steps and it is $O(\log n/n)$ after $O(n^6 \log^2 n)$ steps.*

Proof. The expected value of the initial discrepancy is bounded above by n since each object has an expected volume of 1. In the following, all failure probabilities can be bounded by $O(1/n^2)$. In case of a failure, we will tacitly bound the failure's contribution to the expected discrepancy after $O(n^2 \log n)$ resp. $O(n^6 \log^2 n)$ steps by $O(1/n)$. Next, we will show that with probability $1 - O(1/n^c)$, the critical volume w. r. t. $\ell := w/2$ is always $O(\log n)$. Together with Lemma 1, this claim implies the theorem for the situation after $O(n^2 \log n)$ steps.

To show the claim, like in the proof of Theorem 4, we consider the order statistics $X_{(1)} \geq \cdots \geq X_{(n)}$ of the random volumes. Our goal is to show that with high probability, $X_{(1)} + \cdots + X_{(k)} \leq w/2$ holds for $k := \lceil \delta n \rceil$ and some constant $\delta > 0$. Afterwards, we will prove that $X_{(k)} = O(\log n)$ with high probability.

Each object has a volume of at least 1 with probability $\mathrm{e}^{-1} > 1/3$. By Chernoff bounds, $w \geq n/3$ with probability $1 - 2^{-\Omega(n)}$. To bound $X_{(1)} + \cdots + X_{(k)}$, we use the above-mentioned identity $X_{(i)} = \sum_{j=i}^{n} Y_j/j$. Hence,

$$\begin{aligned} X_{(1)} + \cdots + X_{(k)} &= Y_1 + 2 \cdot \frac{Y_2}{2} + \cdots + k \cdot \frac{Y_k}{k} + k \sum_{i=k+1}^{n} \frac{Y_i}{i} \\ &\leq \sum_{j=1}^{k} Y_j + \sum_{i=1}^{\lceil n/k \rceil} \frac{1}{i} \sum_{j=ik+1}^{(i+1)k} Y_j, \end{aligned}$$

where $Y_j := 0$ for $j > n$. Essentially, we are confronted with $\lceil n/k \rceil$ sums of k exponentially distributed random variables each. A simple calculation (deferred to the last paragraph of this proof) yields that a single sum is bounded above by $2k$ with probability $1 - 2^{-\Omega(k)}$, which is at least $1 - 2^{-\Omega(n)}$ for the values of k considered. Since we consider at most n sums, this statement also holds for all

sums together. Hence, with probability $1 - 2^{-\Omega(n)}$, the considered expression is bounded above by

$$2\lceil \delta n \rceil + \sum_{i=1}^{1/\delta} \frac{2\lceil \delta n \rceil}{i} \leq 2(\delta n + 1) \ln(1/\delta + 2),$$

which is strictly less than $n/6$ for $\delta \leq 1/50$ and n large enough. Together with the above lower bound on w, this implies that with probability $1 - 2^{-\Omega(n)}$, the critical volume is always bounded above by the $\lceil n/50 \rceil$-th largest volume.

How large is $X_{(\lceil n/50 \rceil)}$? Since with probability at least $1 - n\mathrm{e}^{-(c+1)\ln n} \geq 1 - n^{-c}$, all random variables Y_j are bounded above by $(c+1)\ln n$, it follows that with at least the same probability, we have

$$X_{\lceil n/50 \rceil} = \sum_{j=\lceil n/50 \rceil}^{n} \frac{Y_j}{j} \leq (c+1)(\ln n)((\ln n) + 1 - \ln(n/49))$$

(for n large enough), which equals $(c+1)(\ln(49)+1)(\ln n) = O(\log n)$. The sum of all failure probabilities is $O(1/n^c)$, bounding the critical volume as desired.

We still have to show the theorem for the case of $O(n^{c+4}\log^2 n)$ steps. Now we assume that the discrepancy has been decreased to $O(\log n)$ and use the same idea as in the proof of Theorem 4 by investigating steps swapping $X_{(i)}$ and $X_{(i+1)}$ or moving $X_{(n)}$. Above, we have shown that with probability $1 - O(1/n^c)$, the smallest object in the fuller bin is always at most $X_{(k)}$ for some $k \geq n/50$. Since $X_{(k)} - X_{(k+1)} = Y_k/k$, we obtain $X_{(k)} - X_{(k+1)} \leq 50Y_k/n$ with the mentioned probability. Moreover, it was shown that $Y_j \leq (c+1)\ln n$ for all j with at least the same probability. Altogether, $X_{(k)} - X_{(k+1)} \leq 50(c+1)(\ln n/n) =: t^*$ with probability $1 - O(1/n^c)$. Since $X_{(n)} = Y_n/n$, $\mathrm{Prob}(X_{(n)} \leq t^*)$ with probability $1 - O(1/n^c)$, too. In the following, we assume these upper bounds to hold. This implies that as long as the discrepancy is greater than t^*, there is a step flipping at most 2 bits and decreasing the discrepancy.

It remains to establish lower bounds on $X_{(k)} - X_{(k+1)}$ and $X_{(n)}$. We know that $X_{(k)} - X_{(k+1)} \geq Y_k/n$ and obtain $\mathrm{Prob}(X_{(k)} - X_{(k+1)} \geq 1/n^{c+2}) \geq \mathrm{e}^{-1/n^{c+1}} \geq 1 - 1/n^{c+1}$ for any fixed k and $\mathrm{Prob}(X_{(n)} \geq 1/n^{c+2}) \geq 1 - 1/n^{c+1}$. All events together occur with probability $1 - O(1/n^c)$. By the same arguments as in the proof of Theorem 4, the expected time until the discrepancy becomes at most t^* is $O(n^{c+4}\log n)$, and the time is bounded by $O(n^{c+4}\log^2 n)$ with probability $1 - O(1/n^c)$. The sum of all failure probabilities is $O(1/n^c)$. This will imply the theorem.

We still have to show the following claim. The sum S_k of k exponentially distributed random variables with parameter 1 is at most $2k$ with probability $1 - 2^{-\Omega(k)}$. Observe that S_k follows a gamma distribution, i. e.,

$$\mathrm{Prob}(S_k \geq 2k) = \mathrm{e}^{-2k}\left(1 + \frac{2k}{1!} + \cdots + \frac{(2k)^{k-1}}{(k-1)!}\right) \leq \frac{k\mathrm{e}^{-2k}(2k)^{k-1}}{(k-1)!}.$$

By Stirling's formula, the last expression is bounded above by

$$\frac{\mathrm{e}^{-2k+(k-1)} \cdot 2^{k-1} \cdot k \cdot k^{k-1}}{(k-1)^{k-1}} = \mathrm{e}^{-2k+(k-1)} \cdot 2^{k-1} \cdot k \cdot \left(1 - \frac{1}{k}\right)^{-(k-1)} = 2^{-\Omega(k)}.$$

This proves the claim and, therefore, the theorem. □

Theorem 4 and Theorem 5 imply that in both models, the solution of the (1+1) EA after a polynomial number of steps converges to optimality *in expectation.* Moreover, the asymptotic discrepancy after a polynomial number of steps is at most $O(\log n/n)$, i. e., convergent to 0, with probability $1-O(1/n^c)$, i. e., convergent to 1 polynomially fast. This is almost as strong as the above-mentioned result for the LPT rule.

5 Conclusions

In this paper, we have presented a probabilistic analysis for randomized search heuristics on the optimization variant of the PARTITION problem. In the worst case, both the (1+1) EA and RLS with constant probability need exponential time to create solutions being better than $(4/3-\varepsilon)$-approximate; however, parallel runs of the heuristics lead to a PRAS. An average-case analysis with respect to two input distributions shows that the (1+1) EA, inspected after a polynomial number of steps, creates solutions that are in some sense convergent to optimality. By this average-case analysis, we have made a step towards a theoretical justification of the efficiency of randomized search heuristics for practically relevant problem instances.

Acknowledgements. The author thanks Ingo Wegener for helpful discussions.

References

1. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. In: Proc. of 35th STOC. (2003) 232–241
2. Spielman, D.A., Teng, S.H.: Smoothed analysis: Why the simplex algorithm usually takes polynomial time. In: Proc. of 33rd STOC. (2001) 296–305
3. Hochbaum, D.S., ed.: Approximation Algorithms for NP-Hard Problems. Wadsworth Publishing Company (1997)
4. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220** (1983) 1087–1092
5. Baeck, T., Fogel, D.B., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. Institute of Physics Publishing (1997)
6. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276** (2002) 51–81
7. Wegener, I., Witt, C.: On the optimization of monotone polynomials by simple randomized search heuristics. Combinatorics, Probability and Computing (2005) in press.

8. Witt, C.: An analysis of the (μ+1) EA on simple pseudo-boolean functions. In: Proc. of GECCO 2004, LNCS 3102 (2004) 761–773
9. Giel, O., Wegener, I.: Evolutionary algorithms and the maximum matching problem. In: Proc. of 20th STACS, LNCS 2607 (2003) 415–426
10. Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. In: Proc. of GECCO 2004, LNCS 3102 (2004) 713–724
11. Frenk, J.B.G., Rinnooy Kan, A.H.G.: The rate of convergence to optimality of the LPT rule. Discrete Applied Mathematics **14** (1986) 187–197
12. Coffman, Jr., E.G., Whitt, W.: Recent asymptotic results in the probabilistic analysis of schedule makespans. In Chrètienne, P., Coffman, Jr., E.G., Lenstra, J.K., Liu, Z., eds.: Scheduling Theory and its Applications. Wiley (1995) 15–31
13. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics **17** (1969) 263–269
14. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
15. David, H.A., Nagaraja, H.N.: Order Statistics. 3rd edn. Wiley (2003)

Sampling Sub-problems of Heterogeneous Max-cut Problems and Approximation Algorithms

Petros Drineas[1], Ravi Kannan[2], and Michael W. Mahoney[3]

[1] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York 12180
drinep@cs.rpi.edu
[2] Department of Computer Science, Yale University, New Haven, CT 06520
kannan@cs.yale.edu
[3] Department of Mathematics, Yale University, New Haven, CT 06520
mahoney@cs.yale.edu

Extended Abstract

Abstract. Recent work in the analysis of randomized approximation algorithms for NP-hard optimization problems has involved approximating the solution to a problem by the solution of a related sub-problem of constant size, where the sub-problem is constructed by sampling elements of the original problem uniformly at random. In light of interest in problems with a heterogeneous structure, for which uniform sampling might be expected to yield sub-optimal results, we investigate the use of nonuniform sampling probabilities. We develop and analyze an algorithm which uses a novel sampling method to obtain improved bounds for approximating the Max-Cut of a graph. In particular, we show that by judicious choice of sampling probabilities one can obtain error bounds that are superior to the ones obtained by uniform sampling, both for weighted and unweighted versions of Max-Cut. Of at least as much interest as the results we derive are the techniques we use. The first technique is a method to compute a compressed approximate decomposition of a matrix as the product of three smaller matrices, each of which has several appealing properties. The second technique is a method to approximate the feasibility or infeasibility of a large linear program by checking the feasibility or infeasibility of a nonuniformly randomly chosen sub-program of the original linear program. We expect that these and related techniques will prove fruitful for the future development of randomized approximation algorithms for problems whose input instances contain heterogeneities.

1 Introduction

1.1 Background

We are interested in developing improved methods to compute approximate solutions to certain NP-hard optimization problems that arise in applications of

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 57–68, 2005.

graph theory and that have significant heterogeneities and/or nonuniformities. The methods we present here are a first step in that direction; they make use of random sampling according to certain judiciously chosen nonuniform probability distributions and they depend heavily on our recent work on designing and analyzing fast Monte Carlo algorithms for performing useful computations on large matrices [12, 13, 14].

As an application of these methods we design and analyze an algorithm to compute an approximation for the Max-Cut problem. In a Max-Cut problem, also known as a maximum weight cut problem, the input consists of the $n \times n$ adjacency matrix A of an undirected graph $G = (V, E)$ with n vertices, and the objective of the problem is to find a cut, i.e., a partition of the vertices into two subsets V_1 and V_2, such that the number of edges of E that have one endpoint in V_1 and one endpoint in V_2 is maximized. In its weighted version, the input consists of an $n \times n$ weighted adjacency matrix A, and the objective is to find a cut such that the sum of the weights of the edges of E that have one endpoint in V_1 and one endpoint in V_2 is maximized.

Work originating with [3] has focused on designing PTASs for a large class of NP-hard optimization problems, such as the Max-Cut problem, when the problem instances are dense [3, 6, 16, 18, 17, 1, 2]. [3] and [6], using quite different methods, designed approximation algorithms for Max-Cut (and other problems) that achieve an additive error of ϵn^2 (where $\epsilon > 0$, $\epsilon \in \Omega(1)$ is an error parameter) in a time poly(n) (and exponential in $1/\epsilon$); this implies relative error for dense instances of these problems. In [18] it was shown that a constant-sized (with respect to n) sample of a graph is sufficient to determine whether a graph has a cut close to a certain value. This work investigated dense instances of NP-hard problems from the viewpoint of query complexity and property testing and yielded an $O(1/\epsilon^5)$ time algorithm to approximate, among other problems, dense instances of Max-Cut. [16] and [17] examined the regularity properties of dense graphs and developed a new method to approximate matrices; this led to a PTAS for dense instances of all Max-2-CSP, and more generally for dense instances of all Max-CSP, problems. [1, 2] extended this and developed a PTAS for dense instances of all Max-CSP problems in which the sample complexity was poly$(1/\epsilon)$ and independent of n; when applied to the Max-Cut problem this led to an $O\left(\frac{\log 1/\epsilon}{\epsilon^4}\right)$ time approximation algorithm.

In all these cases, these approximation algorithms involve sampling elements of the input uniformly at random in order to construct a sub-problem which is then used to compute an approximation to the original problem with additive error at most ϵn^2 [3, 6, 16, 18, 17, 1, 2]; this then translates into a relative error bound for dense graphs. These methods are not useful for nondense graphs since with such an error bound a trivial approximate solution would always suffice. This uniform sampling does have the advantage that it can be carried out "blindly" since the "coins" can be tossed before seeing the data; then, given either random access or one pass, i.e., one sequential read, through the data, samples from the data may be drawn and then used to compute. Such uniform

sampling is appropriate for problems that have nice uniformity or regularity properties [16].

In many applications of graph theory problems, however, significant heterogeneities are present [22]. For instance, the graph may have a power-law structure, or a large part of the graph may be very sparse and a small subset of vertices (sometimes, but not always a $o(n)$-sized subset) may have most of the edges ($1-o(1)$ of the edges) incident to them. Similarly, in a weighted graph, the total weight of edges incident to most vertices may be small, while among the remaining vertices the total weight of incident edges may be quite large. Neither the adjacency matrix nor the adjacency list representation of a graph used in property testing captures well this phenomenon [18].

With the additional flexibility of several passes over the data, we may use one pass to assess the "importance" of a piece (or set of pieces) of data and determine the probability with which it (or they) should be sampled, and a second pass to actually draw the sample. Such importance sampling has a long history [21]. In recent work, we have shown that by sampling columns and/or rows of a matrix according to a judiciously-chosen and data-dependent nonuniform probability distribution, we may obtain better (relative to uniform sampling) bounds for approximation algorithms for a variety of common matrix operations [12, 13, 14]; see also [8, 9, 10]. The power of using information to construct nonuniform sampling probabilities has also been demonstrated in recent work examining so-called oblivious versus so-called adaptive sampling [4, 5]. For instance, it was demonstrated that in certain cases approximation algorithms (for matrix problems such as those discussed in [12, 13, 14]) which use oblivious uniform sampling cannot achieve the error bounds that are achieved by adaptively constructing nonuniform sampling probabilities [4, 5].

1.2 Our Main Result

In this paper we develop an approximation algorithm for both weighted and unweighted versions of the Max-Cut problem. We do so by using nonuniform sampling probabilities in the construction of the sub-problem to be solved. For weighted graphs, these methods lead to substantial improvements when the average edge weight is much less than the maximum edge weight; for unweighted graphs, we show that at the cost of substantial additional sampling, these methods lead to an additive error improvement over previous results [18, 2].

Let A be the $n \times n$ weighted adjacency matrix of a graph $G = (V, E)$, let ϵ be a constant independent of n, and recall that $\|A\|_F^2 = \sum_{ij} A_{ij}^2$. A main result of this paper, which is presented in a more precise form in Theorem 3, is that there exists an algorithm that, upon being input A, returns an approximation Z to the Max-Cut of A such that with high probability

$$|Z - \textbf{MAX-CUT}\,[A]| \leq \epsilon n \|A\|_F . \tag{1}$$

The algorithm makes three passes, i.e., three sequential reads, through the matrix A and then needs constant additional space and constant additional time (constant, that is, with respect to n) in order to compute the approximation.

The algorithm uses a judiciously-chosen and data-dependent nonuniform probability distribution in order to obtain bounds of the form (1); these probabilities are computed in the first two passes through the matrix.

Our results are of particular interest for weighted graphs. Note that for weighted problems, the ϵn^2 error bound of previous work for unweighted graphs extends easily to $\epsilon n^2 W_{max}$, where W_{max} is the maximum edge weight. For these problems, $\|A\|_F / n$ may be thought of as the average weight over all the edges; one may then view our error bounds as replacing W_{max} in the $\epsilon n^2 W_{max}$ error bound by $\|A\|_F / n$. If only a few of the weights are much higher than this average value, the bound of $\epsilon n \|A\|_F$ given in (1) is much better than the bound of $\epsilon n^2 W_{max}$.

For a complete graph $\|A\|_F^2 = n(n-1)$ since $A_{ij} = 1$ for every $i \neq j$. For general unweighted graphs $\sqrt{2|E|} = \|A\|_F < n$, where $|E|$ the cardinality of the edge set. Thus, in general, the additive error bound (1) becomes $\epsilon n \sqrt{2|E|}$, which is an improvement over the previous results of ϵn^2 [18, 2]. In addition, from this bound we obtain a PTAS for graphs with $|E| = \Omega(n^2)$. Unfortunately, this does not translate into a PTAS for any class of sub-dense graphs. Demonstrating that such a PTAS exists would be significant application of our methodology and is the object of current work; it has been shown recently by other methods that there does exist a PTAS for Max-Cut and other Max-2-CSP problems restricted to slightly subdense, i.e., $\Omega(n^2/\log n)$ edges, graphs [7]. Since we are primarily interested in presenting a methodology to deal with heterogeneities and nonuniformities that arise in applications of graph theory problems, we make no effort to optimize constants or polynomial factors. In particular, although we have a PTAS, both the sampling complexity and the running time of the algorithm are exponential in $1/\epsilon$, which is substantially larger than previous results [18, 2]; we expect that this may be substantially improved.

1.3 Outline of the Paper

In Section 2 we provide a review of relevant linear algebra and of our first intermediate result which is the approximate $C\tilde{U}R$ decomposition results from [14] that will be needed for the proofs of our results. In Section 3 we then present our second intermediate result that deals with approximating the feasibility of a LP by considering random sub-programs of the LP. Then, in Section 4 we present and analyze an algorithm to approximate the Max-Cut of a matrix; in particular, we summarize a proof of Theorem 3 which establishes (1).

1.4 Technical Report and Journal Paper

For the proofs of the results presented here, as well as for more details and discussion related to these results, see the associated technical report [15]. Also, see the associated journal paper [11].

2 Review of Relevant Background

This section contains a review of linear algebra that will be useful throughout the paper; for more detail, see [19, 20, 23] and references therein. This section also contains a review of the compressed approximate $C\tilde{U}R$ decomposition of a matrix. The $C\tilde{U}R$ result is presented in much more generality in [14] and depends critically on related work on computing an approximation to the Singular Value Decomposition (SVD) of a matrix and on computing an approximation to the product of two matrices; see [12, 13, 14] for more details.

2.1 Review of Linear Algebra

For a vector $x \in \mathbb{R}^n$ we let x_i, $i = 1, \ldots, n$, denote the i-th element of x and we let $|x| = \left(\sum_{i=1}^n |x_i|^2\right)^{1/2}$. For a matrix $A \in \mathbb{R}^{m \times n}$ we let $A^{(j)}$, $j = 1, \ldots, n$, denote the j-th column of A as a column vector and $A_{(i)}$, $i = 1, \ldots, m$, denote the i-th row of A as a row vector. We denote matrix norms by $\|A\|_\xi$, using subscripts to distinguish between various norms. Of particular interest will be the Frobenius norm, the square of which is $\|A\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n A_{ij}^2$, and the spectral norm, which is defined by $\|A\|_2 = \sup_{x \in \mathbb{R}^n,\ x \neq 0} \frac{|Ax|}{|x|}$. These norms are related to each other as: $\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\,\|A\|_2$. If the SVD of A is $A = U\Sigma V^T = \sum_{t=1}^{\rho} \sigma_t u^t v^{t^T}$, where ρ is the rank of A, then for $k \leq \rho$ define $A_k = \sum_{t=1}^{k} \sigma_t u^t v^{t^T}$.

2.2 Review of Approximating a Matrix as the Product $C\tilde{U}R$

In [14] we presented and analyzed two algorithms to compute compressed approximate decompositions of a matrix $A \in \mathbb{R}^{m \times n}$. The second approximation (computed with the ConstantTimeCUR algorithm of [14]) is of the form $A' = C\tilde{U}R$, where C is an $m \times c$ matrix consisting of c randomly picked (and suitably rescaled) columns of A, R is an $r \times n$ matrix consisting of r randomly picked (and suitably rescaled) rows of A; the algorithm constructs the $w \times c$ matrix W consisting of w randomly picked (and suitably rescaled) rows of C, and from the SVD of $W^T W$ constructs the $c \times r$ matrix $\tilde{U}$. The $C\tilde{U}R$ approximation may be defined after making three passes through the data matrix A, and $\tilde{U}$ can be constructed using additional RAM space and time that is $O(1)$. In the following theorem we let $c = w = r = s$ for simplicity. Note also that γ is a parameter and k is the rank of the approximation; see [14] for a full discussion and definition of notation.

Theorem 1. *Suppose $A \in \mathbb{R}^{m \times n}$ and let C, $\tilde{U}$, and R be constructed from the* ConstantTimeCUR *algorithm by sampling s columns of A (and then sampling s rows of C) and s rows of A. Let $\delta, \epsilon > 0$. If a spectral norm bound is desired, and hence the* ConstantTimeCUR *algorithm of [14] is run with $\gamma = \Theta(\epsilon)$ and $s = \Omega\left(1/\epsilon^8\right)$, then under appropriate assumptions on the sampling probabilities we have that with probability at least $1 - \delta$ each of the following holds:* $\|C\|_F = \|A\|_F$, $\left\|\tilde{U}\right\|_2 \leq O(1/\epsilon)/\|A\|_F$, $\|R\|_F = \|A\|_F$, *and* $\left\|A - C\tilde{U}R\right\|_2$

$\leq \|A - A_k\|_2 + \epsilon \|A\|_F$. *Thus, if we choose* $k = 1/\epsilon^2$ *(and* $s = \Omega\left(1/\epsilon^8\right)$*) then with probability at least* $1 - \delta$

$$\left\|A - C\tilde{U}R\right\|_2 \leq \epsilon \|A\|_F . \tag{2}$$

3 Sampling Sub-programs of a Linear Program

In this section, we examine relating the feasibility or infeasibility of a Linear Program to the feasibility or infeasibility of a randomly sampled version of that LP.

Theorem 2. *Let* P *be a* $r \times n$ *matrix and* b *be a* $r \times 1$ *vector. Let* $P^{(i)}$ *denote the* i*-th column of* P *and consider the following Linear Program:*

$$Px = \sum_{i=1}^{n} P^{(i)} x_i \leq b \qquad 0 \leq x_i \leq c_i. \tag{3}$$

Suppose q *is a positive integer and* Q *is a random subset of* $\{1, 2, \ldots n\}$ *with* $|Q| = q$ *formed by picking elements of* $\{1, 2, \ldots n\}$ *in* q *i.i.d. trials, where, in each trial, the probability of picking the* i*-th element is*

$$p_i = \mathbf{Pr}\left[i_t = i\right] = \frac{\left|P^{(i)}\right|^2}{\|P\|_F^2}. \tag{4}$$

Let $\mathbf{1}_r$ *denote the* $r \times 1$ *all-ones vector. If the Linear Program (3) is feasible then, with probability at least* $1 - \delta$

$$\sum_{i \in Q} \frac{1}{qp_i} P^{(i)} x_i \leq b + \frac{1}{\delta\sqrt{q}} |x| \, \|P\|_F \mathbf{1}_r \qquad 0 \leq x_i \leq c_i \ , \ i \in Q \tag{5}$$

is feasible as well. If the Linear Program (3) is infeasible then, with probability at least $1 - \delta$

$$\sum_{i \in Q} \frac{1}{qp_i} P^{(i)} x_i \leq b - \frac{1}{\delta\sqrt{q}} |x| \, \|P\|_F \mathbf{1}_r \qquad 0 \leq x_i \leq c_i \ , \ i \in Q \tag{6}$$

is infeasible as well.

Proof: We first claim that Px is well approximated by $\tilde{P}\tilde{x} = \sum_{i \in Q} \frac{1}{qp_i} P^{(i)} x_i$, i.e., that

$$\mathbf{Pr}\left[\left\|Px - \tilde{P}\tilde{x}\right\|_F \leq \frac{1}{\delta\sqrt{q}} |x| \, \|P\|_F\right] \geq 1 - \delta. \tag{7}$$

To establish the claim, first note that

$$\mathbf{E}\left[\left\|Px - \tilde{P}\tilde{x}\right\|_F\right] \leq \left(\frac{1}{q} \sum_{i=1}^{n} \frac{1}{p_i} \left|P^{(i)}\right|^2 |x_i|^2\right)^{1/2} \leq \frac{1}{\sqrt{q}} |x| \, \|P\|_F , \tag{8}$$

and then apply Markov's inequality. (5) then follows immediately since there exists a vector v such that with high probability $\tilde{P}\tilde{x} = Px + v$, where $|v| \leq \frac{1}{\delta\sqrt{q}} |x| \|P\|_F$. (6) follows by using LP duality. For details (and for a related theorem), see [15].

◇

Note that establishing (7) in Theorem 2 uses ideas that are very similar to those used in [12] for approximating the product of two matrices. Once we are given (7) then the proof of (5) is immediate; we simply show that if the original LP has a solution then the sampled LP also has a solution since $\tilde{P}\tilde{x}$ is sufficiently close to Px. On the other hand, proving (6) is more difficult; we must show that the non-existence of a solution of the original LP implies the same for the randomly sampled version of the LP. Fortunately, by LP duality theory the non-existence of a solution in the LP implies the existence of a certain solution in a related LP.

A special case of these results occurs when $c_i = 1$ for all i, since in that case the Cauchy-Schwartz inequality implies $\sum_{i=1}^{n} \left|P^{(i)}\right| \leq \sqrt{n} \|P\|_F$. The induced LPs (5) and (6) in Theorem 2 may then be replaced by

$$\sum_{i \in Q} \frac{1}{qp_i} P^{(i)} x_i \leq b \pm \frac{1}{\delta} \sqrt{\frac{n}{q}} \|P\|_F \mathbf{1}_r \qquad 0 \leq x_i \leq c_i \ , \ i \in Q. \tag{9}$$

4 An Approximation Algorithm for Max-cut

In this section we present and analyze a new approximation algorithm for the Max-Cut problem. Recall that the Max-Cut of a graph G with weighted adjacency matrix A is:

$$\textbf{MAX-CUT}\,[G] = \textbf{MAX-CUT}\,[A] = \max_{x \in \{0,1\}^n} x^T A(\mathbf{1}_n - x), \tag{10}$$

where $\mathbf{1}_n$ is the all-ones $n \times 1$ vector and x is the characteristic vector of the cut, i.e., it is a 0-1 vector whose i-th entry denotes whether vertex i is on the left or right side of the cut.

4.1 The Algorithm

Consider the ApproximateMaxCut algorithm which is presented in Figure 1 and which takes as input an $n \times n$ matrix A, which is the weighted adjacency matrix of a weighted undirected graph G on n vertices, and computes an approximation Z_{LPQ} to **MAX-CUT** $[A]$. In order to compute Z_{LPQ}, the ApproximateMaxCut algorithm uses the ConstantTimeCUR algorithm of [14] to compute a constant-sized description of three matrices, C, $\tilde{U}$, and R, whose product $C\tilde{U}R \approx A$. In addition, from the (not explicitly constructed) matrices C and R two constant-sized matrices, denoted $\tilde{C}$ and $\tilde{R}$, consisting of q rows of C and the corresponding q columns of R, each appropriately rescaled, are

ApproximateMaxCut Algorithm

Input: $A \in \mathbb{R}^{n \times n}$, the weighted adjacency matrix of a graph $G = (V, E)$, and ϵ, an error parameter.

Output: Z_{LPQ}, an approximation to **MAX-CUT** $[A]$.

- Let $s = \Theta(1/\epsilon^8)$ be the number of columns/rows of A that are sampled for the $C\tilde{U}R$ approximation, let $q = \text{poly}(1/\epsilon)\exp(\text{poly}(1/\epsilon))$ be the dimension of the randomly sampled Linear Program, and let Q be the set of indices of the sampled variables.
- Compute (using the ConstantTimeCUR algorithm of [14]) and store the $s \times s$ matrix $\tilde{U}$.
- Compute and store the matrices $\tilde{C}$ and $\tilde{R}$.
- Construct all possible vector pairs $(u, v) \in [-\sqrt{n}\,\|A\|_F, \sqrt{n}\,\|A\|_F]^{2s}$ in increments of $(\epsilon/4s)\sqrt{n}\,\|A\|_F$. Let Ω_Δ be the set of all such pairs.
- For every pair $(u, v) \in \Omega_\Delta$ check whether the Linear Program $\mathbf{LP}_Q(u, v)$

$$u - \tfrac{\epsilon}{4s}\sqrt{n}\,\|A\|_F\,\mathbf{1}_s \leq \quad \textstyle\sum_{i\in Q} \frac{1}{qw_i} C_{(i)} x_i \quad \leq u + \tfrac{\epsilon}{4s}\sqrt{n}\,\|A\|_F\,\mathbf{1}_s$$

$$v - \tfrac{\epsilon}{4s}\sqrt{n}\,\|A\|_F\,\mathbf{1}_s \leq R\mathbf{1}_n - \textstyle\sum_{i\in Q} \frac{1}{qw_i} R^{(i)} x_i \leq v + \tfrac{\epsilon}{4s}\sqrt{n}\,\|A\|_F\,\mathbf{1}_s$$

$$x_i \in [0, 1], \quad i \in Q$$

  is feasible, and select $(\bar{u}, \bar{v})$ such that $u^T \tilde{U} v$ is maximized among all feasible pairs.
- Return $Z_{LPQ} = \bar{u}^T \tilde{U} \bar{v}$.

Fig. 1. The ApproximateMaxCut Algorithm

constructed. These matrices are used in the construction of the linear programs $\mathbf{LP}_Q(u, v)$; the algorithm then checks whether a constant number of these LPs (each on a constant number q of variables) are feasible and returns the maximum of an easily-computed function of the feasible vectors as the approximation Z_{LPQ} of **MAX-CUT** $[A]$.

4.2 Analysis of the Implementation and Running Time

The ApproximateMaxCut algorithm may be implemented with three passes over the matrix and constant (with respect to n, but exponential in $1/\epsilon$) additional space and time. For details, see [15].

4.3 The Main Theorem

Theorem 3 is our main theorem; note that the 3/4 is arbitrary and can be boosted to any number less than 1 using standard methods.

Theorem 3. *Let A be the $n \times n$ weighted adjacency matrix of a graph $G = (V, E)$, let ϵ be fixed, and let Z_{LPQ} be the approximation to the* **MAX-CUT** $[A]$

returned by the ApproximateMaxCut *algorithm. Then, with probability at least* 3/4

$$|Z_{LPQ} - \mathbf{MAX\text{-}CUT}\,[A]| \leq \epsilon n \|A\|_F .$$

The algorithm makes three passes, i.e., three sequential reads, through the matrix A and then uses constant (with respect to n) additional space and constant additional time. The algorithm chooses a random sample of A according to a nonuniform probability distribution.

4.4 Intuition Behind the Proof of Theorem 3

In order to prove Theorem 3, we will require four levels of approximation, and we will have to show that each level does not introduce too much error. We note that the algorithm and its analysis use ideas similar to those used in [17, 2] for the case of uniform sampling.

In the *first* level of approximation we will use the ConstantTimeCUR algorithm of [14] and sample $s = \Theta(1/\epsilon^8)$ rows and columns of the matrix A in order to compute a constant-sized description of C, $\tilde{U}$, and R. As discussed in [14] the description consists of the explicit matrix $\tilde{U}$ and labels indicating which s columns of A and which s rows of A are used in the construction of C and R, respectively. From Theorem 1 we see that under appropriate assumptions on the sampling probabilities (which we will satisfy) $C\tilde{U}R$ is close to A in the sense that $\left\|A - C\tilde{U}R\right\|_2 \leq \epsilon \|A\|_F$ with high probability. A good approximation to $\mathbf{MAX\text{-}CUT}\,[A]$ is provided by $\mathbf{MAX\text{-}CUT}\left[C\tilde{U}R\right]$, which is the Max-Cut of the graph whose weighted adjacency matrix is $C\tilde{U}R$. Thus, it suffices to approximate well

$$\mathbf{MAX\text{-}CUT}\left[C\tilde{U}R\right] = \max_{x\in\{0,1\}^n} x^T C\tilde{U}R(\mathbf{1}_n - x).$$

Since with high probability $\left|C^T x\right| \leq \sqrt{n}\|A\|_F$ and $|R(\mathbf{1}_n - x)| \leq \sqrt{n}\|A\|_F$, both $C^T x$ and $R(\mathbf{1}_n - x)$ lie in $[-\sqrt{n}\|A\|_F, \sqrt{n}\|A\|_F]^s$. Consider the set of vectors $(u, v) \in \Omega$, where $\Omega = [-\sqrt{n}\|A\|_F, \sqrt{n}\|A\|_F]^{2s}$. Suppose we pick the vector pair $(\bar{u}, \bar{v})$ that satisfies the following two conditions:

1. **(feasibility)** There exists a vector $\bar{x} \in \{0,1\}^n$ such that the pair $(\bar{u}, \bar{v})$ satisfies
$$C^T\bar{x} = \bar{u} \qquad \text{and} \qquad R(\mathbf{1}_n - \bar{x}) = \bar{v},$$
2. **(maximization)** $(\bar{u}, \bar{v})$ maximizes the value $u^T\tilde{U}v$ over all such possible pairs.

For such a $(\bar{u}, \bar{v})$ the vector $\bar{x}$ defines a Max-Cut of the graph with weighted adjacency matrix $C\tilde{U}R$ and thus $\bar{u}^T\tilde{U}\bar{v} = \mathbf{MAX\text{-}CUT}\left[C\tilde{U}R\right]$.

We will then perform a *second* level of approximation and discretize the set of candidate vector pairs. Let Ω_Δ denote the discretized set of vector pairs, where the discretization Δ is defined below. Consider the set of vectors $(u, v) \in \Omega_\Delta$ and suppose we pick the vector pair $(\bar{u}, \bar{v})$ that satisfies the following two conditions:

1'. (approximate feasibility) There exists a vector $\bar{x} \in \{0,1\}^n$ such that the pair $(\bar{u}, \bar{v})$ satisfies

$$\bar{u} - \frac{\epsilon}{s}\sqrt{n}\,\|A\|_F\,\mathbf{1}_s \leq \quad C^T\bar{x} \quad \leq \bar{u} + \frac{\epsilon}{s}\sqrt{n}\,\|A\|_F\,\mathbf{1}_s \text{ and}$$
$$\bar{v} - \frac{\epsilon}{s}\sqrt{n}\,\|A\|_F\,\mathbf{1}_s \leq R(\mathbf{1}_n - \bar{x}) \leq \bar{v} + \frac{\epsilon}{s}\sqrt{n}\,\|A\|_F\,\mathbf{1}_s,$$

i.e., there exists a vector $\bar{x} \in \{0,1\}^n$ such that the pair $(\bar{u}, \bar{v})$ satisfies

$$C^T\bar{x} \approx \bar{u} \qquad \text{and} \qquad R(\mathbf{1}_n - \bar{x}) \approx \bar{v},$$

2'. (maximization) $(\bar{u}, \bar{v})$ maximizes the value $u^T\tilde{U}v$ over all such possible pairs.

In this case, we are checking whether every vector pair $(u, v) \in \Omega_\Delta$ in the discretized set is approximately feasible in the sense that a nearby vector pair $(u, v) \in \Omega$ satisfies the feasibility condition exactly. If we choose the discretization in each dimension as $\Delta = \frac{\epsilon}{4s}\sqrt{n}\,\|A\|_F$, then every vector pair $(u, v) \in \Omega$ is near a vector pair $(u, v) \in \Omega_\Delta$. Thus, (subject to a small failure probability) we will not "miss" any feasible vector pairs, i.e., for every exactly feasible $(u, v) \in \Omega$ there exists some approximately feasible $(u, v) \in \Omega_\Delta$. Equally importantly, with this discretization, we only have to check a large but constant number of candidate vector pairs. Taking the maximum of $u^T\tilde{U}v$ over the feasible vector pairs $(u, v) \in \Omega_\Delta$ will lead to an approximation of **MAX-CUT** $\left[C\tilde{U}R\right]$. At this point we have reduced the problem of approximating **MAX-CUT** $[A]$ to that of checking the feasibility of a large but constant number of IPs and returning the maximum of an easily computed function of them.

Next, we reduce this to the problem of checking the feasibility of a large but constant number of constant-sized LPs on a constant number q of variables and returning the maximum of an easily computed function of them. We do this in two steps. First, we will perform a *third* level of approximation and consider the LP relaxation of the IP. Since this LP has a very special structure, i.e., even though the LP lies in an n-dimensional space the number of the constraints is a constant independent of n, there exists a feasible solution for the LP that has at most a constant number of non-integral elements. We will exploit this and will consider an LP which is a slight tightening of the LP relaxation of the IP; we will prove that if the IP is infeasible then the LP is infeasible as well. Then, we will perform a *fourth* level of approximation and construct a constant-sized randomly-sampled LP on a constant number q of variables, such that the infeasibility of the LP on n variables implies, with probability at least $1-\delta^*$, the infeasibility of the LP on q variables.This last level of approximation involves sampling and thus a failure probability. Since there are a constant number $\left(\frac{8s}{\epsilon}\right)^{2s}$ of values of $(u, v) \in \Omega_\Delta$ to check, by choosing δ^* to be a sufficiently small constant independent of n, the probability that any one of the large but constant number of sampling events involved in the construction of the constant-sized LPs will fail can be bounded above by a small constant. Theorem 3 will then follow.

For details of the proof, see [15].

References

1. N. Alon, W.F. de la Vega, R. Kannan, and M. Karpinski. Random sampling and approximation of MAX-CSP problems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 232–239, 2002.
2. N. Alon, W.F. de la Vega, R. Kannan, and M. Karpinski. Random sampling and approximation of MAX-CSPs. *Journal of Computer and System Sciences*, 67:212–243, 2003.
3. S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 284–293, 1995.
4. Z. Bar-Yossef. *The Complexity of Massive Data Set Computations*. PhD thesis, University of California, Berkeley, 2002.
5. Z. Bar-Yossef. Sampling lower bounds via information theory. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 335–344, 2003.
6. W.F. de la Vega. MAX-CUT has a randomized approximation scheme in dense graphs. *Random Structures and Algorithms*, 8(3):187–198, 1996.
7. W.F. de la Vega and M. Karpinski. A polynomial time approximation scheme for subdense MAX-CUT. Technical Report TR02-044, Electronic Colloquium on Computational Complexity, 2002.
8. P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291–299, 1999.
9. P. Drineas and R. Kannan. Fast Monte-Carlo algorithms for approximate matrix multiplication. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 452–459, 2001.
10. P. Drineas and R. Kannan. Pass efficient algorithms for approximating large matrices. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 2003.
11. P. Drineas, R. Kannan, and M.W. Mahoney. Sampling sub-problems of heterogeneous MAX-2-CSP problems and approximation algorithms. *manuscript*.
12. P. Drineas, R. Kannan, and M.W. Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. Technical Report YALEU/DCS/TR-1269, Yale University Department of Computer Science, New Haven, CT, February 2004.
13. P. Drineas, R. Kannan, and M.W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. Technical Report YALEU/DCS/TR-1270, Yale University Department of Computer Science, New Haven, CT, February 2004.
14. P. Drineas, R. Kannan, and M.W. Mahoney. Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. Technical Report YALEU/DCS/TR-1271, Yale University Department of Computer Science, New Haven, CT, February 2004.
15. P. Drineas, R. Kannan, and M.W. Mahoney. Sampling sub-problems of heterogeneous Max-Cut problems and approximation algorithms. Technical Report YALEU/DCS/TR-1283, Yale University Department of Computer Science, New Haven, CT, April 2004.
16. A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 12–20, 1996.

17. A. Frieze and R. Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.
18. O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 339–348, 1996.
19. G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.
20. R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, New York, 1985.
21. J.S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer-Verlag, New York, 2001.
22. M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.
23. G.W. Stewart and J.G. Sun. *Matrix Perturbation Theory*. Academic Press, New York, 1990.

Truthful Approximation Mechanisms for Scheduling Selfish Related Machines

Nir Andelman[1,⋆], Yossi Azar[2,⋆⋆], and Motti Sorani[2]

School of Computer Science, Tel Aviv University, Tel Aviv, 69978, Israel
andelman@cs.tau.ac.il, {azar, mottis}@tau.ac.il

Abstract. We consider the problem of scheduling jobs on related machines owned by selfish agents. Previously, Archer and Tardos showed a 2-approximation randomized mechanism which is truthful in expectation only (a weaker notion of truthfulness). We provide a 5-approximation deterministic truthful mechanism, the first deterministic truthful result for the problem.

In case the number of machines is constant, we provide a deterministic Fully Polynomial Time Approximation Scheme (FPTAS) algorithm, and a suitable payment scheme that yields a truthful mechanism for the problem. This result, which is based on converting FPTAS to monotone FPTAS, improves a previous result of Auletta et al, who showed a $(4+\varepsilon)$-approximation truthful mechanism.

1 Introduction

The emergence of the Internet as the platform for distributed computation changed the point of view of the algorithm designer [14, 15]. The old implicit assumption that the participating machines (agents) act as instructed can no longer be taken for granted. As the machines over the Internet are controlled by different users, they will likely to do what is most beneficial to their owners, manipulate the system and lie when it is possible to maximize their own profit. Where optimization problems are concerned, results can be severe and unexpected when false information is introduced to the classic optimization algorithms, due to the selfish behavior of the agents.

In this paper we deal with the problem *Minimum Makespan* for scheduling jobs on related machines, (also known as $Q||C_{max}$). The *system* allocates jobs with arbitrary sizes to the machines, where each machine is owned by an *agent*. The machines have different speeds, known to their owner only. At first phase, the agents declare their speeds, then given these bids the system allocates the jobs to the machines. The objective of the system is to minimize the makespan. The classic scheduling problem (when information is complete and accurate) is

⋆ Research supported in part by the Israel Science Foundation.
⋆⋆ Research supported in part by the German-Israeli Foundation.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 69–82, 2005.

known to be NP-Complete. Hence research focused on obtaining a polynomial time approximation algorithms for this problem.

The field of *Mechanism Design* provides a scheme to overcome the selfishness of the agents, mainly by paying the agents in order to force them to declare their true properties, thus helping the system to solve the optimization problem correctly. The most famous result in this area is the Vickrey-Clarke-Groves (VCG) Mechanism [6, 16, 9] which applies to *utilitarian* objective only (the sum of the agent's valuations). The Minimum Makespan problem is not utilitarian as we are seeking to minimize the maximum load, not the sum of the loads.

Several optimization problems were re-considered in the context of selfish agents [12]. Even in cases where truthful tools such as VCG are at hand, it turned out that applying them to combinatorial optimization problems is computationally intractable. Ronen and Nisan [13] showed that if the optimal outcome is replaced by the result of a computationally tractable approximation algorithm then the resulting mechanism is no longer necessarily truthful. New attitudes are required to achieve approximation who still retain truthfulness.

Archer and Tardos introduced a framework for designing a truthful mechanism for one-parameter agents [3]. In particular they considered the fundamental problem of scheduling on related machines, and showed a randomized 3-approximation truthful mechanism, later improved to a 2-approximation [4]. Their mechanism utilizes a weaker notion of truthfulness, as it is truthful in expectation only.

1.1 Results in This Paper

Our main results are the following

- We show a deterministic 5-approximation truthful mechanism for scheduling jobs on arbitrary number of related machines.
- We show deterministic truthful FPTAS for scheduling jobs on a fixed number of machines.

All results follow the framework of Archer and Tardos, introducing monotone algorithms together with a *payments scheme* computable in polynomial time.

Our truthful mechanisms are deterministic. Hence truth-telling is a dominant strategy over all possible strategies of an agent. This truthfulness, analogous to universal truthfulness for randomized mechanisms, is stronger than the one use in the 3-approximation randomized mechanism in [3], as the latter is truthful in expectation only.

We also show the existence of truthful deterministic PTAS and FPTAS mechanisms for any fixed number of related machines. Our mechanisms improve the result of the $(4 + \varepsilon)$-approximation truthful mechanism for constant number of machines introduced in [5, 2]. We present both mechanisms since our PTAS is fairly simple to implement and may be more efficient than the FPTAS if the required approximation ratio $1 + \varepsilon$ is moderate.

1.2 Previous Work

The classic problem of scheduling jobs on parallel related machines was dealt by several approximation approaches. The known basic result of an LPT algorithm which sorts the jobs in non-increasing order of job size, then allocates a job to the machine which will be least busy afterwards is 2-approximation [8]. An FPTAS was first introduced by Horowitz and Sahni for the case where the number of machines is constant. Their approach was based on rounding and exact solution by dynamic programming [11]. Finally, in the late 80's Hochbaum and Shmoys introduced a PTAS for the general case of an arbitrary number of machines [10, 7]. Since the problem is strongly NP-Complete, no FPTAS is possible for the general case, and their result remains the best possible, unless P=NP.

Scheduling with selfish agents was first analyzed by Ronen and Nisan. Their results mainly concern scheduling on unrelated machines, known also as $R||C_{max}$. Our problem was first tackled by Archer and Tardos [3] who showed that the former known approximation algorithms for the problem are not truthful. They introduced a truthful randomized mechanism which achieves a 3-approximation to the problem. This approach achieves truthfulness with respect to the *expected profit* only. Thus it possible that even though the expected profit is maximized when telling the truth, there might exist a better (untruthful) strategy.

The first deterministic result is due to Auletta et al [5]. They show a deterministic truthful mechanism which is $(4+\varepsilon)$-approximation for any *fixed* number of machines. The case of arbitrary number of machines remained open previous to our paper.

A different approach by Nisan and Ronen introduces another model in which the mechanism is allowed to observe the machines process their jobs and compute the payments afterwards. Using these *mechanisms with verification* [12] allows application of penalty on lying agents, and was shown to cope well with the existing known approximation algorithms.

2 Preliminaries

We consider the problem of scheduling jobs on related machines. We are given a number of machines, m, and a job sequence with sizes $\sigma = (p_1, p_2, ..., p_n)$. Each machine, owned by an agent, has a *speed* s_i known only to its agent. Alternatively, the secret (sometimes called *type*) of each agent is $t_i = 1/s_i$ which is the number of time units required to process one unit of work (or the *cost per unit of work*). Thus the processing time of job p_j on machine i is $p_j t_j$. The *work* of machine i, denoted by w_i is given by the sum of the processing time of jobs assigned to it (the total work assigned to it). We assume a machine incurs a cost proportional to the total processing time spent. The output is an assignment of jobs to machines. The mechanism's goal is to minimize the maximum completion time over all machines.

A *mechanism* for this problem is a pair $M = (A, P)$, where A is an algorithm to allocate jobs to the machines (agents) and P is a payment scheme. The

mechanism asks each agent to report its bid (their cost per unit of work). Based on these reports, the mechanism uses A to construct an allocation, and pays according to P.

A *strategy* for agent i is to declare a value b_i as its cost per unit of work (which in principle can be larger or smaller than t_i). Let b to be the vector of bids, and b_{-i} denote the vector of bids not including b_i, i.e. $(b_1, b_2, .., b_{i-1}, b_{i+1}, ..., b_m)$. Denote by (b_{-i}, α) the vector generated by inserting the value α to the vector b_{-i}. Notice that if we view b as a sorted vector, then (b_{-i}, α) corresponds also to a sorted vector (thus the index of α might be different than i).

The output of the algorithm $o(b)$ is an allocation of the jobs to the agents, and the profit of agent i is defined as $profit_i(t_i, b) = P_i(b) - t_i w_i(o(b))$. A strategy b_i is *(weakly) dominant* for agent i if bidding b_i always maximizes his profit, i.e. $profit_i(t_i, (b_{-i}, b_i)) \geq profit_i(t_i, (b_{-i}, b'_i))$ for all b_{-i}, b'_i. A mechanism is *truthful* if truth-telling is a dominant strategy for each agent (i.e. t_i is a dominant strategy for all i)

We assume w.l.o.g that the vector of bids b is sorted in non-increasing order of speed (non-decreasing order of cost per unit of work), breaking ties by the original index.

An algorithm A is a c-approximation algorithm if for every instance (σ, t) , $cost(A, \sigma, t) \leq c \cdot opt(\sigma, t)$. For our problem the cost is the maximum completion time. An c-approximation mechanism is one whose output algorithm is an c-approximation. A PTAS (Polynomial-time approximation scheme) is a family of algorithms such that for every $\varepsilon > 0$ there exists a $(1 + \varepsilon)$-approximation algorithm. If the running time is also polynomial in $1/\varepsilon$, the family of algorithms is an FPTAS.

A vector $(v_1, v_2, .., v_n)$ is *lexicographically smaller* than $(u_1, u_2, .., u_n)$ if for some i, $v_i < u_i$ and $v_k = u_k$ for all $1 \leq k < i$.

2.1 Monotone Algorithms

Archer and Tardos showed necessary and sufficient conditions to obtain a truthful mechanism [3].

Definition 1. *Fixing the other agents bids b_{-i}, we define the work-curve for agent i as $w_i(b_{-i}, b_i)$, namely a single-variable function of b_i. A work-curve is decreasing if $w_i(b_{-i}, b_i)$ is a decreasing function of b_i for all b_{-i}.*

A decreasing work-curve means that when an agent bids lower (saying he is faster) more or equal amount of work should be allocated to his machine, given that the other agents' bids are fixed. A *monotone algorithm* is an algorithm that produces an assignment which preserves the decreasing work-curve property for all agents. Several classic approximation algorithms fail to keep this monotonicity, among them the greedy algorithm, and the classic PTAS of Horowitz and Sahni [3, 5].

Definition 2. *A mechanism satisfies the voluntary participation condition if agents who bid truthfully never incur a net loss, i.e. $profit_i(t_i, (b_{-i}, t_i)) \geq 0$ for all agents i, true values t_i and other agents' bids b_{-i}.*

Input: a job sequence σ , and a non-decreasing sorted bids vector $b = (b_1, b_2, .., b_m)$
Output: An allocation of jobs to the m machines

1. Set $d_1 = \frac{5}{8}b_1$
2. For $j \geq 2$, round up the bids to the closest value of $b_1 \cdot 2.5^i$, which is larger than the original bid, i.e. $d_j = b_1 \cdot 2.5^i$, where $i = \left\lfloor log_{2.5} \frac{b_j}{b_1} \right\rfloor + 1$
3. Sort the job sequence in non-increasing order
4. Calculate the valid fractional assignment for the job sequence σ given the new bids-vector d. Let T^f be the value of the fractional solution.
5. For each machine $j = 1 \ldots m$, assign jobs in non-increasing order of job-size to machine j (using bid d_j), until machine j exceeds threshold T^f (or equals it)
6. Return the assignment

Fig. 1. Monotone-RF

Theorem 1. *[3] A mechanism is truthful and admits a voluntary participation if and only if the work-curve of each agent is decreasing,* $\int_0^\infty w_i(b_{-i}, u)du < \infty$ *for all i,* b_{-i} *and the payments in this case should be*

$$P_i(b_{-i}, b_i) = b_i w_i(b_{-i}, b_i) + \int_{b_i}^{\infty} w_i(b_{-i}, u)du \tag{1}$$

Therefore in order to achieve a truthful mechanism we need to design a monotone algorithm, and use the payment scheme as in (1). Since truthfulness is reduced to designing a monotone algorithm, we may assume, for the sake of the monotonicity proof, that the bids are equal to the real speeds.

3 Truthful Approximation for Arbitrary Number of Machines

A classic approximation algorithm for the problem, forms a "valid" fractional allocation of the jobs to the machines, and then uses a simple rounding to get a 2-approximation for the problem. In [3] it has been shown that this simple algorithm is not monotone, thus not truthful.

The main result of this section is a deterministic monotone algorithm which is based on the fractional assignment. Algorithm *Monotone-RF (Monotone Rounded Fractional)*, shown in figure 1, is shown to be a 5-approximation algorithm.

Given a threshold T, we can treat the machines as bins of size T/b_i. A *fractional assignment* of the jobs to the machines, is a partition of each job j into pieces whose sizes sum to p_j and an assignment of these pieces to the machines (bins). A fractional assignment is *valid* if each bin is large enough to contain the sum of all pieces assigned to it, and for every piece assigned to it, it is capable of containing the entire job the piece belongs to. The smallest threshold for which there exist a valid fractional assignment, T^f is a lower bound to the optimal solution, and can be calculated exactly in the following manner (see [3]):

$$T^f = \max_j \min_i \max \{b_i p_j, \frac{\sum_{k=1}^{j} p_k}{\sum_{l=1}^{i} \frac{1}{b_l}}\} \quad (2)$$

This valid fractional assignment with respect to this threshold is obtained by sorting the jobs in non-increasing order, and allocating them to the machines (ordered in non-increasing order of their speeds). Some jobs are sliced between two machines when the threshold is exceeded in the middle of a job.

An important property of the valid fractional assignment is the monotonicity of T^f: as we increase the speed of a machine, T^f is not increased. Let $T^f(b_{-i}, b_i)$ be the the smallest threshold for which there exist a valid fractional assignment, given the bids vector (b_{-i}, b_i).

Observation 1. *[3]* $T^f(b_{-i}, \alpha b_i) \leq \alpha T^f(b_{-i}, b_i)$ *for all* $\alpha > 1$ *and* i.

Lemma 1. *For any machine* i *which is not the fastest* $(i > 1)$*, and for any rounded bids vector* d*:*

$$T^f(d_{-i}, d_i) \leq T^f(d_{-i}, \beta) \leq \frac{5}{4} T^f(d_{-i}, d_i)$$

for all $\beta \geq d_i$

Proof. The first inequality is straight-forward as the allocation for (d_{-i}, β) is also a valid fractional assignment for (d_{-i}, d_i), given any fixed threshold T.

As for the second inequality, we generate a valid fractional assignment which allocates zero work to machine i. This allocation would use a threshold at most $\frac{5}{4} T^f(d_{-i}, d_i)$. Since this allocation is a valid fractional assignment for (d_{-i}, β), the minimal threshold for (d_{-i}, β) might only be smaller than the generated one.

To form an allocation which does not use machine i, for every $2 \leq j \leq i$ take all the pieces previously assigned to machine j and assign them to machine $(j-1)$. The first machine is now allocated the pieces originally assigned to the second machine, along with its original assignment. Since the algorithm assures that $4d_1 \leq d_2$, the assignment is clearly valid, with a threshold which is at most $\frac{5}{4} T^f(d_{-i}, d_i)$. □

We note that Lemma 1 holds for rounded bids vectors created by Monotone-RF, but does not hold in general. The following lemmata consider several scenarios in which machine i slows down. We denote by d' the rounded bid vector obtained after the machine's slowdown. Let i' be the index of the slowing-down machine in d'; Notice that i' might be different than i. We denote by w'_j the total work allocated to machine j given the new bids vector d'. We denote by v_i the rounded speed of machine i, i.e. $v_i = 1/d_i$.

Lemma 2. *Using Algorithm Monotone-RF, when machine* i *which is not the fastest slows down, the total amount of work assigned to the machines faster than it can not decrease. i.e.*

$$\sum_{k=1}^{i-1} w_k \leq \sum_{k=1}^{i-1} w'_k \leq \sum_{k=1}^{i'-1} w'_k$$

Proof. If the rounded bid of machine i is the same as before the slowdown, the assignment is not changed. Thus we consider the case where the new rounded bid is different than the one before the slowdown. Let β be the rounded bid of machine i where $\beta > d_i$. Let i' be the new index of the slowing machine in d'. Clearly $i \leq i'$.

By Lemma 1, $T^f(d_{-i}, \beta) \geq T^f(d_{-i}, d_i)$, i.e. the new threshold used by algorithm Monotone-RF can only increase due to the slowdown. By induction the index of the last job assigned to each of the first $i-1$ machines can be equal, or higher after the slowdown. Thus the total amount of work assigned to the first $i-1$ machines is the same or higher, and the amount of work assigned to the first $i'-1$ machines can only be higher than that. □

Lemma 3. *If the fastest machine slows down yet remains the fastest, the amount of work assigned to it can only decrease.*

Proof. We observe how the bin size of the first machine changes as its speed decreases gradually. As long as the value of $\lfloor \log_{2.5} \frac{b_j}{b_1} \rfloor$ does not change for all $j \geq 2$, all rounded speeds change proportionally, i.e. there is some constant $c > 1$ such that $d' = c \cdot d$. Therefore, the same fractional assignment is calculated (with a new threshold of cT^f) with the same sizes for bins. In the breakpoints where the rounded bid of at least one machine is cut down by 2.5, by Observation 1 the threshold cannot increase, and therefore the bin size of the first machine can only decrease.

Since the fastest machine is always assigned the first jobs, a decrease in its bin size can only decrease the number of jobs assigned to it, and therefore the amount of work assigned to it in the integral assignment also decreases. □

Definition 3. *Given an assignment of jobs by algorithm Monotone-RF, we classify the machines in the following way:*

Full machine *a machine (bin) which the total processing time of the jobs assigned to it is at least its threshold.*

Empty machine *a machine with no jobs allocated to it*

Partially-full machine *a non-empty machine (bin) which is not full. (There is at most one partially-full machine)*

Lemma 4. *When machine i decreases its speed (increases its bid) the total work allocated to it by algorithm Monotone-RF can not increase.*

Proof. Lemma 3 proves the claim when machine i is the fastest machine and remains the fastest. If machine i is not the fastest machine but its rounded bid d_i does not change, then the slowdown has no effect since the same assignment is generated. It remains to prove the claim for the breakpoints, which are when the fastest machine becomes the non-fastest, and when the rounded bid is multiplied

by 2.5. We prove the claim for each step, thus the claim holds for the entire slowdown.

Consider the following cases for the class of machine i before the slowdown:

1. Machine i is the fastest machine ($i = 1$), but after the slowdown another machine j becomes the fastest - we observe the breakpoint where both machines have the same speed and add an artificial stage to the slowdown where the title of the fastest machine passes from i to j (without having the speeds change). The same threshold is calculated in both cases, only the order of the machines changes. The amount of work allocated to machine i when it is considered the fastest is at least $\frac{8}{5} \cdot v_1 \cdot T^f$, while after machine j becomes the fastest it is at most $2 \cdot \frac{v_1}{2.5} \cdot T^f = \frac{4}{5} \cdot v_1 \cdot T^f$, and therefore decreases.
2. Machine i is a full machine which is not the fastest - the threshold used for assigning jobs to the machine is T^f. Due to Lemma 1, $T^f(d) \leq T^f(d') \leq \frac{5}{4}T^f(d)$, where d is the rounded bids vector, and d' is the rounded vector after the slowdown. Before the slowdown the amount of work allocated to it was at least $T^f \cdot v_i$, whereas after slowing down it can become at most $2 \cdot (\frac{5}{4} \cdot T^f) \cdot \frac{v_i}{2.5} = T^f \cdot v_i$. If the machine became partially-full or empty after slowing down, the amount of work allocated to it can only be smaller.
3. Machine i is partially-full - if it becomes empty then the claim is trivial, otherwise some jobs are allocated to machine i. Let $i' \geq i$ be the new index of the machine in the sorted order. Due to Lemma 2 the amount of work allocated to machines with a lower index than i' can be no less than the amount before the slowdown (i.e. $\sum_{k=1}^{i-1} w_k \leq \sum_{k=1}^{i'-1} w'_k$), thus leaving less work to be allocated to machine i.
4. Machine i is empty - The machine stays empty due to Lemma 2. □

Lemma 4 shows that the work-curve of agent i is non-increasing. Hence the following theorem is immediate:

Theorem 2. *Algorithm Monotone-RF provides monotone assignment. Hence A Mechanism Design based on Algorithm Monotone-RF and payment scheme as in (1) is truthful.*

We now analyze the approximation provided by algorithm *Monotone-RF*.

Denote by $k^f(i)$ the index of the last job (or a fraction of a job) assigned to machine i in the fractional assignment. Respectively let $k(i)$ be the index of the last job assigned to machine i by *Monotone-RF*.

Lemma 5. *for all i, $k^f(i) \leq k(i)$*

Proof. By induction. The claim clearly holds for $i = 1$ since $T_1 \geq T^f$. Assume the argument is true for machine i. By induction hypothesis $k^f(i) \leq k(i)$. Since allocation is done in non-increasing order of job size, the first job to be allocated to $i + 1$ by our algorithm might be only smaller than the one allocated by the fractional assignment. Moreover, since the allocation exceeds the fractional threshold, at least the same number of jobs will be assigned to machine i. Thus $k^f(i + 1) \leq k(i + 1)$. □

Theorem 3. *Algorithm Monotone-RF is a 5-approximation.*

Proof. Lemma 5 assures that at the end of the run of algorithm *Monotone-RF* all jobs are allocated. Since the speeds were decreased by at most a factor of 2.5, the threshold $T^f(d)$ may be at most 2.5 times the value of the optimal allocation using the unrounded speeds. Since the speed of the fastest machine is increased by a factor of 1.6, the amount of work assigned to the fastest machine in the fractional solution may be at most $1.6 \cdot 2.5 = 4$ times the value of the optimal allocation.

In the integral solution, since the amount of work assigned to the first machine can exceed the bin's size by at most the size of the second bin, and since the first bin is at least 4 times larger than the second bin, the load on the fastest machine can be at most $1.25 \cdot T^f(d)$, and therefore the load on this machine is at most $1.25 \cdot 4 = 5$ times the optimal. For any other machine, the last job can exceed the threshold by at most $T^f(d)$, and therefore the load on any other machine is at most $2 \cdot T^f(d)$, which is at most $2 \cdot 2.5 = 5$ times the optimal. Therefore, a 5-approximation is achieved. □

To conclude, we need to show that calculating the payments given by (1) can be done in polynomial time. We analyze the number of breakpoints of the integral in that expression. According to Lemma 5 the work curve for machine i is zeroed furthermost when the valid fractional assignment does not use machine i. There is no use in assigning jobs to a machine when its bid β is too high even for the smallest job, i.e. $\beta p_n > T^f$. Using the higher bound $T^f \leq np_1 d_1 < np_1 b_1$, we get a zero assignment for $\beta \geq n\frac{p1}{pn} b_1$. The only exception is when the integral is calculated for the fastest machine, where we get a higher bound of $\beta \geq n\frac{p1}{pn} b_2$. While $\beta \leq b_2$, there is a breakpoint whenever $b_j = 2.5^i \beta$, for some i and for any machine $j > 1$. Therefore, for each factor of 2.5, there are at most $m - 1$ breakpoints (one for each of the other machines), while for $\beta > b_2$, there is one breakpoint for each step.

Thus the number of iterations will be $O(\log_{2.5} n\frac{p1}{pn} + m \log_{2.5} \frac{b_2}{b_1})$ for the fastest machine, $O(m \log_{2.5} n\frac{p1}{pn} \frac{b_2}{b_1})$ in total, which is polynomial in the input size.

4 Truthful PTAS-Mechanism for Any Fixed Number of Machines

We now show a truthful mechanism for any fixed number of machines. Due to simplicity of presentation, we normalize the sizes of the jobs such that the total size of all jobs is one, $\sum_{j=1}^{n} p_j = 1$ (as they are known to the mechanism).

Based on the payments scheme as in (1), it is enough to show a monotone PTAS algorithm. The algorithm, *Monotone-PTAS*, is shown in figure 2. This algorithm shares similar ideas of the PTAS variant of Alon et al [1].

Theorem 4. *A Mechanism Design based on Algorithm Monotone-PTAS and payment scheme as in (1) is truthful.*

Proof. It is suffice to show that Algorithm Monotone-PTAS is monotone. Notice that the job sizes in the instance $\sigma^\sharp$ were generated independently from the bids of the agents. It was shown in [3] that the optimal minimum-lexicographic solution is monotone. □

Theorem 5. *The algorithm Monotone-PTAS achieves a Polynomial Time Approximation Scheme (PTAS)*

Proof. We first show that the algorithm is polynomial. The construction of $\sigma^\sharp$ takes a linear time. As for the rest - the construction of $\sigma^\sharp$ ensures that the minimal job size is $\frac{1}{2} \cdot \frac{\varepsilon^2}{m^2}$. Thus the total number of jobs is no more than $\frac{2m^2}{\varepsilon^2}$, a constant. Solving $\sigma^\sharp$ exactly on m machines while enumerating all the possible allocations takes a constant time.

We now analyze the quality of the approximation. First assume, for the purpose of analysis , that both Opt and our algorithm are not using "slow" machines, i.e. machines whose speed is less than $s_{max} \cdot \frac{\varepsilon}{m}$, where s_{max} is the maximal speed. Let T'_{opt} be the optimal solution for this instance, and T' our solution. Since we solve for chunks whose size is no more than $\frac{\varepsilon^2}{m^2}$, unlike Opt who solves for the original sizes, we can suffer an addition in processing time of maximum $\frac{\frac{\varepsilon^2}{m^2}}{s_{max} \cdot \frac{\varepsilon}{m}} = \frac{\varepsilon}{m \cdot s_{max}}$ (i.e. an additional chunk on the slowest machine used). A lower bound on the optimal solution is $T'_{opt} \geq \frac{1}{m \cdot s_{max}}$, Thus $T' \leq (1+\varepsilon)T'_{opt}$.

We now compare T'_{opt} to performance of Opt when the "slow" machines restriction is removed, namely T_{opt}. The total work done by the "slow" machines in opt is bounded above by $s_{max} \cdot \varepsilon T_{opt}$. If we move this amount of work to the fastest machine we pay maximum extra processing time of εT_{opt}, thus $T'_{opt} \leq (1+\varepsilon)T_{opt}$. Combining these two lower bounds we get that $T \leq T' \leq (1+\varepsilon)T'_{opt} \leq (1+\varepsilon)^2 \cdot T_{opt} \leq (1+3\varepsilon)T_{opt}$ for any $\varepsilon < \frac{1}{2}$, a PTAS. □

Input: a job sequence σ ,a bids vector $b = (b_1, b_2, .., b_m)$ and parameter ε
Output: An allocation of jobs to the m machines that achieves a $(1 + 3\epsilon)$-approximation

1. Construct a new instance $\sigma^\sharp$ based on the original job instance, as follows:
   (a) sort the job sequence in non-increasing order
   (b) $\sigma^\sharp = \{p_j \in \sigma | p_j \geq \frac{\varepsilon^2}{m^2}\}$
   (c) merge the rest of jobs in a greedy manner to chunks of size in the range $[\frac{1}{2} \cdot \frac{\varepsilon^2}{m^2}, \frac{\varepsilon^2}{m^2}]$ and add them to $\sigma^\sharp$
2. Solve Minimum Makespan *exactly* with the instance $(\sigma^\sharp, b)$ to obtain the optimal solution. If several optimal allocations exist, choose the one with the lexicographically minimum schedule (where the machines are ordered according to some external machine-id)
3. Return the same allocation achieved on $\sigma^\sharp$. A small job is allocated to the same machine which his chunk has been allocated.

Fig. 2. Monotone PTAS

To conclude, we need to show that calculating the payments given by (1) can be done in polynomial time. Notice that the integral in this expression has a constant number of breakpoints (the number of all possible allocations to agent i) thus calculating the payments can be done in constant time.

5 Truthful FPTAS-Mechanism for Any Fixed Number of Machines

We now show another truthful mechanism for any fixed number of machines. The mechanism uses a c-approximation algorithm as a black box, to generate a $c(1+\epsilon)$-approximation monotone algorithm. Using an FPTAS as the black box (for example, the FPTAS of Horowitz and Sahni [11]) outputs a monotone FPTAS. Adding a payments scheme as in (1), ensures truthful mechanism. The algorithm, *Monotone-Black-Box*, is shown in figure 3.

Input: a non decreasing sorted job sequence σ ,a bids vector $b = (b_1, b_2, .., b_m)$, a parameter ε and a black box, which is a polynomial time c-approximation.
Output: An allocation of jobs to the m machines that achieves a $c(1+\epsilon)$-approximation

1. Construct a new bid vector $d = (d_1, d_2, \ldots, d_m)$, in the following way:
   (a) round up each bid to the closest value of $(1+\epsilon)^i$
   (b) normalize the bids such that $d_1 = 1$
   (c) for each bid $d_j = (1+\epsilon)^i$, if $i > l+1$ where $l = \lceil \log_{1+\epsilon} cn \cdot \frac{p_1}{p_n} \rceil$ then set $d_j = (1+\epsilon)^{l+1}$
2. Enumerate over all possible vectors $d' = ((1+\epsilon)^{i_1}, (1+\epsilon)^{i_2}, \ldots, (1+\epsilon)^{i_m})$, where $i_j \in \{0, 1, \ldots, l+1\}$. For each vector:
   (a) apply the black box algorithm to d'
   (b) sort the output assignment such that the i-th fastest machine in d' will get the i-th largest amount of work
3. Test all the sorted assignments on d, and return the one with the minimal makespan. In case of a tie, choose the assignment with the lexicographically maximum schedule (i.e. allocating more to the faster machines)

Fig. 3. Monotone Black Box

Theorem 6. *Algorithm Monotone Black Box is a $c(1+\epsilon)$ approximation algorithm.*

Proof. The output assignment is a c-approximation for the vector d, since d is tested in the enumeration, and since sorting the assignment can only improve the makespan. As for the original bid vector b, and rounding the bids add a multiplicative factor of $1+\epsilon$ to the approximation ratio. Normalizing the vector has no effect, as well as trimming the largest bids, since any non zero assignment to a machine with a bid of at least $(1+\epsilon)^l$ cannot be a c-approximation, since

the load on that machine will be more than c times the load of assigning all the jobs to the fastest machine. □

Theorem 7. *If the black box in Algorithm Monotone Black Box is an FPTAS then the algorithm itself is also an FPTAS.*

Proof. By Theorem 6, the algorithm is a $(1+\epsilon)^2$ approximation. It remains to prove that the running time is polynomial in the input, including $\frac{1}{\epsilon}$. In each iteration of the enumeration, applying the black box, sorting the output assignment and testing it on the vector d can be completed in polynomial time, by the assumption that the black box is an FPTAS. The size of the enumeration is $O(l^m)$, where m is a constant and l is polynomial in the input. □

Theorem 8. *Algorithm Monotone Black Box is monotone.*

Proof. We prove that if a machine j raises its bid (lowers its speed) then the amount of work assigned to it cannot increase. We increment the bid in steps, such that in each step the power of $1+\epsilon$ that equals the rounded bid rises by one. We prove the claim for a single step, and therefore, the claim also holds for the entire increment.

We first assume that d_j is not the unique fastest machine (i.e., there is a machine $k \neq j$ such that $d_k = 1$). If $d_j \geq (1+\epsilon)^l$ then by the proof of Theorem 6, the assignment to machine j must be null, otherwise the approximation ratio is not achieved. Clearly, by raising the bid the assignment will remain null, and the claim holds. Therefore, we assume that the normalized rounded bid rises from d_j to $d_j(1+\epsilon)$, the assignment changes from W to W', and the amount of work allocated to machine j changes from w_j to $w'_j > w_j$.

We use $T(W,d)$ to denote the makespan of assignment W on bid vector d. Since the algorithm chooses the optimal assignment among a set that contains both W and W', we have that $T(W,d) \leq T(W',d)$ and $T(W',d') \leq T(W,d')$. Additionally, since the bids in d are smaller than the bids in d', we have that $T(W,d) \leq T(W,d')$ and $T(W',d) \leq T(W',d')$.

Suppose that machine j is the bottleneck in $T(W,d')$, meaning that the load on machine j is the highest. Since $w'_j > w_j$, we have $T(W,d') < T(W',d')$, as the load on machine j increases even more. This is a contradiction to $T(W',d') \leq T(W,d')$, and therefore machine j cannot be the bottleneck in $T(W,d')$. Therefore, if machine j is not the bottleneck in $T(W,d')$, we have that $T(W,d) = T(W,d')$. Since $T(W,d) \leq T(W',d) \leq T(W',d') \leq T(W,d')$, we actually have that $T(W,d) = T(W',d)$ and $T(W,d') = T(W',d')$. Therefore, we have a tie between W and W' for both d and d'. Since in each case the tie is broken differently, it must be that $W = W'$. Since the assignment is sorted (the faster machine is assigned more work), if a machine decreases its speed then the amount of work assigned to it (by the same assignment) cannot increase, which is a contradiction to $w'_j > w_j$.

If machine j is the unique fastest, then due to the normalization of the rounded bids and trimming of high bids, after it raises its bid by one step the new bid vector d' will be as follows: d_j remains 1, bids between $1+\epsilon$ and $(1+\epsilon)^l$

decrease by one step, and bids equal to $(1+\epsilon)^{l+1}$ can either decrease to $(1+\epsilon)^l$ or remain $(1+\epsilon)^{l+1}$.

Let $\hat{d}$ be the same bid vector as d', with all the bids of $(1+\epsilon)^{l+1}$ replaced with $(1+\epsilon)^l$. Since machines that bid $(1+\epsilon)^l$ or more must get a null assignment, then the optimal assignment (among all assignments that are considered by the algorithm) for $\hat{d}$ is the same as d'. The same assignment remains the optimum for vector $\hat{d}(1+\epsilon)$, where all bids are incremented by one step. The bid vector $\hat{d}(1+\epsilon)$ is exactly the bid vector d, with d_j replaced with $1+\epsilon$ (instead of 1). By the same argument from the case where machine j is not the unique fastest, the work assigned to machine j in $\hat{d}(1+\epsilon)$ is at most the same as the work assigned in d, and therefore the algorithm is monotone for the unique fastest machine as well. □

To conclude, we claim that the payments for each agent can be calculated in polynomial time, since the number of breakpoints in the integral is bounded by the number of possible allocations considered by the algorithm, which is polynomial in the input size (including $\frac{1}{\epsilon}$) .

6 Conclusions and Open Problems

We have shown a *deterministic* constant-approximation truthful mechanism for assigning jobs on uniformly related machines, and an FPTAS truthful mechanism for the special case where the number of machines is fixed. The main open question left is whether a truthful PTAS mechanism exists in the case of an arbitrary number of machines.

References

1. N. Alon, Y. Azar, G. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
2. P. Ambrosio and E. Auletta. Deterministic monotone algorithms for scheduling on related machines. In *Proceeding of WAOA*, 2004.
3. A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 482. IEEE Computer Society, 2001.
4. Aaron Archer. *Mechanisms for Discrete Optimization with Rational Agents*. PhD thesis, Cornell University, 2004.
5. Vincenzo Auletta, Roberto De Prisco, Paolo Penna, and Pino Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *Proceedings of 21th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *Lecture Notes in Computer Science*, pages 608–619. Springer Verlag, 2004.
6. E. Clarke. Multipart pricing of public goods. *Public Choice*, 1971.
7. Leah Epstein and Jiri Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *Proceedings of the 7th Annual European Symposium on Algorithms*, pages 151–162. Springer-Verlag, 1999.

8. Teofilo Gonzalez, Oscar H. Ibarra, and Sartaj Sahni. Bounds for LPT schedules on uniform processors. *SIAM Journal on Computing*, 6(1):155–166, March 1977.
9. T. Groves. Incentives in teams. *Econemetrica*, 1973.
10. D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.
11. E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling non-identical processors. *Journal of the Association for Computing Machinery*, 23:317–327, 1976.
12. N. Nisan and A. Ronen. Algorithmic mechanism design. In *31st ACM Symp. on Theory of Computing*, pages 129–140, 1999.
13. Noam Nisan and Amir Ronen. Computationally feasible vcg mechanisms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 242–252. ACM Press, 2000.
14. Christos Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 749–753. ACM Press, 2001.
15. Eva Tardos. Network games. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2004.
16. W. Vickery. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 1961.

Counting in the Two Variable Guarded Logic with Transitivity

Lidia Tendera

Institute of Mathematics and Informatics,
Opole University, Opole, Poland
tendera@math.uni.opole.pl

Abstract. We show that the extension of the two-variable guarded fragment with transitive guards (GF+TG) by functionality statements is undecidable. This gives immediately undecidability of the extension of GF+TG by counting quantifiers. The result is optimal, since both the three-variable fragment of the guarded fragment with counting quantifiers and the two-variable guarded fragment with transitivity are undecidable.

We also show that the extension of GF+TG with functionality, where functional predicate letters appear in guards only, is decidable and of the same complexity as GF+TG. This fragment captures many expressive modal and description logics.

Keywords: guarded fragment, counting, transitivity, decision problem, computational complexity.

1 Introduction

The guarded fragment or GF of first-order logic, introduced by H. Andréka, J. van Benthem and I. Németi [1], has appeared to be a successful attempt to transfer good properties of modal and temporal logics to a naturally defined fragment of predicate logic. These properties include among others decidability, the finite model property, finite axionatization, Craig interpolation and Beth definability.

In the guarded fragment formulas are built as in first-order logic with the only restriction that quantifiers are appropriately relativized by atoms, i.e. neither the pattern of alternations of quantifiers nor the number of variables is restricted. Andréka et al. showed that modal logic can be embedded in GF and they argued convincingly that GF inherits the nice properties of modal logic. The nice behavior of GF was confirmed by Grädel [6] who proved that the satisfiability problem for GF is complete for double exponential time and complete for exponential time, when the number of variables is bounded.

GF was later generalized by van Benthem [18] to the *loosely guarded fragment*, LGF, and by Grädel [7] to the *clique guarded fragment*, CGF, where all quantifiers are relativized by more general formulas, preserving the idea of quan-

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 83–96, 2005.

tification only over elements that are close together in the model. Most of the properties of GF generalize to LGF and CGF.

The above mentioned positive results led directly to the question, if the guarded fragment keeps the good properties when extended by other operators, similarly to modal logic that remains decidable and of fairly low complexity under addition of a variety of operators and features, such as counting modalities, transitive closure modalities and conditions on the accessibility relation.

In fact, E. Grädel and I. Walukiewicz [9] proved that the extension of GF with fixed point operators that captures the modal μ-calculus is a decidable logic and of the same complexity as pure GF.

On the other hand it has been observed in [6] that very modest extensions of GF lead to undecidability: three-variable GF with transitive predicates and three-variable GF with counting quantifiers are both undecidable extensions of GF. The latter result is optimal with respect to the number of variables since the whole two-variable first-order logic with counting quantifiers is decidable [8], [13]. H. Ganzinger, C. Meyer and M. Veanes improved the first result showing that even the two-variable GF with transitive predicates and without equality is undecidable.

The paper by Ganzinger et al. [5] presents also a decidability result for the *monadic* two-variable guarded fragment with transitivity, namely when binary predicates can appear in guards only. This result was sharpened by W. Szwast and L. Tendera [16], who proved that the *guarded fragment with transitive guards*, GF+TG, i.e. of the fragment where transitive predicate letters appear only in guards whereas remaining predicates and the equality symbol may appear everywhere, is decidable in double exponential time. Surprisingly, the complexity stays the same even for the monadic two-variable fragment as it was proved by E. Kieroński in [12]. It is worth mentioning that in contrast to the model-theoretic construction used in [16] (full version in [17]), a resolution based decision procedure for GF+TG without equality has been recently presented by Y. Kazakov and H. de Nivelle in [11].

In this paper we investigate the possibility of extending GF with both transitive predicates and counting. The expressive power for the resulting logic would be well suited to capture many powerful description logics that are used to express properties of aggregated objects [15] and have applications e.g. in the area of conceptual data models [3], [4] and query optimization [10].

Since, as it was mentioned above, the three-variable GF with counting quantifiers and GF^2 with transitivity are both undecidable, we restrict our attention to the two-variable restriction of the guarded fragment with transitive guards, GF^2+TG, and we prove first that the extension of GF^2+TG by functionality statements is undecidable. This gives undecidability of the extension of GF^2+TG by *counting quantifiers*, since functionality of a predicate R can be easily expressed by a guarded formula of the form $\forall x(x = x) \to \exists^{=1} y R(x, y)$.

As a second result we show that if we restrict our attention to the guarded fragment with *transitive* and *functional guards*, i.e. where both transitive and functional predicate letters can appear in guards only, we get a decidable logic.

Moreover, we show that the complexity of the fragment is the same as for simple GF+TG, i.e. functional guards do not increase the complexity of the reasoning. This result can also be seen as an additional confirmation of the status of the guarded fragment as the best classical counterpart of modal and temporal logics: if we use functionality only there where it could appear in a modal formula, we stay in a decidable fragment of the same computational complexity.

The remainder of the paper is organized as follows. In the next section we introduce the guarded fragments in detail. In section 3 we show undecidability of GF+TG with functionality by reducing a domino tiling problem. And in the last section, basing on the ideas from [17], we show that GF+TG with functional guards is decidable in double-exponential time.

2 Guarded Fragments

The *guarded fragment*, GF, of first-order logic with no function symbols is defined as the least set of formulas such that

1. every atomic formula belongs to GF,
2. GF is closed under logical connectives $\neg, \vee, \wedge, \rightarrow$,
3. if $\mathbf{x}, \mathbf{y}$ are tuples of variables, $\alpha(\mathbf{x}, \mathbf{y})$ is an atomic formula containing all the variables of $\{\mathbf{x}, \mathbf{y}\}$ and $\psi(\mathbf{x}, \mathbf{y})$ is a formula of GF with free variables contained in $\{\mathbf{x}, \mathbf{y}\}$, then the formulas

$$\forall \mathbf{y}(\alpha(\mathbf{x}, \mathbf{y}) \rightarrow \psi(\mathbf{x}, \mathbf{y})) \quad \text{and} \quad \exists \mathbf{y}(\alpha(\mathbf{x}, \mathbf{y}) \wedge \psi(\mathbf{x}, \mathbf{y}))$$

belong to GF.

The atom $\alpha(\mathbf{x}, \mathbf{y})$ in the above formulas is called the *guard* of the quantifier. A guard that is a P atom, where P is a predicate letter from the signature, is called a P-guard.

We will use $(\forall \mathbf{y}.\alpha(\mathbf{x}, \mathbf{y}))\psi(\mathbf{x}, \mathbf{y})$, and $(\exists \mathbf{y}.\alpha(\mathbf{x}, \mathbf{y}))\psi(\mathbf{x}, \mathbf{y})$ as alternative notations for $\forall \mathbf{y}(\alpha(\mathbf{x}, \mathbf{y}) \rightarrow \psi(\mathbf{x}, \mathbf{y}))$, and $\exists \mathbf{y}(\alpha(\mathbf{x}, \mathbf{y}) \wedge \psi(\mathbf{x}, \mathbf{y}))$, respectively.

We denote by FO^k the class of first-order sentences with k variables over a relational signature. By GF^k we denote the fragment $\mathrm{GF} \cap \mathrm{FO}^k$.

By a *transitivity statement* we mean an assertion $Trans[P]$, saying that the binary relation P is a transitive relation. By a *functionality statement* we mean an assertion $Functional[P]$, saying that the binary relation P is a graph of a partial function, i.e. a statement of the form $\forall x \exists^{\leq 1} y Pxy$. A binary predicate letter P is called *transitive* (*functional*) if $Trans[P]$ ($Functional[P]$, respectively) holds.

By GF+TG we denote the guarded fragment with *transitive guards* that is the restriction of GF with transitivity statements where all transitive predicate letters appear in guards only and where the equality symbol can appear everywhere. By GF^2+TG we denote the restriction of GF+TG to two variables.

In this paper we are particularly interested in the following logics

– GF^2+TG+F, the extension of GF^2+TG with functionality statements. Note that in this fragment we can declare both the binary relation and its inverse to be functional, see e.g. $\forall x, y\ Pxy \rightarrow Ryx \wedge Functional[R]$.

- GF^2+TG+FG, the restriction of GF^2+TG+F, where functional predicate letters can appear in guards only.
  Here it should be decided if inverses of functional predicates can be declared to be functional as well and whether a relation can be functional and transitive at the same time.
- GF^2+TG+Counting, the extension of GF^2+TG with *counting*, i.e. the fragment where formulas can be built using also *counting quantifiers* of the form $\exists^{\leq m}$, $\exists^{\geq m}$, $m \geq 0$ (in the standard guarded way).

It is clear that

$$\mathrm{GF}^2\text{+TG} \subseteq \mathrm{GF}^2\text{+TG+FG} \subseteq \mathrm{GF}^2\text{+TG+F} \subseteq \mathrm{GF}^2\text{+TG+Counting}.$$

In this paper we show first that GF^2+TG+F is undecidable that immediately gives undecidability of GF^2+TG+Counting. In the next section we show that GF^2+TG+FG is decidable in deterministic double-exponential time.

3 $\mathbf{GF^2}$+TG+F

In this section we prove that the satisfiability problem for the two-variable guarded fragment with transitive guards and functionality statements is undecidable. Note that since GF+TG is decidable, this clearly shows that GF^2+TG+F is more expressive than GF+TG.

Theorem 1. *The satisfiability problem for GF^2+TG+F is undecidable.*

Proof. We modify the undecidability proof of GF^2 with transitivity from [5], where a reduction from the halting problem for Minsky (two-counter) machines is shown. In fact, here the same reduction can be applied but we need to characterize two-dimensional grids with origin using transitive predicates in guards only.

To characterize grids with , we write a sentence `Grid` in GF^2+TG+F with four transitive relations W_0, W_1, B_0, B_1, four functional arc relations $\uparrow^0$, $\uparrow^1$, $\xrightarrow{0}$, $\xrightarrow{1}$, equality and additional unary relations. There is a unary predicate $Node$. In any model of `Grid` we are interested in elements in $Node$, such elements are called *nodes*. It is worth noticing that equality is not an issue here. Instead of using equality we could add an additional functional predicate E and insure that it behaves as equality on nodes in every model of `Grid`, saying: $\forall x Node(x) \Rightarrow E(x,x) \wedge Functional[E]$.

The formula `Grid` is a conjunction of the formulas (1)-(12).

It is easy to express the following properties using additional unary predicate letters *Bottom*, *Left*, *Origo*, $White_l$ and $Black_l$, where $l = \{0, 1\}$.

"The set of nodes is non-empty and all nodes are either black or white and have either 0 or 1 as a label." (1)

"Bottom nodes have no vertical predecessors and all horizontal successors of bottom nodes are bottom nodes, similarly for Left nodes." (2)

The origin element has the following properties.

$$\begin{aligned}&\exists x\,(Origo(x)) \wedge \forall x\,(Origo(x) \Rightarrow White_0(x)) \wedge \\ &\forall x\,(Origo(x) \Rightarrow (Left(x) \wedge Bottom(x))) \wedge \\ &\forall x\,(Bottom(x) \Rightarrow (Left(x) \Rightarrow Origo(x)))\end{aligned} \tag{3}$$

The colors and labels alternate in both horizontal and vertical directions as follows. For $l \in \{0,1\}$, let $\bar{l} = 0$ if $l = 1$ and $\bar{l} = 1$, if $l = 0$:

$$\forall xy\,(x \xrightarrow{l} y \Rightarrow ((White_l(x) \wedge Black_{\bar{l}}(y)) \vee (Black_l(x) \wedge White_{\bar{l}}(y)))) \tag{4}$$

$$\forall xy\,(x \uparrow^{l} y \Rightarrow ((White_l(x) \wedge Black_l(y)) \vee (Black_{\bar{l}}(x) \wedge White_{\bar{l}}(y)))) \tag{5}$$

Every node with a certain label has the following arcs connected to it:

$$\begin{aligned}\forall x\,(White_0(x) \Rightarrow (&\exists y\,(W_0(x,y) \wedge x \xrightarrow{0} y) \wedge \exists y\,(W_0(y,x) \wedge x \xrightarrow{0} y) \wedge \\ &\exists y\,(W_0(x,y) \wedge x \uparrow^0 y) \wedge \exists y\,(W_0(y,x) \wedge x \uparrow^0 y) \wedge \\ &(Bottom(x) \vee (\exists y\,(W_1(x,y) \wedge y \uparrow^1 x) \wedge \exists y\,(W_1(y,x) \wedge y \uparrow^1 x))) \wedge \\ &(Left(x) \vee (\exists y\,(W_1(x,y) \wedge y \xrightarrow{1} x) \wedge \exists y\,(W_1(y,x) \wedge y \xrightarrow{1} x)))))\end{aligned} \tag{6}$$

$$\begin{aligned}\forall x\,(White_1(x) \Rightarrow (&\exists y\,(W_1(x,y) \wedge x \xrightarrow{1} y) \wedge \exists y\,(W_1(y,x) \wedge x \xrightarrow{1} y) \wedge \\ &\exists y\,(W_1(x,y) \wedge x \uparrow^1 y) \wedge \exists y\,(W_1(y,x) \wedge x \uparrow^1 y) \wedge \\ &\exists y\,(W_0(x,y) \wedge y \uparrow^0 x) \wedge \exists y\,(W_0(y,x) \wedge y \uparrow^0 x) \wedge \\ &\exists y\,(W_0(x,y) \wedge y \xrightarrow{0} x) \wedge \exists y\,(W_0(y,x) \wedge y \xrightarrow{0} x)))\end{aligned} \tag{7}$$

$$\begin{aligned}\forall x\,(\mathrm{Black}_0(x) \Rightarrow (&\exists y\,(B_0(x,y) \wedge x \xrightarrow{0} y) \wedge \exists y\,(B_0(y,x) \wedge x \xrightarrow{0} y) \wedge \\ &\exists y\,(B_0(x,y) \wedge x \uparrow^1 y) \wedge \exists y\,(B_0(y,x) \wedge x \uparrow^1 y) \wedge \\ &\exists y\,(B_1(x,y) \wedge y \uparrow^0 x) \wedge \exists y\,(B_1(y,x) \wedge y \uparrow^0 x) \wedge \\ &(Left(x) \vee (\exists y\,(B_1(x,y) \wedge y \xrightarrow{1} x) \wedge \exists y\,(B_1(y,x) \wedge y \xrightarrow{1} x))))\end{aligned} \tag{8}$$

$$\begin{aligned}\forall x\,(\mathrm{Black}_1(x) \Rightarrow (&\exists y\,(B_1(x,y) \wedge x \xrightarrow{1} y) \wedge \exists y\,(B_1(y,x) \wedge x \xrightarrow{1} y) \wedge \\ &\exists y\,(B_1(x,y) \wedge x \uparrow^0 y) \wedge \exists y\,(B_1(y,x) \wedge x \uparrow^0 y) \wedge \\ &\exists y\,(B_0(x,y) \wedge y \xrightarrow{0} x) \wedge \exists y\,(B_0(x,y) \wedge y \xrightarrow{0} x) \wedge \\ &(Bottom(x) \vee (\exists y\,(B_0(x,y) \wedge y \uparrow^1 x) \wedge \exists y\,(B_0(y,x) \wedge y \uparrow^1 x))))\end{aligned} \tag{9}$$

For each of the transitive relations we have the following formulas.

$$\begin{aligned}\forall xy\,(W_l(x,y) \Rightarrow (&x = y \vee x \xrightarrow{l} y \vee y \xrightarrow{l} x \vee x \uparrow^l y \vee y \uparrow^l x \vee \\ &(White_0(x) \wedge White_1(y)) \vee (White_1(x) \wedge White_0(y)) \vee \\ &(Black_0(x) \wedge Black_1(y)) \vee (Black_1(x) \wedge Black_0(y))))\end{aligned} \tag{10}$$

$$\forall xy\,(B_l(x,y) \Rightarrow (x = y \vee x \xrightarrow{l} y \vee y \xrightarrow{l} x \vee x \uparrow^{\bar{l}} y \vee y \uparrow^{\bar{l}} x \vee (White_0(x) \wedge White_1(y)) \vee (White_1(x) \wedge White_0(y)) \vee (Black_0(x) \wedge Black_1(y)) \vee (Black_1(x) \wedge Black_0(y)))) \tag{11}$$

For each unary predicate P and each binary predicate R we have the formulas:

$$\forall x\,(P(x) \Rightarrow Node(x)) \quad \text{and } \forall xy\,(R(x,y) \Rightarrow (Node(x) \wedge Node(y))) \tag{12}$$

Formulas (1)-(5) and (10)-(12) are taken from [5]. The other conjuncts are added to yield the same models for `Grid` as in [5]. To see that our formula `Grid` properly characterizes grids we need to show the following two properties:

Claim. The arc relations are functional in both arguments in all models of `Grid`.

Proof. Consider the relation $\xrightarrow{0}$ and assume that $b \xrightarrow{0} a$ and $c \xrightarrow{0} a$, for some nodes a, b and c. By (4), a has label 1. Assume a is white. By (4), b and c are black with label 0. By (6) and functionality of $\xrightarrow{0}$, we have $W_0(b,a)$ and $W_0(a,c)$. By transitivity of W_0 we have $W_0(b,c)$. By (10), $b = c$. In case a is black, we apply (9) to get $B_0(b,a)$ and $B_0(a,c)$ and we get $b = c$ by transitivity of B_0. The proof for other arc relations is symmetrical. □

We say that the arc relations *induce a diagonal* if whenever $a \to b \uparrow c$ and $a \uparrow b' \to c'$, then $c = c'$, where $\to$ is either $\xrightarrow{0}$ or $\xrightarrow{1}$ and $\uparrow$ is either $\uparrow^0$ or $\uparrow^1$.

Claim. The arc relations induce a diagonal in all models of `Grid`.

Proof. Consider a model of `Grid` and let a be a white node with label 0. Then we have by functionality of the arc relations, unique nodes b, b', c, c' such that $a \uparrow^0 b \xrightarrow{0} c$ and $a \xrightarrow{0} b' \uparrow^0 c'$. By (4) and (5) c and c' are white nodes with label 1. By (6), we have $W_0(a,b')$, $W_0(b',a)$, $W_0(b,a)$ and $W_0(a,b)$. By (7), we have $W_0(b,c)$, $W_0(c,b)$, $W_0(b',c)$ and $W_0(c,b')$. By transitivity of W_0, $W_0(c,c')$. By (10), $c = c'$. The proof for other arc relations is symmetrical. □

Inspecting the proof of the above Theorem we get the following corollary

Corollary 1. *GF^2+TG+F is undecidable, even when no predicate letter is declared at the same time transitive and functional.*

4 $\mathbf{GF^2}$+TG+FG

In this section we prove that the guarded fragment with transitive and functional guards is decidable in double exponential time.

Throughout the section we assume that we have a signature σ that contains unary and binary predicate letters. We denote by *Funct* (*Trans*) the subset of σ with all functional (respectively, transitive) predicate letters and by *FunctInv* the subset of *Funct* with functional inverses. We assume that we can declare the

same predicate letter to be both transitive and functional, i.e. $Funct \cap Trans$ can be nonempty.

The decidability proof follows the lines of the proof for GF+TG from [17]. In particular, we use the same normal form for GF^2+TG+FG. In the definition of *ramified* models we describe good properties of models of sentences in normal form that are later used to enumerate witnesses of the existence of a model (called *carpets of flowers*).

4.1 Basic Definitions

Let $\mathbf{x} = (x_1, \ldots, x_l)$ be a sequence of variables. An *l-type* $t(\mathbf{x})$ is a maximal consistent set of atomic and negated atomic formulas over σ in the variables of $\mathbf{x}$. A type t is often identified with the conjunction of formulas in t. In this paper we need 1- and 2-types that, if not stated otherwise, will be types of the variable x and of the variables x and y, respectively. A 2-type t is *proper* if t contains the formula $x \neq y$. If $t(\mathbf{x})$ is an l-type and $\mathbf{x}' \subseteq \mathbf{x}$ then we denote by $t(\mathbf{x}) \restriction \mathbf{x}'$ the unique type $t(\mathbf{x}')$ in the variables of $\mathbf{x}'$ included in $t(\mathbf{x})$.

Let $\psi(\mathbf{x})$ be a quantifier-free formula in the variables of $\mathbf{x}$. We say that a type t *satisfies* ψ, $t \models \psi$, if ψ is true under the truth assignment that assigns true to an atomic formula precisely when it is a member of t.

In this paper, σ-structures are denoted by Gothic capital letters and their universes by corresponding Latin capitals. If $\mathfrak{A}$ is a σ-structure with the universe A, and if $\mathbf{a}$ is an l-tuple of elements of A, then we denote by $tp^{\mathfrak{A}}(\mathbf{a})$ the unique l-type $t(\mathbf{x})$ realized by $\mathbf{a}$ in $\mathfrak{A}$. If $B \subseteq A$ then $\mathfrak{A} \restriction B$ denotes the substructure of $\mathfrak{A}$ induced on B. Usually we do not distinguish predicate letters and their interpretations in a structure. However, when it is not clear from the context, we use $P^{\mathfrak{A}}$ to denote the interpretation of the predicate letter P in the structure $\mathfrak{A}$.

If $\mathfrak{A}$ and $\mathfrak{B}$ are σ-structures, $a \in A$ and $b \in B$ then we write $(\mathfrak{A}, a) \cong (\mathfrak{B}, b)$ to say that there is an isomorphism f of the structures $\mathfrak{A}$ and $\mathfrak{B}$ such that $f(a) = b$.

Let $\mathfrak{A}$ be a σ-structure, B a binary predicate letter in σ and $\mathfrak{C}$ a substructure of $\mathfrak{A}$. We say that $\mathfrak{C}$ is a *B-clique* if for every $a, b \in C$ we have $\mathfrak{A} \models B(a, b)$. Let $a \in A$. If $B \in Trans$, $B^{\mathfrak{A}}$ is transitive and $\mathfrak{A} \models B(a, a)$ then we denote by $[a]^{\mathfrak{A}}_B$ the maximal B-clique containing a; otherwise $[a]^{\mathfrak{A}}_B$ is the one-element structure $\mathfrak{A} \restriction \{a\}$.

4.2 Normal Form

Definition 1. *A GF^2+TG+FG-sentence is in* normal form *if it is a conjunction of sentences of the following forms:*

$$\exists x\, (\alpha(x) \wedge \psi(x)), \tag{1}$$

$$\forall x\, (\alpha(x) \rightarrow \exists y\, (\beta(x, y) \wedge \psi(x, y))), \tag{2}$$

$$\forall x \forall y\, (\alpha(x, y) \rightarrow \psi(x, y)), \tag{3}$$

where $y \neq x$, α, β are atomic formulas, ψ is quantifier-free and contains neither a transitive nor a functional predicate letter.

The following lemma can be proved in the same way as in [17].

Lemma 1. *With every $GF^2+TG+FG$-sentence Γ of length n over a signature τ one can effectively associate a set $\mathbf{\Delta}$ of $GF^2+TG+FG$-sentences in normal form over an extended signature σ, $\mathbf{\Delta} = \{\Delta_1, \ldots, \Delta_d\}$, such that*

1. *Γ is satisfiable if and only if $\bigvee_{i \leq d} \Delta_i$ is satisfiable,*
2. *$d \leq O(2^n)$, $card(\sigma) \leq n$ and for every $i \leq d$, $|\Delta_i| = O(n \log n)$,*
3. *$\mathbf{\Delta}$ can be computed deterministically in exponential time and every sentence Δ_i can be computed in time polynomial with respect to n.*

For future reference in a sentence in normal form we distinguish the following kinds of conjuncts of the form (2):

$$(\forall x.\alpha(x))\,(\exists y.Fxy)\,\psi(x,y), \quad \text{where } F \in \mathit{Funct}, \tag{2a}$$
$$(\forall x.\alpha(x))\,(\exists y.Fyx)\,\psi(x,y), \quad \text{where } F \in \mathit{FunctInv}, \tag{2b}$$
$$(\forall x.\alpha(x))\,(\exists y.Fyx)\,\psi(x,y), \quad \text{where } F \in \mathit{Funct} \setminus \mathit{FunctInv}, \tag{2c}$$
$$(\forall x.\alpha(x))\,(\exists y.Txy)\,\psi(x,y), \quad \text{where } T \in \mathit{Trans}, \tag{2d}$$
$$(\forall x.\alpha(x))\,(\exists y.Tyx)\,\psi(x,y), \quad \text{where } T \in \mathit{Trans}, \tag{2e}$$
$$(\forall x.\alpha(x))\,(\exists y.Pxy)\,\psi(x,y) \quad \text{or} \quad (\forall x.\alpha(x))\,(\exists y.Pyx)\,\psi(x,y), \quad \text{where } P \notin (\mathit{Funct} \cup \mathit{Trans}). \tag{2f}$$

4.3 Ramified Models

In [16] the notion of *ramified models* was introduced. We show that GF^2+TG+FG-sentences have appropriate ramified models as well. In a *ramified* structure cliques are of exponential size (with respect to the cardinality of the signature), and distinct transitive and/or functional predicates have only disjoint paths.

First, we distinguish some special kinds of 2-types that will be useful later.

Definition 2. *Let $t(x,y)$ be a proper 2-type, B a binary predicate letter in σ, $F \in \mathit{Funct}$ and $T \in \mathit{Trans}$.*

1. *$t(x,y)$ is T-*transitive *if $t \models T(x,y) \vee T(y,x)$. $t(x,y)$ is* transitive *if t is T-transitive, for some $T \in \mathit{Trans}$.*
2. *$t(x,y)$ is F-*functional *if $t \models F(x,y)$. $t(x,y)$ is* functional*, if t is F-functional for some $F \in \mathit{Funct}$.*
3. *$t(x,y)$ is* plain *if t is neither transitive nor functional.*
4. *$t(x,y)$ is* singular *if there exists exactly one predicate letter $B \in \mathit{Funct} \cup \mathit{Trans}$ such that $t \models B(x,y) \vee B(y,x)$.*
   Note that a singular type is either transitive or functional or can be both, if $B \in \mathit{Funct} \cap \mathit{Trans}$.
5. *$t(x,y)$ is B-*symmetric *if $t \models B(x,y) \wedge B(y,x)$ and t is B-*oriented *if $t \models B(x,y) \vee B(y,x)$ but t is not B-symmetric. $t(x,y)$ is* symmetric *if t is singular and B-symmetric, for some $B \in \mathit{Funct} \cup \mathit{Trans}$. $t(x,y)$ is* oriented *if t is singular and B-oriented, for some $B \in \mathit{Funct} \cup \mathit{Trans}$.*

We also need the following operations to construct new types from given ones.

Definition 3. *1. Let $t = t(x, y)$ be a 2-type. By $\bar{t}$ we denote the type $t(y, x)$ obtained from t by swapping the variables x and y.*

2. Let $v(x), w(y)$ be 1-types. The negative link of v *and w, denoted by $\overline{v, w}$, is the unique proper 2-type containing $v(x)$, $w(y)$ and no positive atomic two-variable formula.*

*3. Let t be a proper 2-type over σ and B be a binary predicate letter in σ. The B-*slice of *t, denoted by $\overleftrightarrow{t, B}$, is the unique proper 2-type obtained from t by replacing in t every atomic formula of any of the forms $T(x, y)$, $T(y, x)$, $F(x, y)$, $F(y, x)$, with $T \neq B$, $F \neq B$, for any $T \in Trans$, $F \in Funct$, by its negation.*

Let γ be a σ-sentence of the form $\forall x\, \alpha(x) \to \exists y\, \phi(x, y)$, $\mathfrak{A}$ be a σ-structure and $a \in A$. We say that an element $b \in A$ is a *witness of* γ for a in $\mathfrak{A}$ if $\mathfrak{A} \models \alpha(a) \to \phi(a, b)$. Note that if $\mathfrak{A} \not\models \alpha(a)$ then any element $b \in A$ is a witness of γ for a in $\mathfrak{A}$. Similarly, we say that $a \in A$ is a *witness of* γ of the form $\exists x \phi(x)$ in $\mathfrak{A}$ if $\mathfrak{A} \models \phi(a)$.

We also need the notion of petals from [17] that correspond to cliques of transitive predicates.

Definition 4. *Let Δ be a* GF^2+TG+FG*-sentence in normal form, $\mathfrak{A}$ be a model for Δ, $p \in A$ and $T \in Trans$. We say that a σ-structure $\mathfrak{D}$ is a* T-petal of $[p]^{\mathfrak{A}}_T$ *if there exists a function $g : D \mapsto [p]^{\mathfrak{A}}_T$ that preserves 1-types and the following conditions hold:*

(p1) *$card(D) \leq 3 \cdot |\Delta| \cdot 2^{card(\sigma)}$,*
(p2) *$p \in Im(g)$,*
(p3) *every proper 2-type realized in $\mathfrak{D}$ is a T-slice of some type realized in $[p]^{\mathfrak{A}}_T$,*
(p4) *for every $a \in D$, for every conjunct γ of Δ of the form (2) $\forall x(\alpha(x) \to \exists y(\beta(x, y) \wedge \psi(x, y)))$, where β is a T-guard, if there exists a witness of γ for $g(a)$ in $[p]^{\mathfrak{A}}_T$ (i.e. the type between $g(a)$ and the witness is T-symmetric), then there exists a witness of γ for a in $\mathfrak{D}$.*

In case we consider a binary predicate letter $B \in Funct \cap Trans$, we know that B-cliques contain exactly 1 element. In case $B \notin Funct$, we know from [17] that petals exist. So we have the following proposition.

Proposition 1. *Let Δ be a* GF^2+TG+FG*-sentence in normal form and let $\mathfrak{A}$ be a model for Δ. For every $T \in Trans$, for every $p \in A$, there exists a T-petal of $[p]^{\mathfrak{A}}_T$.*

Below we adapt the notion of *r-ramified* models for GF^2+TG+FG-sentences in normal form. In such models the cardinality of all T-cliques is bounded by the constant r and paths of elements corresponding to distinct transitive predicates do not have common edges. In addition, elements belonging to two distinct cliques containing a common element are connected by negative types.

Definition 5. *Let Δ be a* GF^2+TG+FG*-sentence in normal form over σ, let $\mathfrak{R}$ be a model for Δ and let r be a positive integer. We say that $\mathfrak{R}$ is an* r-ramified model *for Δ if the following conditions hold:*

1. *for every $T \in Trans$, for every $a \in R$, the cardinality of $[a]^{\mathfrak{R}}_T$ is bounded by r,*
2. *for every $a, b \in R$ such that $a \neq b$, $tp^{\mathfrak{R}}(a,b)$ is either singular or plain,*
3. *for every $T, T' \in Trans$ such that $T \neq T'$, for every $a, b, c \in R$, $b \neq a, c \neq a$, if $b \in [a]^{\mathfrak{R}}_T$ and $c \in [a]^{\mathfrak{R}}_{T'}$ then $tp^{\mathfrak{R}}(b,c) = \overline{tp^{\mathfrak{R}}(b), tp^{\mathfrak{R}}(c)}$.*

We have the following theorem. The proof has been omitted due to space limits.

Theorem 2. *Every satisfiable* GF^2+TG+FG*-sentence Δ in normal form has a $3 \cdot |\Delta| \cdot 2^{card(\sigma)}$-ramified model.*

4.4 Syntactic Witness for Satisfiability

In this section we fix a GF^2+TG+FG-sentence Δ in normal form and we aim at defining a syntactic witness for satisfiability of Δ.

We first define 1- and 2-types that may appear in models of Δ.

Definition 6. *A 1-type $s(x)$ is* Δ-acceptable *if*

a. *for every conjunct of Δ of the form* (2) $\forall x\,(\alpha(x) \to \exists y\,(x = y \wedge \psi(x,y)))$, *we have $s \models \alpha(x) \to \psi(x,x)$, and*
b. *for every conjunct of Δ of the form* (3) $\forall x \forall y\,(\alpha(x,y) \to \psi(x,y))$, *we have $s \models (\alpha(x,x) \to \psi(x,x))$.*

A 2-type $t(x,y)$ is Δ-acceptable *if $t{\restriction}x$ and $t{\restriction}y$ are Δ-acceptable and for every conjunct of Δ of the form* (3) $\forall x \forall y(\alpha(x,y) \to \psi(x,y))$, *we have $t \models (\alpha(x,y) \to \psi(x,y)) \wedge (\alpha(y,x) \to \psi(y,x))$.*

Now, we define Δ-flowers as fragments of ramified models of Δ visible from a single point of the model. Each flower has a pistil corresponding to a fixed element of the model, the pistil has a color (defined by a 1-type). Each flower has petals corresponding to cliques of transitive predicates, leaves corresponding to colors of elements connected by transitive oriented edges with the pistil and stamens (defined by 2-types) corresponding to functional witnesses for the pistil.

Definition 7. *Let r be a positive integer. An* r-flower *$\mathcal{F}$ is a tuple*

$$\mathcal{F} = \langle p,\ tp(p),\ \{\mathfrak{D}_T\}_{T \in Trans},\ \{In_T\}_{T \in Trans},\ \{t_F\}_{F \in Funct} \rangle\ ,$$

where $tp(p)$ is a 1-type, each $\mathfrak{D}_T$ is a σ-structure, each In_T is a set of 1-types and each t_F is either a 1-type or a singular F-functional 2-type such that the following conditions hold

1. *for every* $T, T' \in Trans$, $T \neq T'$, *we have* $D_T \cap D_{T'} = \{p\}$.
2. *for every* $T \in Trans$, $\mathfrak{D}_T = [p]_T^{\mathfrak{D}_T}$, $tp^{\mathfrak{D}_T}(p) = tp(p)$, $card(D_T) \leq r$ *and every proper 2-type realized in* $\mathfrak{D}_T$ *is singular* T*-transitive. Moreover, if* T *is functional, then* $card(D_T) = 1$.
3. *for every* $F \in Funct$, $t_F \restriction x = tp(p)$ *and* $t \models (F(x,x) \vee F(x,y)) \wedge (F(x,x) \rightarrow x = y)$. *If, additionally,* $F \in Trans$ *then either* t_F *is an oriented 2-type and* $In_F = \emptyset$ *or* t_F *is a 1-type and* $card(In_F) \leq 1$.

An r-flower $\mathcal{F}$ *is called a* Δ-flower *if additionally,* $r = 3 \cdot |\Delta| \cdot 2^{card(\sigma)}$ *and*

4. *for every* $T \in Trans$,
   for every $a, b \in D_T$, $tp^{\mathfrak{D}_T}(a,b)$ *is* Δ*-acceptable,*
   for every $s \in In_T$ *there is a* Δ*-acceptable, singular transitive 2-type* t *such that* $t \models tp(p)(x) \wedge s(y) \wedge T(y,x) \wedge \neg T(x,y)$ *(*t *is proper,* T*-oriented).*
5. *for every* $F \in Funct$, t_F *is* Δ*-acceptable,*
6. *for every conjunct* γ *of the form* (2a)*:* $(\forall x.\alpha(x))\,(\exists y.Fxy)\,\psi(x,y)$, $t_F(x,y) \models \alpha(x) \rightarrow Fxy \wedge \psi(x,y)$.

The element p *is called* pistil, *the structures* $\mathfrak{D}_T$ *are* T-petals, *the sets* In_T *are called* T-leaves *and the types* t_F *are* F-stamens *of the* r*-flower* $\mathcal{F}$.

Let γ be a sentence of the form (2) and $\mathcal{F}$ a flower. If either for some of the types t_F in $\mathcal{F}$ we have $t_F(x,y) \models \alpha(x) \rightarrow \beta(x,y) \wedge \psi(x,y)$, or there is a witness of γ for p in some structure $\mathfrak{D}_T$, then we say that *there is a witness of* γ *for* p *in* $\mathcal{F}$.

The flowers in a model are connected, so we need the following definition.

Definition 8. *Let* $\mathcal{F}$ *and* $\mathcal{W}$ *be* r*-flowers and* t *be either a singular or plain 2-type. We say that* $\mathcal{W}$ *is* connectable to $\mathcal{F}$ with t *if*

1. $t \models tp(p^{\mathcal{W}})(x) \wedge tp(p^{\mathcal{F}})(y)$,
2. *if* t *is* T*-transitive then* $t \models T(x,y) \wedge \neg T(y,x)$, $tp(p^{\mathcal{W}}) \in In_T^{\mathcal{F}}$ *and*

$$In_T^{\mathcal{F}} \supseteq In_T^{\mathcal{W}} \cup \{tp^{\mathfrak{D}_T^{\mathcal{W}}}(d) : d \in \mathfrak{D}_T^{\mathcal{W}}\}\ ,$$

3. *if* $t \models F(x,y)$, *then* $t = t_F^{\mathcal{W}}$ *and if* $t \models F(y,x)$, *then* $t = t_F^{\mathcal{F}}$. *(By Def. 7, if* $F \in Trans$ *then* t *cannot be* F*-symmetric.)*

We say that $\mathcal{W}$ *is* petal-connectable *to* $\mathcal{F}$ *at a point* $a \in D_T^{\mathcal{F}}$, *if* $\langle \mathfrak{D}_T^{\mathcal{F}}, a\rangle \cong \langle \mathfrak{D}_T^{\mathcal{W}}, p^{\mathcal{W}}\rangle$ *and* $In_T^{\mathcal{W}} = In_T^{\mathcal{F}}$.

Note that if $\mathcal{W}$ is connectable to $\mathcal{F}$ with a plain 2-type t then also $\mathcal{F}$ is connectable to $\mathcal{W}$ with $\bar{t}$.

Now, we define carpets of flowers as sets of flowers that are syntactic witnesses of a model of a GF^2+TG+FG-sentence.

Definition 9. *Let* $\boldsymbol{\mathcal{F}}$ *be a set of* Δ*-flowers. We say that* $\boldsymbol{\mathcal{F}}$ *is a* Δ-carpet *if the following conditions hold:*

(c1) $\boldsymbol{\mathcal{F}}$ contains flowers necessary for conjuncts of the form (1):
for every conjunct γ *of the form* (1)*:* $\exists x(\alpha(x) \wedge \psi(x))$ *there exists* $\mathcal{F} \in \boldsymbol{\mathcal{F}}$ *such that* $tp(p^{\mathcal{F}}) \models \alpha(x) \wedge \psi(x)$,

(c2) every element of a petal of a flower from $\boldsymbol{\mathcal{F}}$ can be a pistil of another flower from $\boldsymbol{\mathcal{F}}$:
for every $\mathcal{F} \in \boldsymbol{\mathcal{F}}$*, for every* $T \in Trans$*, for every* $a \in D_T^{\mathcal{F}}$ *there exists* $\mathcal{W} \in \mathcal{F}$ *such that* $\mathcal{W}$ *is petal-connectable to* $\mathcal{F}$ *at* a,

(c3) every stamen of a flower from $\boldsymbol{\mathcal{F}}$ can be a pistil of another flower from $\boldsymbol{\mathcal{F}}$:
for every $\mathcal{F} \in \boldsymbol{\mathcal{F}}$*, for every* $F \in Funct$*, if* t_F *is a proper 2-type then there exists* $\mathcal{W} \in \boldsymbol{\mathcal{F}}$ *such that* $\mathcal{W}$ *is connectable to* $\mathcal{F}$ *with* t_F,

(c4) every pistil of a flower from $\boldsymbol{\mathcal{F}}$ may get witnesess for all conjuncts of the form (2b) as pistils of another flower from $\boldsymbol{\mathcal{F}}$:
for every $\mathcal{F} \in \boldsymbol{\mathcal{F}}$*, for every* $F \in FunctInv$*, if* $tp(p)^{\mathcal{F}} \models \neg F(x,x)$ *and there is a conjunct* γ *of* Δ *of the form* (2b) *such that there is no witness of* γ *for* $p^{\mathcal{F}}$ *in* $\mathcal{F}$*, then there exists* $\mathcal{W} \in \boldsymbol{\mathcal{F}}$ *and a* Δ*-acceptable proper 2-type* t *such that* $t = t_F^{\mathcal{W}}$*,* $\mathcal{F}$ *is connectable to* $\mathcal{W}$ *with* t*, and for every conjunct* γ *of the form* (2b)*:* $\gamma = (\forall x.\alpha(x))\,(\exists y.Fyx)\,\psi(x,y)$, $t \models \alpha(x) \rightarrow F(y,x) \wedge \psi(x,y)$ *holds.*

(c5) every pistil of a flower from $\boldsymbol{\mathcal{F}}$ may get witnesess for all remaining conjuncts of the form (2) as pistils of another flower from $\boldsymbol{\mathcal{F}}$:
for every $\mathcal{F} \in \boldsymbol{\mathcal{F}}$*, for every conjunct* γ *of* Δ *of the form* (2)*:* $\gamma = \forall x(\alpha(x) \rightarrow \exists y(\beta(x,y) \wedge \psi(x,y)))$ *that is not of the form* (2b)*, if there is no witness of* γ *for* $p^{\mathcal{F}}$ *in* $\mathcal{F}$*, then there exists* $\mathcal{W} \in \boldsymbol{\mathcal{F}}$ *and a* Δ*-acceptable proper 2-type* t *such that* $t \models \beta(x,y) \wedge \psi(x,y)$*, and*
(a) if γ *is of the form* (2c)*, then* $t = t_F^{\mathcal{W}}$ *and* $\mathcal{F}$ *is connectable to* $\mathcal{W}$ *with* t,
(b) if γ *is of the form* (2d)*, then* $\mathcal{W}$ *is connectable to* $\mathcal{F}$ *with* t,
(c) if γ *is of the form* (2e)*, then* $\mathcal{F}$ *is connectable to* $\mathcal{W}$ *with* t,
(d) if γ *is of the form* (2f)*, then* t *is plain and* $\mathcal{W}$ *is connectable to* $\mathcal{F}$ *with* t.

The main result of this section is the following theorem.

Theorem 3. *The sentence* Δ *is satisfiable if and only if there exists a* Δ*-carpet.*

The above Theorem allows to reduce the satisfiability problem for the sentence Δ to the existence of a Δ-carpet that is shown in the next section to be verifiable in double exponential time. The proof is omitted due to space limits.

4.5 Algorithm for Satisfiability

We prove in this section that the satisfiability problem for GF^2+TG+FG is decidable in deterministic double exponential time. In fact we design an alternating algorithm working in exponential space and the result follows from the theorem that alternating exponential space coincides with double exponential time (see [2] for details on alternating complexity classes).

Theorem 4. *The satisfiability problem for* $GF^2 + TG + FG$ *is* 2Exptime*-complete.*

Proof. Hardness follows from the lower bound for monadic-GF^2+TG given by Kieroński in [12]. For the upper bound we proceed as in [17].

Let Γ be a GF^2+TG+FG-sentence over a signature τ. Let $\mathbf{\Delta}$ be the set of sentences in normal form over a signature σ given by Lemma 1. Then, Γ is satisfiable if and only if there exists a satisfiable sentence $\Delta \in \mathbf{\Delta}$. By Theorem 3, the satisfiability of a sentence $\Delta \in \mathbf{\Delta}$ can be tested by checking the existence of a Δ-carpet. This can be done by an alternating decision procedure that, roughly speeking, keeps at every moment two different Δ-flowers and a counter for the number of steps already performed. The size of one Δ-flower is exponential in the length of Γ. The number of steps can be bounded by the number of distinct Δ-flowers and therefore the space required by the alternating procedure is exponential with respect to the length of Γ. Details are omitted due to space limits.

4.6 Future Work

We plan to extend the decidability result to the fragment with constants. Constans correspond to nominals that are used in the context of both modal and description logics. The technique presented in this paper cannot be directly applied because using constants we can write formulas that do not have ramified models in the sense of Definition 5.

References

1. H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *ILLC Research Report ML-1996-03*, University of Amsterdam. Journal version: *J. Philos. Logic*, 27 (1998), no. 3, 217-274.
2. J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II.* Springer, 1990.
3. D. Calvanese, M. Lenzerini, and D.Nardi. A unified framework for class based representation formalism. In *Proc. of 7th International Conference on Knowledge Representation*, pp. 109-120, Germany. Morgan Kaufman.
4. E. Franconi and G. Ng. The i.com Tool for Intelligent Conceptual Modelling. In *Proc. of 7th International Workshop on Knowledge Representation meets Databases (KRDB'99)*, pp. 45-53 Berlin, Germany. Vol. 29 of CEUR Workshop Proceedings (`http://ceur-ws.org/`).
5. H. Ganzinger, C. Meyer, and M. Veanes. The two-variable guarded fragment with transitive relations. In *14th Annual IEEE Symposium on Logic in Computer Science*, pp. 24–34, 1999.
6. E. Grädel. On the restraining power of guards. *J. Symbolic Logic*, 64:1719–1742, 1999.
7. E. Grädel. Decision procedures for guarded logics. In H. Ganzinger, editor, *Automated Deduction - (CADE-16), 16th International Conference in Artificial Intelligence*, volume 1632 LNCS, pp. 31–51 Trento, Italy. Springer.
8. E. Grädel, M. Otto, and E. Rosen. Two-variable logic with counting is decidable. In *Proc. of 12th IEEE Symposium on Logic in Computer Science (LICS'97)*, pp. 306–317 Warsaw, Poland. IEEE Computer Society Press.
9. E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *14th Annual IEEE Symposium on Logic in Computer Science*, pp. 45–54, 1999.

10. I. Horrocks and U. Sattler and S. Tessaris and S. Tobies. How to decide Query Containment under Constraints using a Description Logic. In *Proc. of 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'2000)*, volume 1955 LNCS, pp. 308-325. Springer.
11. Y. Kazakov and H. de Nivelle. A Resolution Decision Procedure for the Guarded Fragment with Transitive Guards. In *2nd Int. Joint Conference on Automated Reasoning*, volume LNCS 3097, pp. 122–136. Springer, 2004.
12. E. Kieroński. The two-variable guarded fragment with transitive guards is 2ExpTime-Hard. In *Proc. Foundations of Software Science and Computational Structures, 6th Int. Conf.*, LNCS 2620, pp. 299–312. Springer, 2003.
13. L. Pacholski, W. Szwast, and L. Tendera. Complexity results for two-variable logic with counting. In *Proc. of 12th IEEE Symposium on Logic in Computer Science*, pp. 306–317. IEEE Computer Society Press.
14. L.Pacholski, W.Szwast, and L.Tendera. Complexity of first-order two-variable logic with counting. SIAM J. of Computing 29 (4) (2000) 1083-1117.
15. U. Sattler. Description Logics for the Representation of Aggregated Objects. In *Proc. of the 14th European Conference on Artificial Intelligence (ECAI2000)*, pp. 239-243 Berlin, Germany. IOS Press, Amsterdam.
16. W. Szwast and L. Tendera. On the decision problem for the guarded fragment with transitivity. In *16th Annual IEEE Symposium on Logic in Computer Science*, pp. 147–156, 2001.
17. W. Szwast and L. Tendera. The guarded fragment with transitive guards. *Annals of Pure and Applied Logic*, 128:227-276, 2004.
18. J. van Benthem. Dynamics bits and pieces. *ILLC Research Report LP-97-01*, University of Amsterdam, 1997.

The Variable Hierarchy of the μ-Calculus Is Strict*

Dietmar Berwanger[1] and Giacomo Lenzi[2]

[1] Mathematische Grundlagen der Informatik, RWTH Aachen, D-52056 Aachen
berwanger@cs.rwth-aachen.de
[2] Dipartimento di Matematica, Università di Pisa, via Buonarroti 2, I-56127 Pisa
lenzi@mail.dm.unipi.it

Abstract. The modal μ-calculus L_μ attains high expressive power by extending basic modal logic with monadic variables and binding them to extremal fixed points of definable operators. The number of variables occurring in a formula provides a relevant measure of its conceptual complexity. In a previous paper with Erich Grädel we have shown, for the existential fragment of L_μ, that this conceptual complexity is also reflected in an increase of semantic complexity, by providing examples of existential formulae with k variables that are not equivalent to any existential formula with fewer than k variables.

In this paper, we prove an existential preservation theorem for the family of L_μ-formulae over at most k variables that define simulation closures of finite strongly connected structures. Since hard formulae for the level k of the existential hierarchy belong to this family, it follows that the bounded variable fragments of the full modal μ-calculus form a hierarchy of strictly increasing expressive power.

Keywords: μ-calculus, structural complexity.

1 Introduction

Among the various formalisms for reasoning about dynamic systems, the modal μ-calculus L_μ enjoys a prominent position due to its high expressive power and model-theoretic robustness, in balance with its fairly manageable computational complexity. As such, L_μ offers a frame of reference for virtually every logic for specifying the operational behaviour of reactive and concurrent systems.

Typically, such systems are modelled as transition structures with elementary states labelled by propositions and binary transition relations labelled by actions. A simple language for speaking about these models is basic modal logic, or Hennessy-Milner logic [10], which extends propositional logic by modalities associated to actions, i.e., existential and universal quantifiers over the successors of a state which are reachable via the specified action.

* This research has been partially supported by the European Community Research Training Network "Games and Automata for Synthesis and Validation" (GAMES).

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 97–109, 2005.

The μ-calculus of Kozen [12] extends basic modal logic by adding monadic variables bound by least and greatest fixed points of definable operators. This provides a notion of recursion which invests the logic with high expressive power. On the other hand, the variables also import considerable conceptual complexity.

A well studied source of conceptual complexity is the *alternation depth* of L_μ-formulae, that is, the number of (genuine) alternations between least and greatest fixed points. In [7] Bradfield showed that the alternation hierarchy of the μ-calculus is semantically strict; variants of this result have also been proved by Lenzi [15] and Arnold [1]. Hence, this notion of syntactic complexity of a formula is reflected in its semantic complexity.

Interestingly, most of the formalisms commonly used for process description allow translations into low levels of the L_μ alternation hierarchy. On its first level this hierarchy already captures, for instance, PDL as well as CTL, while their expressive extensions ΔPDL and CTL* do not exceed the second level. Still, the low levels of this hierarchy do not exhaust the significant properties expressible in L_μ. A comprehensive example of formulae distributed over all levels of the alternation hierarchy is provided by parity games. Thus, strictly on level n, there is a formula stating that the first player has a winning strategy in parity games with n priorities.

By reusing fixed point variables several times, it is possible to write many L_μ-formulae, even with highly nested fixed-point definitions, using only very few variables. For any k, let us denote by $\mathrm{L}_\mu[k]$ the fragment of L_μ consisting of those formulae that make use of at most k distinct fixed-point variables. It turns out that most specification properties of transition structures can be embedded into $\mathrm{L}_\mu[2]$. This is actually the case for all the aforementioned formalisms, CTL, PDL, CTL*, and ΔPDL (see [17]). But the expressive power of the two-variable fragment of L_μ goes well beyond this. As shown in [3], the formulae describing the winning position of a parity game, can also be written with only two variables.

In this context, the question arises, whether a higher number of variables is indeed necessary, or, in other words, whether the number of variables of a formula is reflected as a measure of its semantic complexity.

As a first step towards answering this question, we have proved, together with Grädel in [5], that the variable hierarchy of the existential fragment of L_μ is strict. This syntactic fragment, consisting of the formulae built from atoms and negated atoms by means of boolean connectives, existential modalities, and least and greatest fixed points, admits a strong semantic characterisation. In [8], D'Agostino and Hollenberg proved that it captures precisely those L_μ-expressible properties ψ that are preserved under extensions, in the sense that whenever $\mathcal{K}, v \models \psi$ and $\mathcal{K} \subseteq \mathcal{K}'$, then also $\mathcal{K}', v \models \psi$. Unfortunately, the technique used in their proof does not comply with a variable-confined setting, and the question whether the variable hierarchy is strict for the full μ-calculus remained open.

To witness the strictness of the variable hierarchy in the restricted existential case considered in [5], we provided examples of formulae $\psi^k \in \mathrm{L}_\mu[k]$, for each level k, that cannot be equivalently expressed by any formula over less than k variables using only existential modalities. Essentially, the formulae ψ^k describe

the class of structures extending a clique with k states, where every pair of states i, j is linked by a transition labelled ij.

In the present paper, we prove a preservation theorem stating that every formula defining the extensions of a finite strongly connected structure can be transformed into an existential formula without increasing the number of variables. In particular, this holds for the witnesses ψ^k to the strictness of the existential hierarchy provided in [5]. Consequently, even if the use of universal modalities is allowed, none of the formulae ψ^k can be equivalently written as a formula with less than k variables. In this way, we can answer positively the question concerning the strictness of the variable hierarchy of the full μ-calculus by reducing it to the existential case.

Besides revealing a new aspect of the rich inner structure of the μ-calculus, this result settles an open question formulated in [17] regarding the expressive power of Parikh's Game Logic GL. This logic, introduced in [16] as a generalisation of PDL for reasoning about games, subsumes ΔPDL and intersects nontrivially all the levels of the L_μ alternation hierarchy [3]. When interpreted on transition structures, GL can be translated into $\mathrm{L}_\mu[2]$. However it was unknown, up to now, whether the inclusion in L_μ was proper. The strictness of the variable hierarchy implies that already $\mathrm{L}_\mu[3]$ is more expressive than GL.

The paper is structured as follows. In Section 2, we introduce the necessary background on the μ-calculus. Section 3 is dedicated to the proof of the Preservation Theorem. We conclude by stating the Hierarchy Theorem in Section 4.

2 The Modal μ-Calculus

Fix a set ACT of actions and a set PROP of atomic propositions. A transition structure for ACT and PROP is a structure $\mathcal{K}$ with universe K (whose elements are called *states*), binary relations $E_a \subseteq K \times K$ for each $a \in$ ACT, and monadic relations $p \subseteq K$ for each atomic proposition $p \in$ PROP.

Syntax. For a set ACT of actions, a set PROP of atomic propositions, and a set VAR of monadic variables, the formulae of L_μ are defined by the grammar

$$\varphi ::= \mathit{false} \mid \mathit{true} \mid p \mid \neg p \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where $p \in$ PROP, $a \in$ ACT, and $X \in$ VAR. An L_μ-formula in which no universal modality $[a]\varphi$ occurs is called *existential*.

The number of distinct variables appearing in an L_μ-formula induces the following syntactic hierarchy. For any $k \in \mathbb{N}$, the *k-variable fragment* $\mathrm{L}_\mu[k]$ of the μ-calculus is the set of formulae $\psi \in \mathrm{L}_\mu$ that contain at most k distinct variables.

Semantics. Formulae of L_μ are evaluated on transition structures at a particular state. Given a sentence ψ and a structure $\mathcal{K}$ with state v, we write $\mathcal{K}, v \models \psi$ to denote that ψ holds in $\mathcal{K}$ at state v. The set of states $v \in K$ such that $\mathcal{K}, v \models \psi$ is denoted by $[\![\psi]\!]^{\mathcal{K}}$.

Here, we only define $[\![\psi]\!]^{\mathcal{K}}$ for fixed-point formulae ψ. Towards this, note that a formula $\psi(X)$ with a monadic variable X defines on every transition structure $\mathcal{K}$ (providing interpretations for all free variables other than X occurring in ψ) an operator $\psi^{\mathcal{K}} : \mathcal{P}(K) \to \mathcal{P}(K)$ assigning to every set $X \subseteq K$ the set $\psi^{\mathcal{K}}(X) := [\![\psi]\!]^{\mathcal{K},X} = \{v \in K : (\mathcal{K}, X), v \models \psi\}$. As X occurs only positively in ψ, the operator $\psi^{\mathcal{K}}$ is *monotone* for every $\mathcal{K}$, and therefore, by a well-known theorem due to Knaster and Tarski, has a least fixed point $\mathrm{lfp}(\psi^{\mathcal{K}})$ and a greatest fixed point $\mathrm{gfp}(\psi^{\mathcal{K}})$. Now we put

$$[\![\mu X.\psi]\!]^{\mathcal{K}} := \mathrm{lfp}(\psi^{\mathcal{K}}) \text{ and } [\![\nu X.\psi]\!]^{\mathcal{K}} := \mathrm{gfp}(\psi^{\mathcal{K}}).$$

Least and greatest fixed points can also be constructed inductively. Given a formula $\nu X.\psi$, we define for each ordinal α, the stage X^α of the gfp-induction of $\psi^{\mathcal{K}}$ by $X^0 := K$, $X^{\alpha+1} := [\![\psi]\!]^{(\mathcal{K},X^\alpha)}$, and $X^\alpha := \bigcap_{\beta<\alpha} X^\beta$ if α is a limit ordinal. By monotonicity, the stages of the gfp-induction decrease until a fixed point is reached. By ordinal induction, one easily proves that this inductively constructed fixed point coincides with the greatest fixed point. The *finite approximants* of a formula $\nu X.\varphi$ are defined by $\varphi_0 := \mathit{true}$ and $\varphi_{n+1} = \varphi[X/\varphi_n]$ (the formula obtained by replacing every occurrence of X in φ, by φ_n). Obviously, $\nu X.\varphi$ implies φ_n for all n. Likewise, but starting from false, one defines the approximants φ_n of $\mu X.\varphi$.

The validity of existential L_μ-formulae is preserved under model extensions and, more generally, under the following notion of simulation.

Definition 1. A *simulation* from a structure $\mathcal{K}$ to a structure $\mathcal{K}'$ is a relation $Z \subseteq K \times K'$ respecting the atomic propositions $p \in \textsc{prop}$ in the sense that $\mathcal{K}, v \models p$ iff $\mathcal{K}', v' \models p$, for $(v, v') \in Z$, which satisfies the following condition. For all $(v, v') \in Z$, $a \in \textsc{act}$, and every w such that $(v, w) \in E_a$, there exists a $w' \in K'$ such that $(v', w') \in E'_a$ and $(w, w') \in Z$. We say that $\mathcal{K}', u'$ *simulates* $\mathcal{K}, u$ and write $\mathcal{K}, u \lesssim \mathcal{K}', u'$, if between the structures there is a simulation containing (u, u').

As a modal logic, the μ-calculus distinguishes between transitions structures only up to behavioural equivalence, captured by the notion of bisimulation.

Definition 2. A *bisimulation* between two transition structures $\mathcal{K}$ and $\mathcal{K}'$ is a simulation Z from $\mathcal{K}$ to $\mathcal{K}'$ such that the inverse relation Z^{-1} is a simulation from $\mathcal{K}'$ to $\mathcal{K}$. Two transition structures $\mathcal{K}, u$ and $\mathcal{K}', u'$ are *bisimilar*, if there is a bisimulation Z between them, with $(u, u') \in Z$.

An important model-theoretic feature of modal logics is the *tree model property* meaning that every satisfiable formula is satisfiable in a tree. This is a straightforward consequence of bisimulation invariance, since $\mathcal{K}, u$ is bisimilar to its *tree unravelling*, i.e., a tree whose nodes correspond to the finite paths in $\mathcal{K}, u$. Every such path π inherits the atomic propositions of its last node v; for every node w reachable from v in $\mathcal{K}$ via an a transition, π is connected to its prolongation by w via an a-transition.

Another significant feature of L_μ is its *finite model property*.

Theorem 1 ([13])**.** *Every satisfiable* L_μ*-formula has a finite model.*

Since the unravelling of a finite model is a finitely branching tree, we obtain the following corollary.

Corollary 1. *Every satisfiable* L_μ*-formula holds in some finitely branching tree.*

For later use, we state a further consequence of the finite model property.

Corollary 2. *For* $\psi \in \mathrm{L}_\mu$*, let* $\psi[\nu := \nu^n]$ *denote the result of replacing every occurrence* $\nu X.\eta$ *of a* ν*-predicate in* ψ *with its* n*-th approximant* η_n*. Then, a formula* $\varphi \in \mathrm{L}_\mu$ *implies* ψ *if, and only if,* φ *implies* $\psi[\nu := \nu^n]$*, for each* n*.*

Model-Checking Games. The semantics of L_μ can also be described in terms of *parity games*, in which two players form a path in a given graph with nodes labelled by natural numbers called priorities. If a player cannot move, he loses. If this never occurs, the winner is decided by looking at the (infinite) sequence of priorities occurring in the play. The first player wins if the least priority appearing infinitely often in this sequence is even, otherwise his opponent wins. The Forgetful Determinacy Theorem states that these games are always determined, and the winner has a memoryless winning strategy, that is, a strategy that does not depend on the history of the play but only on the current position.

Theorem 2 (Forgetful Determinacy, [9])**.** *In any parity game, one of the players has a memoryless winning strategy.*

Given a transition structure $\mathcal{K}, v_0$ and a L_μ-sentence ψ, the model-checking game $\mathcal{G}(\mathcal{K}, \psi)$ is a parity game associated with the problem whether $\mathcal{K}, v_0 \models \psi$. Deviating from the more traditional way to define this game with positions associated to subformulae of ψ (see, e.g., [4, 18]), we use a variant more familiar in automata theory which, instead of subformulae, refers to their closure [9, 14].

Definition 3. Let $\psi \in \mathrm{L}_\mu$ be a formula without free variables. For each subformula φ in ψ, we define its *closure* $\mathrm{cl}_\psi(\varphi)$ as the formula obtained by replacing recursively every free occurrence of a variable in φ by its binding definition. By $\mathrm{cl}(\psi)$ we denote the set of closures of all subformulae in ψ.

The positions in the game $\mathcal{G}(\mathcal{K}, \psi)$ are pairs (v, φ) of states $v \in K$ and sentences $\varphi \in \mathrm{cl}(\psi)$. The first player, called Verifier, moves from the positions $(v, \varphi_1 \vee \varphi_2)$, $(v, \langle a \rangle \varphi)$, (v, p) with $v \notin p$, and $(v, \neg p)$ with $v \in p$ and his opponent, called Falsifier, moves from every other position. All plays start at (v_0, ψ) and can proceed as follows:

- no moves are possible from (v, α) where α is atomic or negated atomic;
- from $(v, \varphi_1 \vee \varphi_2)$ or $(v, \varphi_1 \wedge \varphi_2)$ available moves lead to (v, φ_1) and (v, φ_2);
- from $(v, \langle a \rangle \varphi)$ or $(v, [a]\varphi)$ there are available moves to all positions (w, φ) where w is an a-successor of v;
- from $(v, \lambda X.\varphi(X))$ the play moves to $(v, \varphi(\lambda X.\varphi(X)))$.

The priority labelling assigns even priorities to positions $(v, \nu X.\varphi)$ and odd priorities to positions $(v, \mu X.\varphi)$, respecting the nesting of greatest and least fixed-point operators. For details (which are not needed in this paper), see [4].

Theorem 3 ([18]). *Verifier has a winning strategy in the model-checking game* $\mathcal{G}(\mathcal{K}, \psi)$ *from position* (u, ψ) *iff* $\mathcal{K}, u \models \psi$.

Simultaneous Fixed Points. There is a variant of L_μ that admits simultaneous fixed points of several formulae. This does not increase the expressive power but allows more transparent formalisations. The mechanism for building simultaneous fixed-point formulae is the following. Given formulae $\varphi_1, \ldots, \varphi_n$ and variables $X_1, \ldots, X_n$, we can write an *equational system* $S := \{X_1 = \varphi_1, \ldots, X_n = \varphi_n\}$ and build formulae $(\mu X_i : S)$ and $(\nu X_i : S)$. On every structure $\mathcal{K}$, the system S defines an operator $S^{\mathcal{K}}$ mapping an n-tuple $\bar{X} = (X_1, \ldots, X_n)$ of sets of states to $S_1^{\mathcal{K}}(\bar{X}), \ldots, S_n^{\mathcal{K}}(\bar{X})$ so that, for each i we have: $S_i^{\mathcal{K}}(\bar{X}) := [\![\varphi_i]\!]^{(\mathcal{K}, \bar{X})}$. As $S^{\mathcal{K}}$ is monotone, it has extremal fixed points $\mathrm{lfp}(S) = (X_1^\mu, \ldots, X_n^\mu)$ respectively $\mathrm{gfp}(S) = (X_1^\nu \ldots, X_n^\nu)$, and we set $[\![(\mu X_i : S)]\!]^{\mathcal{K}} := X_i^\mu$ and $[\![(\nu X_i : S)]\!]^{\mathcal{K}} := X_i^\nu$.

It is known that simultaneous least fixed points can be eliminated in favour of nested individual fixed points.

Proposition 1 ([2]). *Every formula in* L_μ *with simultaneous fixed points can be translated into an equivalent formula in plain* L_μ *without increasing the number of variables.*

3 The Preservation Theorem

The key argument in our proof of the Hierarchy Theorem consists in the preservation property stated in the current section, which implies that the formulae proposed in [5] to separate the hierarchic levels of the existential fragment also witness the strictness of the full μ-calculus variable hierarchy.

This preservation property concerns formulae which define simulation closures of certain structures. The *simulation closure* of a rooted transition structure $\mathcal{K}, v_0$ is the class

$$(\mathcal{K}, v_0)^{\lesssim} := \{\, \mathcal{K}', v_0' \mid \mathcal{K}, v_0 \lesssim \mathcal{K}', v_0' \,\}.$$

Clearly, if $\mathcal{K}$ is finite, this class can be axiomatised by an L_μ-formula. For convenience, we will use simultaneous fixed points. Let S be the system defining, for every node $v \in K$, a proposition X_v via the equation

$$X_v = \bigwedge_{p \mid v \in p} p \wedge \bigwedge_{a \in \mathrm{ACT}, (v,w) \in E_a} \langle a \rangle X_w.$$

It can be easily seen that on any transition structure $\mathcal{K}'$, the greatest solution of this system maps every variable X_v to the set $\{\, v' \in K' \mid \mathcal{K}, v \lesssim \mathcal{K}', v' \,\}$. Hence, for any state $v' \in [\![\nu X_v : S]\!]^{\mathcal{K}'}$, we have $\mathcal{K}, v \lesssim \mathcal{K}', v'$.

For further use, let us denote the L_μ-formula obtained as a translation of the equational expression $\nu X_v : S$ by $\psi_v^{\mathcal{K}}$. For the formula $\psi_{v_0}^{\mathcal{K}}$ associated to the designated root v_0 of $\mathcal{K}$, we write $\psi^{\mathcal{K}}$, and call it the *canonical axiom* of $(\mathcal{K}, v_0)^{\lesssim}$.

Our main technical contribution is stated in the following theorem.

Theorem 4. *Every formula over k variables $\psi \in \mathrm{L}_\mu[k]$ that defines the simulation closure $(\mathcal{K}, v_0)^{\lesssim}$ of a finite strongly connected structure is equivalent to an existential formula $\psi' \in \mathrm{L}_\mu[k]$.*

To prove that universal modalities can be safely eliminated from any formula ψ of the considered kind, we take a detour and first show that they can be eliminated from the formula expressing that a node at which ψ holds is reachable. To refer to this formula, we use a shorthand borrowed from temporal logics:

$$\mathsf{F}\psi := \mu X.\psi \vee \bigvee_{a \in \mathrm{ACT}} \langle a \rangle X.$$

Lemma 4 in the second part of this section then states that from any formula equivalent to $\mathsf{F}\psi$, an existential formula equivalent to ψ can be recovered without increasing the number of variables.

Lemma 1. *Let $\mathcal{K}$ be a finite strongly connected structure and let $\psi^{\mathcal{K}}$ be the canonical axiom of its simulation closure $(\mathcal{K}, v_0)^{\lesssim}$. Then, every formula $\chi \equiv \mathsf{F}\psi^{\mathcal{K}}$ can be transformed, without increasing the number of variables, into an equivalent formula χ' with the following properties:*

(i) *no universal modalities occur in χ';*
(ii) *χ' is of shape $\mathsf{F}\psi$, where ψ contains no μ-operators;*
(iii) *every formula $\varphi \in \mathrm{cl}(\chi')$ holds at some vertex of $\mathcal{K}$.*

Proof. (i) Given an L_μ-formula χ, we say that a subformula $\langle a \rangle \varphi$ starting with a diamond is *vital*, if $\mathrm{cl}_\chi(\varphi)$ implies $\mathsf{F}\psi^{\mathcal{K}}$. Dually, a subformula $[a]\varphi$ starting with a box is vital, if the negation $\neg\, \mathrm{cl}_\chi(\varphi)$ implies $\mathsf{F}\psi^{\mathcal{K}}$.

Eliminating Vital Boxes. For $\chi \equiv \mathsf{F}\psi^{\mathcal{K}}$, let χ' be the formula obtained by replacing any occurrence of a vital box-subformula $[a]\varphi$ with *true*. Then, χ obviously implies χ'. For the converse, let us consider a tree model $\mathcal{T}$ of χ'. If, at all its nodes, $\mathcal{T}, v \models [a]\, \mathrm{cl}_\chi(\varphi)$ holds, then $\mathcal{T} \models \chi$. Else, there exists a node $v \in T$ with $\mathcal{T}, v \models \langle a \rangle \neg\, \mathrm{cl}_\chi(\varphi)$. But, since $[a]\varphi$ is vital, this means that $\mathcal{T}, v$ and hence $\mathcal{T}$ verifies $\mathsf{F}\psi^{\mathcal{K}}$. Either way, we obtain $\mathcal{T} \models \chi$ and hence $\chi \equiv \chi'$.

Eliminating Non-vital Modalities. By iterating the above elimination step a finite number of times, we obtain a formula $\chi \equiv \mathsf{F}\psi^{\mathcal{K}}$ without vital box-subformulae. Let now χ' be the formula obtained from χ by substituting simultaneously all remaining (i.e., non-vital) box-subformulae with *false* and all non-vital diamond-subformulae with *true*.

We will first show that the obtained formula χ' implies χ. Let $\mathcal{T}$ be a tree model of χ' and, for every non-vital subformula $\langle a \rangle \varphi$ of χ, let $\mathcal{T}_\varphi$ be a tree model

of $\mathrm{cl}_\chi(\varphi) \wedge \neg \mathsf{F}\psi^{\mathcal{K}}$. With the latter models, we construct an extension $\mathcal{T}'$ of $\mathcal{T}$ by introducing for every node $v \in T$ and every non-vital subformula $\langle a \rangle \varphi$ of χ, a fresh copy of $\mathcal{T}_\varphi$ to which we connect v via an a-edge.

Since χ' contains no box-subformulae, it is closed under extensions. Consequently $\mathcal{T}' \models \chi'$ and Verifier has a winning strategy σ in the model-checking game $\mathcal{G}(\mathcal{T}', \chi')$. Also, for every tree $\mathcal{T}_\varphi$, Verifier has a winning strategy σ_φ in the game $\mathcal{G}(\mathcal{T}_\varphi, \mathrm{cl}_\chi(\varphi))$. We can combine these strategies, to obtain a winning strategy for Verifier in the game $\mathcal{G}(\mathcal{T}', \chi)$ as follows. Move according to σ unless a position with a non-vital subformula of χ is met; up to that point, the play cannot leave T, otherwise, since $\mathsf{F}\psi^{\mathcal{K}}$ is falsified at any node $w \in T' \setminus T$, any vital subformula $\langle a \rangle \varphi$ would fail at w. Moreover, no subformula $[a]\varphi$ can occur, as it would correspond to a *false* position in $\mathcal{G}(\mathcal{T}', \chi')$. Consequently, σ leads the play to a position $(v, \langle a \rangle \varphi)$ where $v \in T$ and $\langle a \rangle \varphi$ is non-vital. At that event, let the Verifier choose the a-successor at the root of $\mathcal{T}_\varphi$ and proceed with his memoryless winning strategy σ_φ for the remaining game. In this way, Verifier finally wins any play of $\mathcal{G}(\mathcal{T}', \chi)$. Notice that, for all nodes $w \in T' \setminus T$, we have $\mathcal{T}', w \not\models \mathsf{F}\psi^{\mathcal{K}}$, and hence $\mathcal{T}'$ verifies $\mathsf{F}\psi^{\mathcal{K}}$ (or, equivalently, χ) if, and only if, $\mathcal{T}$ does. Hence, we have the following chain of implications, showing that χ' implies χ:

$$\mathcal{T} \models \chi' \implies \mathcal{T}' \models \chi' \implies \mathcal{T}' \models \chi \implies \mathcal{T} \models \chi.$$

For the converse, consider a tree model $\mathcal{T} \models \chi$ and, for every (non-vital) subformula $[a]\varphi$ of χ, a tree model $\mathcal{T}_{\neg\varphi} \models \neg\, \mathrm{cl}_\chi(\varphi) \wedge \neg \mathsf{F}\psi^{\mathcal{K}}$. As in the previous step, we construct an extension $\mathcal{T}'$ of $\mathcal{T}$ by connecting every node $v \in T$ via an a-edge to a fresh copy of $\mathcal{T}_{\neg\varphi}$, for every subformula $[a]\varphi$ of χ. Since $\chi \equiv \mathsf{F}\psi^{\mathcal{K}}$ is preserved under extensions, $\mathcal{T}'$ is still a model of χ. Let σ be a winning strategy for Verifier in the model-checking game $\mathcal{G}(\mathcal{T}', \chi)$. We will show that σ is also a winning strategy for Verifier in $\mathcal{G}(\mathcal{T}, \chi')$.

Notice that, in $\mathcal{G}(\mathcal{T}', \chi)$ Falsifier has a winning strategy from every position $(v, [a]\varphi)$ with $v \in T$, by moving to the a-successor of v at the root of $\mathcal{T}_{\neg\varphi}$. Consequently, any play according to Verifier's strategy σ will avoid such positions. Besides this, at every position $(v, \langle a \rangle \varphi)$ where $v \in T$ and $\langle a \rangle \varphi$ is a vital subformula of χ, the strategy σ will appoint a successor position (w, φ) with $w \in T$, otherwise, since any a-successor $w' \in T' \setminus T$ falsifies $\mathsf{F}\psi^{\mathcal{K}}$, φ would fail too. Summarising, every play of $\mathcal{G}(\mathcal{T}', \chi)$ according to σ, will avoid universal modalities and meet only nodes $v \in T$, unless at some position a non-vital subformula $\langle a \rangle \varphi$ occurs. But under these conditions, we can replicate every play of $\mathcal{G}(\mathcal{T}', \chi)$ according to σ as a play of $\mathcal{G}(\mathcal{T}, \chi')$: in case a non-vital subformula $\langle a \rangle \varphi$ of χ is met in the former game, Verifier immediately wins $\mathcal{G}(\mathcal{T}, \chi')$, since the non-vital diamond-subformulae have been replaced by *true*. Otherwise, the outcome of the play is the same for both games and Verifier wins as well.

This concludes the proof that $\chi \equiv \chi'$.

(ii) By the above result, we can assume without loss that $\chi \equiv \mathsf{F}\psi^{\mathcal{K}}$ contains no box-modalities. For n being the number of states in $\mathcal{K}$, let ψ be the formula obtained by replacing every occurrence of a least fixed-point subformula $\mu X.\varphi$ in χ by it's n-th approximant φ^n. Then, by definition of the μ-operator, ψ implies χ

and thus $\mathsf{F}\psi$ implies $\mathsf{F}\chi$, which is equivalent to χ. Conversely, since $\mathcal{K}, v_0 \models \chi$ and $\mathcal{K}$ has n states, we have $\mathcal{K}, v_0 \models \psi$. Since ψ is preserved under extensions, this means that $\psi^{\mathcal{K}}$ implies ψ. Accordingly $\mathsf{F}\psi^{\mathcal{K}}$, which is equivalent to χ, implies $\mathsf{F}\psi$. Hence, $\chi \equiv \mathsf{F}\psi$.

Note that the transformation of χ into $\mathsf{F}\psi$ does not increase the number of variables, as we can pick any of the variables already occurring in χ to expand the F-notation.

(iii) By the previous argument, we can assume that χ is of shape $\mathsf{F}\psi$ where ψ contains no boxes, i.e., $\chi = \mu X.\psi \vee \bigvee_a \langle a \rangle X$. Clearly, χ itself holds at every node of $\mathcal{K}$ and therefore, for every transition a occurring in $\mathcal{K}$, there is a node $v \in K$ where $\langle a \rangle \chi$, and thus $\mathrm{cl}_\chi\big(\langle a \rangle X\big)$, holds. Hence, any subformula φ of χ, with $\mathcal{K}, v \not\models \mathrm{cl}_\chi(\varphi)$ for all v, must actually be a subformula of ψ. Let ψ' be the formula obtained by replacing every such occurrence φ in ψ with *false*. On the one hand, ψ' then obviously implies ψ. On the other hand, as $\mathcal{K}, v_0 \models \mathsf{F}\psi$, there must exists a node v of $\mathcal{K}$ where ψ holds. At that node we also have $\mathcal{K}, v \models \psi'$ and, because ψ' is closed under extensions, this means that $\psi'^{\mathcal{K}}_v$ implies ψ'. But then $\mathsf{F}\psi^{\mathcal{K}}$ implies $\mathsf{F}\psi'$ and, by $\mathsf{F}\psi \equiv \mathsf{F}\psi^{\mathcal{K}}$, it follows that $\mathsf{F}\psi$ implies $\mathsf{F}\psi'$. □

Radical Formulae and Crisp Models. Before we proceed towards proving the Preservation Theorem, we will introduce some notions which will be useful in the proof of Lemma 4

Given a formula $\psi \in \mathrm{L}_\mu$, we call a subformula φ *radical*, if it appears directly under a modal quantifier in ψ. We refer to the closure of radicals in ψ by

$$\mathrm{cl}_0(\psi) := \{\psi\} \cup \{\, \varphi \in \mathrm{cl}(\psi) \mid \langle a \rangle \varphi \in \mathrm{cl}(\psi) \text{ or } [a]\varphi \in \mathrm{cl}(\psi) \text{ for some } a \in \textsc{act} \,\}.$$

Radical formulae are the first to be met when a play of the model-checking game reaches a new node of the transition structure. For this reason, we need to care for game positions carrying radical formulae when merging strategies of different games.

Let $\mathcal{M}$ be a model of $\psi \in \mathrm{L}_\mu$ and σ a winning strategy for Verifier in $\mathcal{G}(\mathcal{M}, \psi)$. For any node $v \in M$, we define the *strategic type* of v in $\mathcal{M}$ under σ as follows:

$$\mathrm{tp}^{\mathcal{M}}_\sigma(v) := \{\, \varphi \in \mathrm{cl}_0(\psi) \mid \text{position } (v, \varphi) \text{ is reachable in } \mathcal{G}(\mathcal{M}, \psi) \text{ following } \sigma \,\}.$$

In arbitrary games, the type of a node can be rather complex. However, for existential formulae, Verifier has full control over the moves in the transition structure. In the ideal case, he can foresee for every node, a single radical formula to be proven there.

Given a transition structure $\mathcal{M}$ and a formula ψ, we say that a Verifier strategy σ in the model-checking game $\mathcal{G}(\mathcal{M}, \psi)$ is *crisp*, if the strategic type $\mathrm{tp}^{\mathcal{M}}_\sigma(v)$ of any $v \in M$ consists of not more than one radical. Accordingly, we call a model $\mathcal{M}$ of ψ crisp (under σ), if Verifier has a crisp winning strategy σ in the associated model-checking game.

The subsequent lemmas, that can be easily proved, provide us with sharp tools for manipulating models of existential formulae.

Lemma 2. *Every existential formula $\psi \in \mathrm{L}_\mu$ satisfied in some model $\mathcal{M} \models \psi$ also has a tree model $\mathcal{T}$ bisimilar to $\mathcal{M}$ which is crisp. Moreover, if $\mathcal{M}$ is finitely branching, then $\mathcal{T}$ can be chosen so as well.*

Lemma 3. *Let $\mathcal{T}$ be a crisp tree model of a formula $\psi \in \mathrm{L}_\mu$ under a strategy σ and let $x \in T$ be a node with strategic type $\mathrm{tp}^{\mathcal{T}}_{\sigma}(x) = \{\varphi\}$. Then, for every crisp tree model $\mathcal{S}$ of φ, the tree $\mathcal{T}[x/\mathcal{S}]$, obtained by replacing the subtree of $\mathcal{T}$ rooted at x with $\mathcal{S}$, is still a crisp model of ψ.*

We are now ready for the final step, the elimination of the F-operator.

Lemma 4. *Let $\psi^{\mathcal{K}}$ be the canonical axiom for the simulation closure $(\mathcal{K}, v_0)^{\lesssim}$ of a finite strongly connected structure $\mathcal{K}$. Then, every formula ψ so that $\mathsf{F}\psi \equiv \mathsf{F}\psi^{\mathcal{K}}$ can be transformed, without increasing the number of variables, into a formula ψ' without universal modalities, so that $\psi' \equiv \psi^{\mathcal{K}}$.*

Proof. According to Lemma 1, we can assume that ψ contains no universal modalities or least fixed point operators and that (the closure of) every subformula is true at some node in $\mathcal{K}$.

We will first show that for any node v in $\mathcal{K}$, there is a subformula φ of ψ whose closure $\mathrm{cl}_\psi(\varphi)$ implies $\psi^{\mathcal{K}}_v$. Actually, we always find a radical formula with this property.

Towards a contradiction, let us assume that $\psi^{\mathcal{K}}_v$ is not implied by any radical subformula of ψ. This means that every $\varphi \in \mathrm{cl}_0(\psi)$ has a tree model $\mathcal{T}_\varphi$ which falsifies $\psi^{\mathcal{K}}_v$. According to Corollary 2, we can choose $\mathcal{T}_\varphi$ to be a finitely branching tree that falsifies already an approximant of $\psi^{\mathcal{K}}_v$ to some finite stage m_φ. Observe that this approximant $(\psi^{\mathcal{K}}_v)[\nu := \nu^{m_\varphi}]$ is a modal formula. Let us denote its modal depth by n_φ. Further, let us fix a number n which is greater than any n_φ for $\varphi \in \mathrm{cl}_0(\psi)$ and co-prime to every number up to $|K|$.

By Lemma 2, we can assume without loss of generality that each $\mathcal{T}_\varphi$ is a crisp model of φ, this being witnessed by a crisp winning strategy for Verifier in the game $\mathcal{G}(\mathcal{T}_\varphi, \varphi)$. In particular, $\mathcal{T}_\psi$ is a crisp model of ψ. Let σ_ψ be a crisp winning strategy for Verifier in the model-checking game $\mathcal{G}(\mathcal{T}_\psi, \psi)$.

By means of these, we construct a sequence of trees $(\mathcal{T}_i)_{0 \le i < \omega}$, together with crisp Verifier strategies σ_i witnessing that $\mathcal{T}_i \models \psi$. To start, we set $\mathcal{T}_0 := \mathcal{T}_\psi$ and $\sigma_0 := \sigma_\psi$. In every step $i > 0$, the tree $\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ by performing the following manipulations at depth $n(i+1)$. For each subtree of $\mathcal{T}_i$ rooted at a node x of this depth, we check whether $\mathcal{T}_i, x \models \psi^{\mathcal{K}}_v$. If this is not the case, the subtree remains unchanged. Else, we look at the strategic type of x under σ_i. If the type is empty, we simply cut all successors of x. Otherwise, $\mathrm{tp}^{\mathcal{T}_i}_{\sigma_i}(x)$ consists of a single radical formula φ, and we replace the subtree $\mathcal{T}_i, x$ with $\mathcal{T}_\varphi$. According to Lemma 3, the resulting tree $\mathcal{T}_{i+1}$ is a model of ψ, and the composition of the strategy σ_i with the crisp strategies σ_φ on the newly appended subtrees $\mathcal{T}_\varphi$ yields a crisp Verifier strategy σ_{i+1} for the model-checking game $\mathcal{G}(\mathcal{T}_{i+1}, \psi)$.

By construction, each of the trees $\mathcal{T}_i$ is finitely branching and the sequence $(\mathcal{T}_i)_{0 \le i < \omega}$ converges in the prefix topology of finitely branching trees (see [11]). Let $\mathcal{T}_\omega$ be the limit of this sequence. Since no μ-operators occur in ψ, its model

class is topologically closed on finitely branching trees, according to [11]. Consequently, $\mathcal{T}_\omega$ is still a model of ψ. By our hypothesis, ψ implies $\mathsf{F}\psi^{\mathcal{K}}$. Thus, at some depth d in $\mathcal{T}_\omega$ a node x with $\mathcal{T}_\omega, x \models \psi_v^{\mathcal{K}}$ appears. Since $\mathcal{K}$ is strongly connected, v must lie on a cycle in $\mathcal{K}$. Hence, for $k \leq |K|$ being the length of such a cycle, there exist nodes y with $\mathcal{T}_\omega, y \models \psi_v^{\mathcal{K}}$ at every depth $d + jk$. However, our construction eliminated all subtrees carrying the similarity type of v at depths multiple of n. Since n was chosen to be co-prime to any integer up to $|K|$, it follows that $\mathcal{T}_\omega$ cannot satisfy ψ. This is a contradiction which invalidates our assumption that $\psi_v^{\mathcal{K}}$ is not implied by any $\varphi \in \mathrm{cl}_0(\psi)$.

Hence, for every node $v \in K$, there exists a formula $\varphi_v \in \mathrm{cl}_0(\psi)$ implying $\psi_v^{\mathcal{K}}$. We can show that the converse also holds, if v is maximal with respect to the preorder $\lesssim$, in the sense that for every w with $v \lesssim w$ we have $w \lesssim v$. Recall that, by Lemma 1 (iii), the formula φ_v must be verified at some node w in $\mathcal{K}$. Since φ_v is existential and thus preserved under extension, it follows that $\psi_w^{\mathcal{K}}$ implies φ_v, which further implies $\psi_v^{\mathcal{K}}$. But this means that $v \lesssim w$ and, by maximality of v, that w simulates v. Hence, $\mathcal{K}, v \models \varphi_v$ and consequently $\psi_v^{\mathcal{K}} \equiv \varphi_v$.

This concludes the proof for the case when v_0 is maximal in $\mathcal{K}$ with respect to $\lesssim$. Otherwise, we could not guarantee, of course, that $\varphi_{v_0} \equiv \psi_{v_0}^{\mathcal{K}}$. But in that case, a formula equivalent to $\psi_v^{\mathcal{K}}$ can be recovered from $\mathrm{cl}_0(\psi)$ without great difficulty. □

4 The Hierarchy Theorem

In [5], it was shown that every level k of the variable hierarchy contains existential formulae which are not equivalent to any existential formula from a lower hierarchical level. Examples of such formulae are obtained by considering so-called clique structures $\mathcal{C}^k$ over the set of states $\{0, \ldots, k-1\}$ with transition relations $E_{ij} = \{(i, j)\}$, for $0 \leq i, j < k$. For each k, the canonical axiom ψ^k of the simulation closure of $\mathcal{C}^k$ is an existential L_μ-formula over k variables. The Hierarchy Theorem for the existential fragment states that, if we restrict to formulae using only existential quantification, k variables are indeed necessary.

Theorem 5 ([5])**.** *For every $k > 0$, the simulation closure of $\mathcal{C}^k$ cannot be defined by any existential formula in $\mathrm{L}_\mu[k-1]$.*

However, this left open the question whether a formula from $\mathrm{L}_\mu[k-1]$ which uses universal quantification may be equivalent to ψ^k. Due to our Preservation Theorem, we are now able to assert that this cannot be the case.

Theorem 6. *For every $k > 0$, the formula $\psi^k \in \mathrm{L}_\mu[k]$ defining the simulation closure of $\mathcal{C}^k$ is not equivalent to any formula in $\mathrm{L}_\mu[k-1]$.*

Proof. Let us assume that there exists a formula $\psi \in \mathrm{L}_\mu[k-1]$ equivalent to ψ^k. Since ψ defines the simulation closure of $\mathcal{C}^k$, a finite strongly connected structure,

we can apply Theorem 4 to conclude that there also exists a formula $\psi' \in \mathrm{L}_\mu[k-1]$ using only existential modalities which is equivalent to ψ^k. But this contradicts the Hierarchy Theorem 5 for the existential fragment. □

As a direct consequence, we can separate the expressive power of Parikh's Game Logic [16] and the μ-calculus, thus answering an open question posed by Pauly in [17]. Since Game Logic can be translated into the two variable fragment of L_μ, its expressive power is strictly subsumed already by $\mathrm{L}_\mu[3]$.

Corollary 3. *The modal μ-calculus is strictly more expressive than Game Logic interpreted over transition structures.*

Notice that the examples of strict formulae for $\mathrm{L}_\mu[k]$ given in [5] use a vocabulary consisting of k^2 actions. In a forthcoming paper [6], we provide examples of hard formulae over a fixed alphabet of only two actions for every level k.

References

1. A. Arnold, *The mu-calculus alternation-depth is strict on binary trees*, RAIRO Informatique Théorique et Applications, 33 (1999), pp. 329–339.
2. A. Arnold and D. Niwiński, *Rudiments of μ-calculus*, North Holland, 2001.
3. D. Berwanger, *Game logic is strong enough for parity games*, Studia Logica, 75 (2003), pp. 205–219. Special issue on Game Logic and Game Algebra edited by M. Pauly and R. Parikh.
4. D. Berwanger and E. Grädel, *Games and model checking for guarded logics*, in Proceedings of LPAR 2001, Lecture Notes in Computer Science Nr. 2250, Springer, 2001, pp. 70–84.
5. D. Berwanger, E. Grädel, and G. Lenzi, *On the variable hierarchy of the modal mu-calculus*, in Computer Science Logic, CSL 2002, J. Bradfield, ed., vol. 2471 of LNCS, Springer-Verlag, 2002, pp. 352–366.
6. D. Berwanger and G. Lenzi, *Robbers, guardians, and a few diamonds. Hard patterns in the μ-calculus variable hierarchy.* Submitted.
7. J. Bradfield, *The modal μ-calculus alternation hierarchy is strict*, Theoretical Computer Science, 195 (1998), pp. 133–153.
8. G. d'Agostino and M. Hollenberg, *Logical questions concerning the μ-calculus: interpolation, Lyndon, and Łos-Tarski*, Journal of Symbolic Logic, 65 (2000), pp. 310–332.
9. A. Emerson and C. Jutla, *Tree automata, mu-calculus and determinacy*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 368–377.
10. M. C. B. Hennessy and R. Milner, *On observing nondeterminism and concurrency*, in Automata, Languages and Programming, 7th Colloquium, J. W. de Bakker and J. van Leeuwen, eds., vol. 85 of LNCS, Springer-Verlag, 1980.
11. D. Janin and G. Lenzi, *On the logical definability of topologically closed recognizable languages of infinite trees*, Computing and Informatics, 21 (2002), pp. 185–203.
12. D. Kozen, *Results on the propositional μ-calculus*, Theoretical Computer Science, 27 (1983), pp. 333–354.
13. ———, *A finite model theorem for the propositional μ-calculus*, Studia Logica, 47 (1988), pp. 233–241.

14. O. Kupferman, M. Vardi, and P. Wolper, *An automata-theoretic approach to branching-time model checking*, Journal of the ACM, 47 (2000), pp. 312–360.
15. G. Lenzi, *A hierarchy theorem for the mu-calculus*, in Proceedings of the 23rd International Colloquium on Automata, Languages and Programming, ICALP '96, F. Meyer auf der Heide and B. Monien, eds., vol. 1099 of Lecture Notes in Computer Science, Springer-Verlag, July 1996, pp. 87–97.
16. R. Parikh, *The logic of games and its applications*, Annals of Discrete Mathematics, 24 (1985), pp. 111–140.
17. M. Pauly, *Logic for Social Software*, PhD thesis, University of Amsterdam, 2001.
18. C. Stirling, *Bisimulation, modal logic and model checking games*, Logic Journal of the IGPL, 7 (1999), pp. 103–124.

The Core of a Countably Categorical Structure

Manuel Bodirsky

Humboldt-Universität zu Berlin
bodirsky@informatik.hu-berlin.de

Abstract. A relational structure is a *core*, if all endomorphisms are embeddings. This notion is important for the classification of the computational complexity of constraint satisfaction problems. It is a fundamental fact that every finite structure S has a core, i.e., S has an endomorphism e such that the structure induced by $e(S)$ is a core; moreover, the core is unique up to isomorphism.

We prove that this result remains valid for ω-categorical structures, and prove that every ω-categorical structure has a core, which is unique up to isomorphism, and which is again ω-categorical. We thus reduced the classification of the complexity of constraint satisfaction problems with ω-categorical templates to the classifiaction of constraint satisfaction problems where the templates are ω-categorical cores. If Γ contains all primitive positive definable relations, then the core of Γ admits quantifier elimination. We discuss further consequences for constraint satisfaction with ω-categorical templates.

1 Introduction

The notion of a core has applications in the theory of constraint satisfaction. We therefore start with a brief introduction to constraint satisfaction; for formal definitions, see Section 2. Let Γ be a structure with a relational signature τ. The constraint satisfaction problem for the so-called *template* Γ is the following computational problem:

CSP(Γ)
INSTANCE: A finite structure S of the same relational signature τ as the template Γ.
QUESTION: Is there a homomorphism $h : S \to \Gamma$, i.e., a map h from S to Γ that preserves all relations from τ?

We want to stress that Γ is not part of the input. Each Γ defines a computational problem. Some of them might be equivalent, though. A classification into tractable and hard problems in this class is intensively studied for *finite* Γ, but still not complete. See [10,18,22,24,28], just to mention a few highlights on that subject.

The class of constraint satisfaction problems with an *infinite* template was not yet studied systematically. One example of a constraint satisfaction problem with

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 110–120, 2005.

an infinite template is CSP($(\mathbb{Q};<)$), where $(\mathbb{Q};<)$ is the countable dense linear order on the rational numbers. The binary relation for the linear order can be interpreted as a set of directed edges, and thus the corresponding computational problem is digraph-acyclicity, which can be solved in polynomial time.

It turns out that many other interesting computational problems can be formulated with templates that are *ω-categorical*. A structure is ω-categorical, if all countable models of its first-order theory are isomorphic to Γ. This is for instance the case for $(\mathbb{Q};<)$. On the one hand, ω-categoricity is a rather strong model-theoretic assumption on a relational structure, and many techniques for constraint satisfaction with finite templates extend to ω-categorical templates. On the other hand, the class of computational problems that can be formulated as a constraint satisfaction problem with such a template is large, as demonstrated by the following list of well-known computational problems.

- Allen's interval algebra, and all its fragments [4, 25, 27].
- Problems in phylogenetic analysis [20, 29].
- Tree description constraints from computational linguistics [8, 9, 16].
- All problems in monotone monadic SNP without inequality [7, 18].

The class *strictly* contains all constraint satisfaction problems with finite templates: the subclasses of problems mentioned above can not be formulated with finite templates.

Note that the image of a relational structure Γ under an endomorphism (i.e., a homomorphism from Γ to Γ) has the same constraint satisfaction problem as Γ. Sometimes one can simplify the formulation of a constraint satisfaction problem with template Γ by considering CSP(Γ'), where Γ' is an endomorphic image of Γ. This works particularly well for finite structures, where we have a canonical choice for Γ', namely the *core* of Γ. A finite relational structure A is called a *core* if every endomorphism of A is an automorphism of A; a core A is called *a core of* B if A is the image of an endomorphism of B. The following is well-known and easy to prove.

Theorem 1. *Every* finite *relational structure has a core, which is unique up to isomorphism.*

Therefore, we speak of *the* core of a finite relational structure A; see [21]. The countable dense linear order $(\mathbb{Q};<)$ has many endomorphisms that are not automorphisms. However, all endomorphisms are injective, and *strong*, i.e., they preserve not only the order $<$ but also its complement $\geq$. Various generalizations of the notion of a core for infinite structures were studied by Bauslaugh [5, 6]. He introduced the notation below. If there is a homomorphism from Γ to H, we also write $\Gamma \to H$.

- $I(\Gamma)$ holds if every endomorphism of Γ is an injection.
- $S(\Gamma)$ holds if every endomorphism of Γ is a surjection.
- $N(\Gamma)$ holds if every endomorphism of Γ is strong, i.e., preserve also the complements of the relations in Γ.
- $i(\Gamma)$ holds if there is a substructure $H \subseteq \Gamma$ s.t. $\Gamma \to H$ and $I(H)$.

– $s(\Gamma)$ holds if there is a substructure $H \subseteq \Gamma$ s.t. $\Gamma \to H$ and $S(H)$.
– $n(\Gamma)$ holds if there is a substructure $H \subseteq \Gamma$ s.t. $\Gamma \to H$ and $N(H)$.

Bauslaugh canonically defined combinations of the uppercase properties. For instance the property *ISN* of a finite structure A states that A is a core. For finite structures, *ISN* is equivalent to *IN, IS, I*, and to *S*. These properties all generalize the definition of a core for finite structures, and they are all inequivalent for infinite structures. For combinations of the lowercase properties we additionally require that the same subgraph H of Γ has the required properties. Finite structures A satisfy isn. Infinite structures in general satisfy none of the properties i, s, or n. For ω-categorical structures, we argue that the following definition is most useful.

Definition 1. *A (finite or infinite) structure Γ is a core, if all endomorphisms are injective and strong.*

Hence, the dense linear order of the rational numbers is a core. This definition corresponds to IN-cores in the terminology of Bauslaugh, and generalizes the definition for finite structures.

We will show that for every ω-categorical structure there is an endomorphism c such that the image of c is a core; in the terms above, we show that ω-categorical templates Γ satisfy $in(\Gamma)$. Moreover, we show that this core is unique up to isomorphism, and again ω-categorical. If we add all primitive positive formulas to the signature of an ω-categorical core, the resulting relational structure admits quantifier elimination. The implications of these results for constraint satisfaction are discussed in Section 5; in particular we generalize an important result of [11] for constraint satisfaction with finite templates to all ω-categorical templates.

Examples. To illustrate the concepts we have seen so far, we formulate several well-known computational problems as constraint satisfaction problems. With Theorem 3 in the next section it will be easy to check that the corresponding templates are all ω-categorical. Three more examples follow at the end of Section 3, since we need the concept of amalgamation to define them conveniently. In all these examples, it is fairly easy to check that the chosen template is a core.

Betweenness. An important NP-hard problem is Betweenness [19], since the hardness of many fragments of Allen's Interval Algebra [4, 25] is most easily checked by reduction from Betweenness. Given a finite set V, and a collection C of ordered triples (x, y, z) of distinct elements from V, the computational question is whether there is an injective function $f : V \to \{1, \ldots, |V|\}$ such that, for each $(a, b, c) \in V$, we have either $f(a) < f(b) < f(c)$ or $f(c) < f(b) < f(a)$. The formulation as a constraint satisfaction problem is straightforward, using the rational numbers as the base set.

Switching-Acyclicity. The following problem was studied in [9]. Given a digraph $D = (V; E)$, can we partition the vertices V into two parts, such that the

graph that arises from D by switching all arcs between the two parts is acyclic? To formulate this as a constraint satisfaction problem with an ω-categorical template, consider a dense subset X of $\mathbb{Q}$, and switch the order $<$ between the elements of X and $\mathbb{Q} - X$, and leave the edges within X and within $\mathbb{Q} - X$ unchanged. The resulting structure is called $S(2)$ and is isomorphic for all choices of dense sets X. The constraint satisfaction problem of $S(2)$ is the problem described above. For equivalent definitions of $S(2)$ and an hardness-proof of its constraint satisfaction problem, see [7, 9].

Partial tree descriptions. Our next example was studied in computational linguistics [16], and the first polynomial time algorithm can be found in [8]. Let D be a digraph with two types of arcs, called *ancestorship* and *non-ancestorship* arcs. The question is whether D is a consistent partial tree description, i.e., whether we can find a forest with oriented edges on the vertex set of D, such that for every ancestor arc in D there is a directed path in the forest, and for every non-ancestor arc there is no directed path in the forest. Here we choose the following *dense proper semilinear order* [2, 13, 17] as a template. The domain of the structure is the set Λ of all non-empty finite sequences $a = (q_0, q_1, \ldots, q_{n-1})$ of rational numbers. Let $a < b$ if either

- b is a proper initial subsequence of a, or
- $b = (q_0, \ldots, q_{n-1}, q_n)$ and $a = (q_0, \ldots, q_{n-1}, q'_n, q_{n+1}, \ldots, q_m)$, where the rational number q_n is smaller than q'_n.

The relation $<$ corresponds to ancestorship edges, this is, we write $a < b$ if b is an ancestor of a. The set of all ordered pairs of distinct points not in $<$, denoted by $\not\leq$, corresponds to the non-ancestorship edges. $\mathrm{CSP}((\Lambda; <, \not\leq))$ is the constraint satisfaction problem we are interested in.

Non-cores. Of course, there are plenty of ω-categorical structures that are not cores, for instance the Random graph $\mathbf{R}$ [12, 23], whose core is the complete graph K_ω on countably many vertices (the constraint satisfaction problem of $\mathbf{R}$ and K_ω is trivial). Another example is the structure $(\Lambda; <)$, i.e., the template for partial tree descriptions introduced above without the relation $\not\leq$. This structure contains an isomorphic copy of the core $(\mathbb{Q}; <)$, and there is an endomorphism from Λ to this copy.

2 Model-Theoretic Preliminaries

Let A and B be relational structures with the same relational signature τ. A mapping $f : A \to B$ is called a *homomorphism*, if for all relations $R \in \tau$ and $x_1, \ldots, x_n \in A$ the relation $R(f(x_1), \ldots, f(x_n))$ holds in B whenever $R(x_1, \ldots, x_n)$ holds in A. A homomorphism is called *strong*, if $R(x_1, \ldots, x_n)$ holds in A *if and only if* $R(f(x_1), \ldots, f(x_n))$ holds in B. A strong and injective homomorphism is also called an *embedding*. Homomorphisms that are not

embeddings are called *strict*. A homomorphism from A to A is called an *endomorphism* of A, and a bijective strong homomorphism from A to A is called an *automorphism* of A.

A formula ϕ is *primitive (primitive positive)*, if it is of the form $\exists \overline{x}.\psi_1 \wedge \cdots \wedge \psi_k$, where ψ_i are literals (atomic formulas). It is called *existential (existential positive)*, if it is of the form $\exists \overline{x}.\Psi$ where Ψ is quantifier-free (and negation-free). The strongest of these four syntactic restrictions, *primitive positivity*, is important for constraint satisfaction, since the expansion of a template with a primitive positive definable (short, *p.p.-definable*) relation does not change the complexity of the corresponding constraint satisfaction problem. This is an easy observation, see e.g. [24].

A structure Γ admits *quantifier elimination*, if every first-order formula has in Γ a quantifier-free definition. For example, consider modules, and add all p.p.-definable relations to the signature. The theorem of Baur and Monk says that the resulting structure admits quantifier elimination (see e.g. [23]). As we will see at the end of this section, cores behave similarly in this respect. It is sometimes possible to apply the following well-known theorem to eliminate *negations* in existential formulas.

Theorem 2. *Let T be a first-order theory such that every homomorphism between models of T is an embedding. Then every existential formula is equivalent to an existential positive formula with respect to T.*

For the proofs for the statements in this section we refer to the full version of this paper. If Γ is a structure with relational signature τ, then $Th(\Gamma)$ denotes the set of all first-order sentences (where the atomic formulas are built from the symbols in τ and equality) that hold in Γ. The following is an important consequence of Theorem 2.

Proposition 1. *Let Γ be a τ-structure with finitely many p.p.-definable relations of arity k, for all $k \geq 1$, and where τ contains a relation symbol for each of them. If all homomorphisms between models of* $\mathrm{Th}(\Gamma)$ *are embeddings, then Γ allows quantifier elimination.*

3 Countably Categorical Structures

Finite structures are up to isomorphism determined by their first-order theory. We can not expect this for infinite structures: by the theorem of Löwenheim-Skolem, every consistent theory with a model of cardinality λ has models of arbitrary cardinality $\geq \lambda$. However, it might still be the case that all models of a certain cardinality are isomorphic. If this is the case for the countable models, we call the theory *ω-categorical*. A countable structure is called *ω-categorical* (or *countably categorical*), if its first-order theory is ω-categorical. Throughout the paper we only consider relational and at most countable structures and signatures. Despite the powerful theorems quoted below, the class of ω-categorical structures remains somewhat mysterious, and all classification results require

some additional properties (stability in e.g. [26], or homogeneity in [15]). All notions that are used here are standard and can be found e.g. in [23].

Theorem 3 (Engeler, Ryll-Nardzewski, Svenonius). *The following properties of a structure Γ are equivalent:*

1. *the structure Γ is ω-categorical;*
2. *for each $n \geq 1$, there are finitely many orbits of n-tuples in the automorphism group $Aut(\Gamma)$ of Γ;*
3. *for each $n \geq 1$, there are finitely many inequivalent formulas with n free variables over Γ.*

Permutation groups with the second property in Theorem 3 are called *oligomorphic* [12]. We need yet another characterization of ω-categoricity, which is of a more combinatorial nature, and links the concept via Fraïssé's theorem to *homogeneity* and *amalgamation classes*.

A structure is *homogeneous* (sometimes also called *ultra-homogeneous*) if every isomorphism between finite substructures of Γ can be extended to an automorphism (in this paper, substructure always means *induced substructure*, as in [23]). Every ω-categorical structure can be expanded by first-order definable relations such that the resulting structure is homogeneous. An ω-categorical structure has quantifier elimination if and only if it is homogeneous (2.22 in [12]). An example of an ω-categorical structure that is not homogenous is $(\Lambda; <, \not\leq)$ [17]. For an example of a homogeneous structure that is not ω-categorical, consider the expansion of a countable structure Γ by unary singleton predicates for each element in Γ. This structure is homogeneous, since there are no distinct isomorphic substructures in Γ, and it is not ω-categorical, since the number of orbits in the automorphism group of Γ is infinite.

The next theorem asserts that a countable homogeneous structure is up to isomorphism characterized by its *age*, i.e., the set of its finite substructures. A class of finite relational structures $\mathcal{C}$ is an amalgamation class if $\mathcal{C}$ is nonempty, closed under isomorphisms and taking substructures, and has the *amalgamation property*. The amalgamation property says that for all $A, B_1, B_2 \in \mathcal{C}$ and embeddings $e_1 : A \to B_1$ and $e_2 : A \to B_2$ there exists $C \in \mathcal{C}$ and embeddings $f_1 : B_1 \to C$ and $f_2 : B_2 \to C$ such that $f_1 e_1 = f_2 e_2$.

Theorem 4 (Fraïssé). *A countable class $\mathcal{C}$ of finite relational structures with countable signature is the age of a countable homogeneous structure if and only if $\mathcal{C}$ is an amalgamation class. In this case the homogeneous structure is up to isomorphism unique and called the* Fraïssé-limit *of $\mathcal{C}$.*

The following templates of well-known constraint satisfaction problems are easily defined with amalgamation classes.

Triangle-Free Colorings. The class of all triangle-free graphs is an amalgamation class. Let us denote its Fraïssé-limit by $\not\triangle$. Clearly, CSP($\not\triangle$) is tractable; but it can not be formulated with a *finite* template. The structure $[\not\triangle, \not\triangle]$, i.e., the structure that consists out of two copies of $\not\triangle$, where all vertices between the two

copies are linked, has an interesting constraint satisfaction problem, which can be formulated as follows: Given a graph, can we partition its vertices into two parts such that both parts do not contain a triangle? This problem is a rather typical example from the class *monotone monadic SNP without inequality (MMSNP)*, a fragment of existential second-order logic introduced in [18] in the context of constraint satisfaction. A general result on so-called *G-free colorability* implies its NP-hardness [1].

Quartet Compatibility. The next example is an important structure in the theory of infinite permutation groups [12]. A *boron tree* is a finite tree in which all vertices have degree one (*hydrogen atoms*) or degree three (*boron atoms*). On the hydrogen atoms of a boron tree we can define a quaternary relation $xy|uv$ that holds when the paths joining x to y and u to v are disjoint. The class of all structures $\mathcal{D}$ with a quaternary relation that stem from a boron tree as defined above is an amalgamation class [2]. Let D be the Fraïssé-limit of $\mathcal{D}$. Then CSP(D) is a well-known NP-hard problem [29] that was independently studied in phylogenetic analysis (without any reference to constraint satisfaction), and is called *quartet-compatibility*: Given a collection C of quartets $xy|uv$ over a set X, is there some tree with leaf set X such that for each quadruple $xy|uv$ in C the paths from x to y and from u to v do not have common vertices?

Rooted Triple Consistency. The next problem is studied in phylogenetic analysis, again without notice that the problem can be stated as a constraint satisfaction problem. If we fix a point a in the previous structure D and consider the ternary relation ':' defined by $x : yz \Leftrightarrow ax|yz$, we again obtain an ω-categorical structure (this is a *C-set* in [2]). The age of this structure now contains the finite structures T that come from finite *rooted* trees, and the relation $x : yz$ says that the least common ancestor of y and z is strictly below the least common ancestor of x, y, and z in the tree T. The corresponding constraint satisfaction problem is known as the rooted triple consistency problem [29], and tractable. The first polynomial time algorithm for this problem goes back to [3], motivated by a question in database theory.

4 The Core of a Countably Categorical Structure

We define the notion of a core of a relational structure, and prove that every countably categorical structure has a core, which is again ω-categorical and unique up to isomorphism. Recall from Definition 1 that a relational structure Γ is a core, if every endomorphism of Γ is an embedding of Γ in Γ. In other words, cores do not have strict endomorphisms.[1]

Definition 2. *We say that a structure Γ' is a* core of Γ *if Γ' is a core and is induced by the image of an endomorphism of Γ.*

[1] A concept with a related flavour is the concept of a *simple model of a theory* defined in [23] and, slightly different, in [14].

We start with a proposition that states the existence of a 'youngest' endomorphic image of an ω-categorical structure. The proof employs a typical technique for ω-categorical structures.

Proposition 2. *Let Γ be an ω-categorical relational τ-structure. Then there exists an endomorphism c of Γ such that for every other endomorphism g, all finite substructures of $c(\Gamma)$ embed into $g(\Gamma)$. This is, there exists an endomorphic image of Γ of smallest age.*

Proof. Let $\mathcal{S}$ be the set of all finite τ-structures S such that there is an endomorphism g of Γ such that S is not a substructure of $g(\Gamma)$. We have to show that there is an endomorphism c such that $c(\Gamma)$ does not contain any substructure from $\mathcal{S}$. For the construction of c we consider the following tree. Let $a_1, a_2, \ldots$ be an enumeration of Γ. The vertices on level n of the tree are equivalence classes of *good* homomorphisms from $\{a_1, \ldots, a_n\}$ to Γ. A homomorphism h is good, if $h(\{a_1, \ldots, a_n\})$ does not contain any substructure from $\mathcal{S}$. Two homomorphisms g_1 and g_2 are equivalent, if there exists an automorphism α of Γ such that $g_1 = \alpha(g_2)$. Clearly, if a homomorphism is good, then all equivalent homomorphisms are also good. A vertex u on level $n+1$ in the tree is connected to a vertex v on level n, if some homomorphism from u is the restriction of some homomorphism from v. Because of ω-categoricity, the tree is finitely branching. We want to show that the tree has vertices on each level n, and iteratively construct a sequence $h_1, h_2, \ldots, h_k$ of homomorphisms from $\{a_1, \ldots, a_n\}$ to Γ, where the last endomorphism h_k induces a good homomorphism. Initially, if the structure induced by $\{a_1, \ldots, a_n\}$ does not have a substructure from $\mathcal{S}$, we can choose the identity as a good homomorphism. Otherwise, there is a substructure $S \in \mathcal{S}$ on the elements $\{a_1, \ldots, a_n\}$ and an endomorphism e such that $e(\Gamma)$ does not contain S. Hence, $h_1 := e|_{\{a_1,\ldots,a_n\}}$ is a strict homomorphism.

In step i, if the structure induced by $h_i(\{a_1, \ldots, a_n\})$ does not have a substructure from $\mathcal{S}$, then h_i is a good homomorphism, and we are again done. Otherwise there is an endomorphism e of Γ and a structure $S \in \mathcal{S}$ on elements from $h_i(\{a_1, \ldots, a_n\})$, such that $e(\Gamma)$ does not contain S. We can then define a strict homomorphism $h_{i+1} : \{a_1, \ldots, a_n\} \to \Gamma$ by $h_{i+1}(x) := e(h_i(x))$. Since in the sequence $h_1(\{a_1, \ldots, a_n\})$, $h_2(\{a_1, \ldots, a_n\}), \ldots$ either the number of vertices decreases or the number of tuples in relations increases, and since Γ is ω-categorical, the sequence has to be finite. Hence, there exists a good homomorphism from $\{a_1, \ldots, a_n\}$ to Γ, for all $n \geq 0$. By König's tree lemma, there exists an infinite path in the tree. Since adjacency in the tree was defined by restriction between homomorphisms, this path defines an endomorphism c of Γ. By construction, $c(\Gamma)$ does not contain a substructure from $\mathcal{S}$. □

As a direct consequence all cores of Γ have the same age as $c(\Gamma)$. The following theorem is one of the main results of this paper; again, the proofs can be found in the full version of the paper, available at `www.informatik.hu-berlin.de/~bodirsky`.

Theorem 5. *If Γ is ω-categorical and contains all primitive positive definable relations, then it has a homogeneous and ω-categorical core, which is unique up to isomorphism.*

The idea of the proof is to show that every homomorphism between two models Γ_1 and Γ_2 of $\mathrm{Th}(c(\Gamma))$ is strong and injective.

Corollary 1. *Every countably categorical structure has a core, which is again ω-categorical and unique up to isomorphism.*

5 Adding Constants to the Signature

Why is it useful to look at the constraint satisfaction problem of the core of Γ, instead of the constraint satisfaction problem of Γ itself? One reason will be given in this section. First a useful fact for ω-categorical cores.

Proposition 3. *Let Γ be an ω-categorical core, and let R be an orbit of k-tuples in $Aut(\Gamma)$. Then R has a primitive positive definition in Γ.*

Proof. Let Γ' be the expansion of Γ by all p.p.-definable relations. Since R is an orbit also in Γ', all k-tuples in R are isomorphic to some substructure S of Γ'. By Theorem 5, Γ' is homogeneous, and *all* k-tuples in Γ' that are isomorphic to S are contained in R. Thus, R has a definition as a conjunction φ of atomic formulas. We replace all relation symbols in φ that are contained in the signature of Γ', but not in the signature of Γ, by their primitive positive definition. The resulting formula is equivalent to a primitive positive definition of R in Γ. □

One of the most often cited results in [11] is that if Γ is a finite core, adding a singleton-relation does not increase the complexity of the constraint satisfaction problem. In this section we will show that the same holds for constraint satisfaction problems where the template is an ω-categorical core.

Theorem 6. *Let Γ be an ω-categorical core, and Γ' be the expansion of Γ by a unary singleton relations $C = \{c\}$. If CSP(Γ) is tractable, then so is CSP(Γ'). (If CSP(Γ') is NP-hard, then so is CSP(Γ).)*

Proof. We show how to solve $\mathrm{CSP}(\Gamma')$ in polynomial time, under the assumption that $\mathrm{CSP}(\Gamma)$ can be solved in polynomial time. Let S' be an instance of $\mathrm{CSP}(\Gamma')$. Let P be the orbit of c in the automorphism group of Γ. By Proposition 3, P is p.p.-definable in Γ. Thus we can assume without loss of generality that Γ and Γ' contain the relation P. Replace all occurrences of the relation C in S' by the relation P. Solve the resulting instance S of $\mathrm{CSP}(\Gamma)$; by assumption this is possible in polynomial time. If S is not satisfiable, then in particular S' could not have been satisfiable. On the other hand, if there is a homomorphism h from S to Γ, we claim that there is a homomorphism from S' to Γ'. Since P is the orbit of the element c, there is an automorphism a of Γ such that $h \circ a$ is a solution of the instance S' of $\mathrm{CSP}(\Gamma')$. □

6 Discussion

We showed that every ω-categorical structure Γ has a core, which is unique up to isomorphism. Since the core of Γ has the same constraint satisfaction

problem as Γ, and since the core is again ω-categorical, we reduced the classification of constraint satisfaction with ω-categorical templates to the classification of constraint satisfaction problems where the template is an ω-categorical core.

The complexity of a constraint satisfaction problem does not change if we expand the template by a p.p.-definable relation. The following result resembles the theorem of Baur and Monk, and is of theoretical interest in constraint satisfaction: if we expand the core by all p.p.-definable relations, the resulting structure admits quantifier elimination. Finally we proved that a result known for constraint satisfaction with finite templates [11] remains valid for ω-categorical structures: if we expand an ω-categorical core by a singleton relation then the resulting constraint satisfaction problem has the same complexity.

References

1. D. Achlioptas. The complexity of G-free colourability. *Discrete Mathematics*, 165:21–30, 1997.
2. S. Adeleke and P. M. Neumann. Structure of partially ordered sets with transitive automorphism groups. *AMS Memoir*, 57(334), 1985.
3. A. Aho, Y. Sagiv, T. Szymanski, and J. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.
4. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
5. B. Bauslaugh. Core-like properties of infinite graphs and structures. *Disc. Math.*, 138(1):101–111, 1995.
6. B. Bauslaugh. Cores and compactness of infinite directed graphs. *Journal of Combinatorial Theory, Series B*, 68(2):255–276, 1996.
7. M. Bodirsky. Constraint satisfaction with infinite domains. Dissertation an der Humboldt-Universität zu Berlin, 2004.
8. M. Bodirsky and M. Kutz. Pure dominance constraints. In *Proceedings of STACS'02*, pages 287–298, 2002.
9. M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. In *Proceedings of CSL'03*, pages 44–57, Vienna, 2003.
10. A. Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of LICS'03*, pages 321–330, 2003.
11. A. Bulatov, A. Krokhin, and P. G. Jeavons. Classifying the complexity of constraints using finite algebras. *Submitted*, 2003.
12. P. J. Cameron. *Oligomorphic Permutation Groups*. Cambridge Univ. Press, 1990.
13. P. J. Cameron. The random graph. *R. L. Graham and J. Nešetřil, Editors, The Mathematics of Paul Erdõs*, 1996.
14. Chang and Keisler. *Model theory*. Princeton University Press, 1977.
15. G. Cherlin. The classification of countable homogeneous directed graphs and countable homogeneous n-tournaments. *AMS Memoir*, 131(621), January 1998.
16. T. Cornell. On determining the consistency of partial descriptions of trees. In *Proceedings of the ACL*, pages 163–170, 1994.
17. M. Droste. Structure of partially ordered sets with transitive automorphism groups. *AMS Memoir*, 57(334), 1985.

18. T. Feder and M. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1999.
19. M. Garey and D. Johnson. *A guide to NP-completeness*. CSLI Press, 1978.
20. D. Gusfield. *Algorithms on strings, trees, and sequences. Computer Science and Computational Biology*. Cambridge University Press, New York, 1997.
21. P. Hell and J. Nesetril. The core of a graph. *Discrete Math.*, 109:117–126, 1992.
22. P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990.
23. W. Hodges. *A shorter model theory*. Cambridge University Press, 1997.
24. P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *JACM*, 44(4):527–548, 1997.
25. P. Jeavons, P. Jonsson, and A. A. Krokhin. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *JACM*, 50(5):591–640, 2003.
26. A. H. Lachlan. Stable finitely homogeneous structures: A survey. In *Algebraic Model Theory, NATO ASI Series*, volume 496, pages 145–159, 1996.
27. B. Nebel and H.-J. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *JACM*, 42(1):43–66, 1995.
28. T. J. Schaeffer. The complexity of satisfiability problems. In *Proceedings of STOC'78*, pages 216–226, 1978.
29. M. Steel. The complexity of reconstructing trees from qualitative charaters and subtrees. *Journal of Classification*, 9:91–116, 1992.

How Common Can Be Universality for Cellular Automata?

Guillaume Theyssier

LIP (UMR CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1),
École Normale Supérieure de Lyon,
46 allée d'Italie 69364 LYON cedex 07 France
Guillaume.Theyssier@ens-lyon.fr

Abstract. We address the problem of the density of intrinsically universal cellular automata among cellular automata or a subclass of cellular automata. We show that captive cellular automata are almost all intrinsically universal. We show however that intrinsic universality is undecidable for captive cellular automata. Finally, we show that almost all cellular automata have no non-trivial sub-automaton.

Keywords: cellular automata, universality, zero-one law.

Cellular automata are simple discrete dynamical systems involving full uniformity and perfect synchronism. They are capable of producing very complex behaviours despite their apparent simplicity and therefore constitute an idealistic model to study the paradigm of complex systems. Besides its ability to capture any sequential computations, the model of cellular automata possesses a natural notion of intrinsic universality. A cellular automaton is intrinsically universal if it is able to directly simulate any other cellular automaton. There is no general definition of what is an acceptable simulation but in [1] a natural and rather minimal class of acceptable simulation is introduced and give rise to the formal notion of intrinsic universality adopted in this paper.

Since the very beginning of cellular automata theory great efforts have been devoted to the design of particular cellular automata having some desired property. The property of being intrinsically universal was of course especially studied and the quest for the smallest intrinsically universal cellular automaton has now almost reached the limits (closed in dimension 2 and higher by [2] and reduced to a 4 states gap in dimension 1 by [3]). However, these tricky constructions only give results concerning *sufficient* conditions and do not respond to the problem of how strong is the intrinsic universality requirement for a cellular automaton in general, or, said differently, how many different ways there are to achieve intrinsic universality. Unfortunately, this converse problem reveals to be difficult since the set of non intrinsically universal cellular automata is not recursively enumerable (see [1]) whereas the set of intrinsically universal one is.

In the present paper we tackle this problem using a different point of view: we study *density* of intrinsically universal cellular automata. Our main result is that

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 121–132, 2005.

a simple hypothesis on the local transition map gives rise to a class of cellular automata (namely *captive* cellular automata) for which density follows a zero-one law over an interesting class of properties. We then show that intrinsic universality belongs to that class and, using the zero-one law, that almost all captive cellular automata are intrinsically universal. We show however that the set of non intrinsically universal captive cellular automata is not recursively enumerable. Back to the general case, we show that almost all cellular automata lack of any non-trivial local structure making a strong difference with the captive case.

1 Formal Framework

Although many results extend to higher dimensions, we will only consider one-dimensional CA. Besides, this paper is not concerned with comparisons between different shapes of neighbourhood and we consider only VON NEUMANN-like neighbourhood (connected and centred). Formally a CA is triple $\mathcal{A} = (A, r, f_{\mathcal{A}})$ where A is a finite set of *states*, r is a positive integer (the *radius* of the neighbourhood) and $\mathcal{A}$ is a map from A^{2r+1} to A. Configurations are maps from $\mathbb{Z}$ to A giving each cell a particular state. The local transition function $f_{\mathcal{A}}$ induces a global evolution rule on configurations denoted $\mathcal{A}$ and defined as follows: $\forall c \in A^{\mathbb{Z}}$, $\mathcal{A}(c)$ is such that $\forall i \in \mathbb{Z}$, $\big(\mathcal{A}(c)\big)(i) = f_{\mathcal{A}}\big(c(i-r), c(i-r+1), \ldots, c(i+r)\big)$. In the sequel, when considering a CA $\mathcal{A}$ (resp. $\mathcal{B}$), we implicitly refer to the triple $(A, r_{\mathcal{A}}, \mathcal{A})$ (resp. $(B, r_{\mathcal{B}}, \mathcal{B})$) where the same symbol $\mathcal{A}$ (resp. $\mathcal{B}$) denotes both the local and the global map.

Local maps in CA are arbitrary, but in this paper we pay a special attention to a particular regularity they may possess which is captured by the notion of *sub-automaton*. Up to renaming, a *sub-automaton* of a CA $\mathcal{A}$ is a subset of the states set which is stable under iterations of $\mathcal{A}$. Formally the sub-automaton relation, denoted by $\sqsubseteq$, is defined as follows.

Definition 1. *$\mathcal{A} \sqsubseteq \mathcal{B}$ if there is an injective map i from A to B such that $\bar{i} \circ \mathcal{A} = \mathcal{B} \circ \bar{i}$, where $\bar{i} : A^{\mathbb{Z}} \to B^{\mathbb{Z}}$ denotes the uniform extension of i.*

When $X \subseteq A$ is stable for $\mathcal{A}$ ($\mathcal{A}(X^{\mathbb{Z}}) \subseteq X^{\mathbb{Z}}$), we denote by $\mathcal{A}_X$ the restriction of $\mathcal{A}$ to X (then $\mathcal{A}_X \sqsubseteq \mathcal{A}$). Besides, when $|A| = |B|$, $\mathcal{A} \sqsubseteq \mathcal{B}$ implies $\mathcal{B} \sqsubseteq \mathcal{A}$ and $\mathcal{A}$ is isomorphic to $\mathcal{B}$ what is denoted by $\mathcal{A} \sim \mathcal{B}$.

Intrinsic universality we now define relies on a formal notion of direct simulation between CA. A restricted version of this notion was first introduced by J. MAZOYER and I. RAPAPORT in [4] and N. OLLINGER extended it in [5]. $\mathcal{A}$ can simulate $\mathcal{B}$ (denoted by $\mathcal{A} \preceq \mathcal{B}$) if, up to some regular spatio-temporal transformations, $\mathcal{A}$ is a sub-automaton of $\mathcal{B}$. Transformations considered here are very simple: they allow grouping several cells in one block and running several steps of a CA in one. Formally, for any finite set A and any $m \in \mathbb{N}$ ($m \neq 0$), let $o^m : A^{\mathbb{Z}} \to (A^m)^{\mathbb{Z}}$ be the map such that

$$\forall c \in A^{\mathbb{Z}}, \forall z \in \mathbb{Z} : \big(o^m(c)\big)(z) = \big(c(mz), c(mz+1), \ldots, c(m(z+1)-1)\big).$$

Then, denoting the CA $o^m \circ \mathcal{A}^n \circ (o^m)^{-1}$ (with states set A^m) by $\mathcal{A}^{<m,n>}$, the relation $\preceq$ is defined as follows:

$$\mathcal{A} \preceq \mathcal{B} \Leftrightarrow \exists m_a, m_b, n_a, n_b : \mathcal{A}^{<m_a,n_a>} \sqsubseteq \mathcal{B}^{<m_b,n_b>}.$$

$\preceq$ defines a quasi-order on the set of CA (see [5]) and naturally induces an equivalence relation denoted by $\simeq$ and an order on equivalence classes of $\simeq$. In addition to being natural for the model of CA, this simulation relation nicely captures the examples of intrinsically universal CA already present in the literature (see [2, 6]).

Definition 2. *$\mathcal{A}$ is intrinsically universal if* $\forall \mathcal{B}, \exists m, n : \mathcal{B} \sqsubseteq \mathcal{A}^{<m,n>}$.

An important fact is that the set of intrinsically universal CA is exactly the maximal class of $\simeq$ (see [5] for a detailed proof) and we will use alternatively this characterisation and the definition above.

A *property* $\mathcal{P}$ is a set of CA. $\mathcal{A}$ has the property $\mathcal{P}$ if $\mathcal{A} \in \mathcal{P}$. A property $\mathcal{P}$ is said to be *increasing* if: $\forall \mathcal{A}, \forall \mathcal{B}$, $\mathcal{A} \sqsubseteq \mathcal{B}$ implies $\mathcal{A} \in \mathcal{P} \Rightarrow \mathcal{B} \in \mathcal{P}$. Similarly, $\mathcal{P}$ is said to be decreasing if: $\forall \mathcal{A}, \forall \mathcal{B}$, $\mathcal{A} \sqsubseteq \mathcal{B}$ implies $\mathcal{B} \in \mathcal{P} \Rightarrow \mathcal{A} \in \mathcal{P}$. Notice that since the $\sim$ relation is included in $\sqsubseteq$, increasing (or decreasing) properties are closed under renaming of states—a natural requirement when studying CA. We will specifically concentrate on monotonic properties in section 2. For now, just notice that intrinsic universality is an increasing property.

To measure how common a property is among CA, we consider its density using the following canonical enumeration of CA: a radius r is fixed and we let the number of states grow. To avoid irrelevant consideration of states renaming we consider only CA whose states are integers. Precisely, A_n denotes the set of CA of radius r with states set $\{1, \ldots, n\}$ and the density of properties is defined as follows.

Definition 3. *The* density *of property $\mathcal{P}$ is* $\mu(\mathcal{P}) = \lim_{n\to\infty} \frac{|A_n \cap \mathcal{P}|}{|A_n|}$ *when the limit exists.*

To end this section, we give some useful notations. If $a \in A$ then $\overline{a}$ denotes the configuration of $A^{\mathbb{Z}}$ made solely of a. Similarly, if $f : A \to A$, $\overline{f}$ denotes its uniform extension to configurations of $A^{\mathbb{Z}}$, and f^k its extension to A^k ($f^k(a_1 \cdots a_k) = f(a_1) \cdots f(a_k)$). If c is a configuration, $L(c)$ denotes the set of words appearing in c. Finally, if w is a word, $\Sigma(w)$ denotes the set of letters appearing in w.

2 A Class of Cellular Automata Inducing a Zero-One Law for Monotonic Properties

In this section we consider a sub-class of CA (namely *captive* cellular automata) which was first introduced in [7]. Captive cellular automata are CA such that any subset of the states set is stable (*i.e.* induces a sub-automaton). We insist

that this class does not rely on any structural assumption on the states set and that it is characterised a property of the local transition map.

Definition 4. *$\mathcal{A}$ of radius r is a* captive *cellular automaton (*CCA *for short) if $\forall u \in A^{2r+1}$ we have $\mathcal{A}(u) \in \Sigma(u)$.*

We address the problem of the density of universality among CA from that class. Formally, if C_n denotes the set of CCA on states set $\{1,\dots,n\}$, the density of a property $\mathcal{P}$ among CCA is $\mu_C(\mathcal{P}) = \lim_{n\to\infty} \frac{|C_n \cap \mathcal{P}|}{|C_n|}$ when the limit exists.

Quite surprisingly, the structure of CCA allows to globally solve the problem of density for any monotonic property.

Lemma 1. *For any $\mathcal{B} \in C_n$ ($n \geq 2$), there exists a rational $\lambda_{\mathcal{B}} \in]0,1[$ such that for all $m \geq n$ and $X = \{a_1,\dots,a_n\} \subseteq \{1,\dots,m\}$, we have :*

$$\frac{\left|\{\mathcal{A} \in C_m : \mathcal{A}_X \sim \mathcal{B}\}\right|}{|C_m|} = \lambda_{\mathcal{B}}.$$

Proof. Let $m \geq n$ be fixed and consider X a subset of $\{1,\dots,m\}$ of size n. The equivalence relation $\equiv_X$ such that $\mathcal{A} \equiv_X \mathcal{B} \Leftrightarrow \mathcal{A}_X = \mathcal{B}_X$ is well-defined on C_m since the set X always induces a sub-automaton for any CCA. It is clear that $\equiv_X$ has exactly $|C_n|$ equivalence classes (independently of X), each of the same size $\frac{|C_m|}{n^{n^{2r+1}}}$. Besides, $\{\mathcal{A} : \mathcal{A}_X \sim \mathcal{B}\}$ is the union of a number b of classes of $\equiv_X$ depending only on $\mathcal{B}$. Therefore,

$$\frac{\left|\{\mathcal{A} \in C_m : \mathcal{A}_X \sim \mathcal{B}\}\right|}{|C_m|} = \frac{b}{|C_n|}$$

and the lemma follows. □

Theorem 1. *For any monotonic property $\mathcal{P}$ which is non-trivial in C, we have:*

- *if $\mathcal{P}$ is decreasing in C then $\mu_C(\mathcal{P}) = 0$;*
- *if $\mathcal{P}$ is increasing in C then $\mu_C(\mathcal{P}) = 1$.*

Proof. First suppose $\mathcal{P}$ is non-trivial and decreasing. There must therefore be some $\mathcal{B} \in C_n \setminus \mathcal{P}$ for some $n \in \mathbb{N}$. Now for $m \in \mathbb{N}$ let $m = kn + r$ be the Euclidean division of m by n and for $1 \leq i \leq k$ let $X_i = \{(i-1)n+1,\dots,in\}$. Then we have:

$$\mathcal{P} \cap C_m \subseteq \bigcap_{1\leq i\leq k} \{\mathcal{A} \in C_m : \mathcal{A}_{X_i} \not\sim \mathcal{B}\}$$

because $\mathcal{A} \in \mathcal{P}$ implies $\forall i, 1 \leq i \leq k : \mathcal{A}_{X_i} \in \mathcal{P}$. Now since the sets X_i are pairwise disjoint, the events "$\mathcal{A}_{X_i} \not\sim \mathcal{B}$" are pairwise independent. Hence, expressing the set inclusion above in terms of probabilities we get:

$$\frac{\left|\mathcal{P} \cap C_m\right|}{|C_m|} \leq \prod_{1\leq i\leq k} \frac{\left|\{\mathcal{A} \in C_m : \mathcal{A}_{X_i} \not\sim \mathcal{B}\}\right|}{|C_m|} = (1-\lambda_{\mathcal{B}})^k,$$

the right-hand equality being derived from lemma 1. Finally, taking the limit when $m \to \infty$ for both sides, we conclude: $\mu_C(\mathcal{P}) \leq \lim_{k\to\infty}(1-\lambda_{\mathcal{B}})^k = 0$.

Now suppose $\mathcal{P}$ is a non-trivial increasing property. Then $\neg\mathcal{P}$ is a non-trivial decreasing property. From what we have shown before $\mu_C(\neg\mathcal{P}) = 0$. Thus $\mu_C(\mathcal{P}) = 1$. □

The first immediate implication of theorem 1 is of dynamical nature (see [8] for a definition of expansivity).

Corollary 1. *For any fixed radius, almost no* CCA *is injective, or expansive, or surjective.*

Proof. Since the surjectivity property is obviously non-trivial for CCA, it is sufficient to show that it is decreasing and applying theorem 1 we get that almost no CCA is surjective. The fact that surjectivity is decreasing comes directly from theorem 5.9 of [9] which states that $\mathcal{A}$ is surjective if and only if the number of preimages of any word under $\mathcal{A}$ is uniformly bounded (independently of the word).

The facts that both injectivity and expansivity implies surjectivity are classical results (see [9]), but we insist that injectivity and expansivity are also decreasing non-trivial properties. □

In the case of CCA, the answer to the central question addressed in this paper is now obtained as a direct corollary of theorem 1.

Corollary 2. *There exists an integer r_0 such that for any fixed radius $r \geq r_0$, almost all* CCA *are intrinsically universal:* $\mu_C\big(\{\mathcal{A} : \forall\mathcal{B}, \mathcal{B} \preceq \mathcal{A}\}\big) = 1$.

Proof. The property of being intrinsically universal is obviously increasing so it is sufficient to show that it is non-trivial and the result follows from theorem 1. The existence of intrinsically universal CCA was first pointed out in [7]. Definition 5 and lemma 2 show that there is an intrinsically universal CCA of radius 7 and it is not difficult to tune the construction to lower its radius down to 5. □

Although almost all CCA are intrinsically universal as shown above, we are going to show that the problem of whether a given CCA is intrinsically universal or not is undecidable. This fact may seem scheming compared with the ubiquity of universality in CCA. But, overall, it has a noticeable structural consequence on the class CCA concerning the limit between universality and non-universality as pointed out by corollary 3.

The proof is a reduction from the same decision problem with any CA as input. It essentially relies on the transformation τ (given hereafter). We insist that algorithmic constructions on captive cellular automata are non-classical and involve new construction techniques because no Cartesian product can generally be used and every state which eventually appears at a position must already be present locally—thus a fixed radius implies a limited number of states being potentially used at each time whatever the states set is.

We now give the construction τ. It transforms a CA $\mathcal{A}$ into a CCA $\tau(\mathcal{A})$ simulating it. As usual, the simulation occurs on a particular set of "legal" configurations. On such configurations each cell of $\mathcal{A}$ is directly simulated by some *data cell* of $\tau(\mathcal{A})$ surrounded by several *control cells*. The main idea is as follows. For a *data cell* to change its state we must guarantee that its future state already appears in its neighbourhood. For that purpose, *signal cells* placed regularly along the line take periodically each state of A (thanks to a shift behaviour) and thus eventually allow the transition of their neighbouring *data cell*. The construction uses 3 other types of *control cells*: 2 are used to ensure the global synchronism of the simulation—which is the difficult part—and 1 is used to propagate encoding errors—a feature essential for the correctness of the reduction. The synchronism is controlled by *offset cells* and *memory cells*. The configuration formed by successive *signal cells* is spatially periodic and consist in an alternation of letters of A and offset indicators. Offset indicators are placed in such a way that all *offset cell* are "aligned" with their corresponding offset indicator at the same time. Finally, the *memory cells* are used to keep the result of transitions—which occur asynchronously—until *offsets cells* are "aligned" with their indicators. Each time this "alignment" occurs *data cells* are updated with saved transition results and *memory cells* are cleaned up. For sake of simplicity we only give the explicit construction of τ on CA with radius 1 but it is straightforward to extend it to any radius.

Definition 5. *Let $\mathcal{A}$ be a* CA *(supposed of radius* 1 *in the present definition). Let $O = \{o_0, \dots, o_{n-1}\}$ (with $n = |A|$) and $\{\kappa\}$ be sets of states disjoint with A. Denote by $W_{\mathcal{A}}$ the set of words of the form $O \cdot (A \cup O) \cdot \{\kappa\} \cdot (A \cup O) \cdot (A \cup O) \cdot A$. Let $C_{\mathcal{A}}$ be the set of configurations which are a bi-infinite concatenation of words of $W_{\mathcal{A}}$. Finally let $K_S = \{a_j o_{j+1 \bmod n}, 0 \le j \le n-1\} \cup \{o_j a_j, 0 \le j \le n-1\}$ and $K_O = \{o_j o_{j+1 \bmod n}, 0 \le j \le n-1\}$. $\tau(\mathcal{A})$ is the* CCA *of radius* 7 *and state set $A_\tau = A \cup \{\kappa\} \cup O$ defined as follows:*

1. *for any* offset *states $o, o', o'' \in O$,* signal *states $s_1, s_1', s_1'', s_2, s_2', s_2'' \in A \cup O$,* memory *states $m, m', m'' \in A \cup O$ and* data *states $d, d', d'' \in A$,*

u	$\tau(\mathcal{A})(u)$
$d''\, o\, s_1\, \kappa\, s_2\, m\, d\, \boxed{o'}\, s_1'\, \kappa\, s_2'\, m'\, d'\, o''\, s_1''$	$\begin{cases} o' & \text{if } oo' \in K_O \\ \kappa & \text{otherwise,} \end{cases}$
$o\, s_1\, \kappa\, s_2\, m\, d\, o'\, \boxed{s_1'}\, \kappa\, s_2'\, m'\, d'\, o''\, s_1''\, \kappa$	$\begin{cases} s_2' & \text{if } s_1's_2' \in K_S \\ \kappa & \text{otherwise,} \end{cases}$
$s_1\, \kappa\, s_2\, m\, d\, o'\, s_1'\, \boxed{\kappa}\, s_2'\, m'\, d'\, o''\, s_1''\, \kappa\, s_2''$	κ
$\kappa\, s_2\, m\, d\, o'\, s_1'\, \kappa\, \boxed{s_2'}\, m'\, d'\, o''\, s_1''\, \kappa\, s_2''\, m''$	$\begin{cases} s_1'' & \text{if } s_2's_1'' \in K_S \\ \kappa & \text{otherwise,} \end{cases}$
$s_2\, m\, d\, o'\, s_1'\, \kappa\, s_2'\, \boxed{m'}\, d'\, o''\, s_1''\, \kappa\, s_2''\, m''\, d''$	$\begin{cases} s_1' & \text{if } s_1' \in \{\mathcal{A}(dd'd''), o'\}, \\ m' & \text{otherwise,} \end{cases}$
$m\, d\, o'\, s_1'\, \kappa\, s_2'\, m'\, \boxed{d'}\, o''\, s_1''\, \kappa\, s_2''\, m''\, d''\, o$	$\begin{cases} m' & \text{if } s_1' = o' \\ d' & \text{otherwise,} \end{cases}$

2. *for any* $u \in A_\tau^{15} \setminus L(C_\mathcal{A})$, $\tau(\mathcal{A})(u) = \begin{cases} u_8 & \text{if } \kappa \notin \Sigma(u), \\ \kappa & \text{if } \kappa \in \Sigma(u) \end{cases}$

The 6 cases in the first part of the definition above are mutually exclusive because of a different position of state κ in u. Actually, in a configuration of $C_\mathcal{A}$, the type of a cell and the way it behaves is determined by its distance to the closest κ on its left, precisely: 3 for data cells, 2 for memory cells, 1 and 5 for signal cells and 4 for offset cells. Besides notice that for any configuration $c \in C_\mathcal{A}$, $\tau(\mathcal{A})$ checks whether

- the configuration formed by the successive offset cells in c, the *offset configuration* of c, is periodic of period $o_0 \dots o_{n-1}$;
- the configuration formed by the successive signal cells in c, the *signal configuration* of c, is periodic of period $o_0 a_0 o_1 a_1 \dots o_{n-1} a_{n-1}$.

Such configurations are characterised by the set of words of length 2 they contain (K_O and K_S respectively) and thus the checks can be done locally (line 1, 2 and 4 of the first point of definition 5). In the following, we will denote by $\Gamma_\mathcal{A}$ the set of configurations from $C_\mathcal{A}$ whose offset configuration and signal configuration are periodic with the respective periods given above. Informally, $\Gamma_\mathcal{A}$ is the set of "legal" configurations and it is straightforward to verify that $\Gamma_\mathcal{A}$ is stable under $\tau(\mathcal{A})$. The following lemma shows that $\tau(\mathcal{A})$ can simulate $\mathcal{A}$ on such configurations.

Lemma 2. *For any* CA $\mathcal{A}$, $\mathcal{A} \preceq \tau(\mathcal{A})$.

Proof. We show that $\mathcal{A}^{<n,1>} \sqsubseteq \tau(\mathcal{A})^{<6n,2n>}$ where $n = |A|$. Adopting the notations of definition 5, denote by ψ the following map from $A \times \{0, \dots, n-1\}$ to A_τ^6:

$$(\alpha, j) \mapsto o_j a_j \kappa o_{j+1 \bmod n} o_j a.$$

Then we define an injection Υ from A^n to A_τ^{6n} as follows:

$$\Upsilon(\alpha_0, \dots, \alpha_{n-1}) = \psi(\alpha_0, 0) \dots \psi(\alpha_j, j) \dots \psi(\alpha_{n-1}, n-1).$$

Now consider the set E of configurations of the form $\Upsilon(A^n)^\mathbb{Z}$. E is precisely the set of "legal" configurations ($E \subseteq \Gamma_\mathcal{A}$) which have just been "synchronised" (copy of memory cells to data cells and cleaning of memory cells). It is easy to verify that for $c \in E$ we have $\tau(\mathcal{A})^{<6n,2n>}(c) \in E$ (the spatial period of the signal configuration of c has length $2n$). Finally, $\tau(\mathcal{A})$ simulate 1 iteration of $\mathcal{A}$ every $2n$ iterations through the encoding Υ, precisely:

$$\begin{array}{ccc} (A^n)^\mathbb{Z} & \xrightarrow{\overline{\Upsilon}} & E \\ \downarrow{\scriptstyle \mathcal{A}^{<n,1>}} & & \downarrow{\scriptstyle \tau(\mathcal{A})^{<6n,2n>}} \\ (A^n)^\mathbb{Z} & \xrightarrow{\overline{\Upsilon}} & E \end{array}$$

To see this, first notice that starting from $c \in E$ data cells remain in the same state during $2n-1$ steps until they take the state of their neighbouring memory

cell at step $2n$. Second, memory cells are initially in a state from O and they wait for a particular state of A to be displayed by their corresponding signal cell. It is precisely the state obtained when applying a transition of $\mathcal{A}$ to the 3 local data cells. This state must eventually appear within $2n-1$ steps in each signal cell (thanks to the particular form of the signal configuration) so that after $2n-1$ steps, each memory cell contains the result of the transition of $\mathcal{A}$. After $2n$ steps the configuration is finally synchronised by the copy of memory cells to data cells and the cleaning of memory cells. □

Lemma 3. *For any* CA $\mathcal{A}$ *and any surjective* CA $\mathcal{B}$*, if* $\mathcal{B} \sqsubseteq \tau(\mathcal{A})^{<m,m'>}$ *for some* $m, m' \in \mathbb{N}$ *then either* $\mathcal{B}$ *is the identity map, or* $\mathcal{B} \preceq \mathcal{A}$.

Proof. Let $\phi : B \to A_\tau^m$ be the injection involved in the relation $\mathcal{B} \sqsubseteq \tau(\mathcal{A})^{<m,m'>}$ and let L_ϕ be the semi-group generated by the set of words $\{\phi(b), b \in B\}$. Comparing L_ϕ to the language $L(\Gamma_\mathcal{A})$, two cases are to be considered:

- if $L_\phi \not\subseteq L(\Gamma_\mathcal{A})$ then either κ does not appear in L_ϕ and then $\mathcal{B} = id$ (since $\tau(\mathcal{A})$ does nothing on configuration without κ), or there is some $b_0 \in B$ such that κ appears in $\phi(b_0)$. In the latter case let $w \in L_\phi \setminus L(\Gamma_\mathcal{A})$ be a concatenations of words from $\phi(B)$ and consider the periodic configuration c_0 of period $\phi(b_0)w$. c_0 can be chosen different from $\overline{\kappa}$ (otherwise it implies that $\phi(b) = \kappa^m \ \forall b \in B$, hence $\mathcal{B}$ has only 1 state and then clearly $\mathcal{B} \preceq \mathcal{A}$). Let $p = |\phi(b_0)w|$. Then $\kappa^p \notin L(c_0)$. Besides, since $\mathcal{B}$ is surjective and $c_0 \in (\phi(B))^\mathbb{Z}$, the simulation of $\mathcal{B}$ by $\tau(\mathcal{A})$ implies that there exists some $c_{-1} \in \phi(B)^\mathbb{Z}$ such that $\tau(\mathcal{A})^{m'}(c_{-1}) = c_0$. Then, if $p-1 \geq 2$, $\kappa^{p-1} \notin L(c_{-1})$ (otherwise $\kappa^p \in L(c_0)$) and we can continue the same reasoning so that there must be some $c \in \phi(B)^\mathbb{Z}$ such that $\kappa^2 \notin L(c)$ and $\tau(\mathcal{A})^t(c) = c_0$ for some t. Clearly $\kappa \in L(c)$ since $\kappa \in L(c_0)$ and $\tau(\mathcal{A})$ is a CCA. Finally, by surjectivity of $\mathcal{B}$, there is c' such that $\tau(\mathcal{A})(c') = c$. Again we have $\kappa \in L(c')$ and thus $c' \in \Gamma_\mathcal{A}$ because a configurations not in $\Gamma_\mathcal{A}$ containing a κ necessarily leads to a configuration containing κ^2. Since $\Gamma_\mathcal{A}$ is stable under $\tau(\mathcal{A})$, we must have $c_0 \in \Gamma_\mathcal{A}$: contradiction with the initial choice of c_0.
- now suppose $L_\phi \subseteq L(\Gamma_\mathcal{A})$. If $c \in \Gamma_\mathcal{A}$ and c' are such that $\tau(\mathcal{A})^t(c') = c$ for some t then $c' \in \Gamma_\mathcal{A}$ because any $d \in \tau(\mathcal{A})\left(A_\tau^\mathbb{Z} \setminus \Gamma_\mathcal{A}\right)$ is such that $\kappa^2 \in L(d)$. Moreover the orbit of c' enters E (defined in proof of lemma 2) every $2n$ steps, so when $t \geq 2n$ we can consider

$$\chi(c) = \max_{t' \leq t} \{\tau(\mathcal{A})^{t'}(c') : \tau(\mathcal{A})^{t'}(c') \in E\}.$$

  Notice that the definition is independent of t and c' (because from $\chi(c)$ to c, $\tau(\mathcal{A})$ does not alter the state of data cells and since $\chi(c) \in E$ it is entirely determined by its data cells). Since $\mathcal{B}$ is surjective, any configuration of $\phi(B)^\mathbb{Z}$ can be reached in arbitrarily many steps so χ is well-defined on $\phi(B)^\mathbb{Z}$. Notice also that χ is a local mapping (each bloc of 6 states of c is mapped to a single bloc of 6 states in $\chi(c)$) and that it is injective (by determinism of $\tau(\mathcal{A})$). Moreover, on $\phi(B)^\mathbb{Z}$, χ commutes with $\tau(\mathcal{A})^{2n}$. Finally, notice that the mapping $\overline{\Upsilon}$ is one-to-one from $(A^n)^\mathbb{Z}$ to E. Then, from lemma 2 and properties of χ, we have the following commutative diagram:

$$\begin{array}{ccccccc}
(B^{6n})^{\mathbb{Z}} & \xrightarrow{\overline{\phi^{6n}}} & (A_\tau^{6mn})^{\mathbb{Z}} & \xrightarrow{\overline{\chi^{mn}}} & (\Upsilon(A^n)^m)^{\mathbb{Z}} & \xrightarrow{\overline{(\Upsilon^m)^{-1}}} & (A^{mn})^{\mathbb{Z}} \\
\downarrow{\scriptstyle \mathcal{B}^{<6n,2n>}} & & \downarrow{\scriptstyle \tau(\mathcal{A})^{<6nm,2nm'>}} & & \downarrow{\scriptstyle \tau(\mathcal{A})^{<6nm,2nm'>}} & & \downarrow{\scriptstyle \mathcal{A}^{<nm,m'>}} \\
(B^{6n})^{\mathbb{Z}} & \xrightarrow{\overline{\phi^{6n}}} & (A_\tau^{6mn})^{\mathbb{Z}} & \xrightarrow{\overline{\chi^{mn}}} & (\Upsilon(A^n)^m)^{\mathbb{Z}} & \xrightarrow{\overline{(\Upsilon^m)^{-1}}} & (A^{mn})^{\mathbb{Z}}
\end{array}$$

(here χ denotes the local map from A_τ^6 to A_τ^6 mentioned above). This shows $\mathcal{B}^{<6n,2n>} \sqsubseteq \mathcal{A}^{<nm,m'>}$ by the injection $(\Upsilon^m)^{-1} \circ \chi^{mn} \circ \phi^{6n}$. Hence $\mathcal{B} \preceq \mathcal{A}$ by definition. □

Theorem 2. *There exists r_0 such that for any fixed radius $r \geq r_0$, it is undecidable to know whether a* CCA *of radius r is intrinsically universal or not.*

Proof. Let $\mathcal{X}$ be the following CA over $\{0,1\}$: $\mathcal{X}(a,b,c) = b + c \bmod 2$. $\forall m,n$, $\mathcal{X}^{<m,n>}$ is always surjective but neither the identity map so we deduce from lemma 2 and lemma 3 that $\forall \mathcal{A}$: $\mathcal{X} \preceq \mathcal{A} \Leftrightarrow \mathcal{X} \preceq \tau(\mathcal{A})$.

N. Ollinger established in [1] the undecidability of intrinsic universality[1] by giving a recursive construction U_q such that for any CA $\mathcal{A}$ of radius 1:

- $U_q(\mathcal{A})$ has radius r_U (where r_U only depends on U_q, not on $\mathcal{A}$),
- if $\mathcal{A}$ is q-nilpotent over periodic configurations then $\mathcal{X} \not\preceq U_q(\mathcal{A})$,
- and if $\mathcal{A}$ is not q-nilpotent over periodic configurations then $U_q(\mathcal{A})$ is intrinsically universal.

A CA is q-nilpotent over periodic configurations if every periodic configuration leads to the same configuration $\overline{q}$ in finite time. The problem of nilpotency over periodic configurations for CA of radius 1 was proven undecidable in [10]. From the previous observation, the recursive construction $\tau \circ U_q$ has the same properties as U_q and maps to CCA of fixed radius $r_0 = 7r_U$. □

The following corollary shows that, as in the general case, there is no limit on how complex a CCA of fixed radius can be without being intrinsically universal. Hence non-universal CCA cannot be reduced to a negligible set of simple exceptions among an overwhelming majority of universal objects.

Corollary 3. *There exits r_0 such that for any $r \geq r_0$ and for any non-universal* CCA *$\mathcal{A}$ of radius r, there is a non-universal* CCA *$\mathcal{B}$ of radius r such that $\mathcal{A} \preceq \mathcal{B}$ but $\mathcal{B} \not\preceq \mathcal{A}$.*

Proof. Let $r \geq r_0$ be a fixed radius for all following CA, where r_0 is the constant of theorem 2. First we show that there is no CCA $\mathcal{A}$ which is non-universal and such that for all non-universal CCA $\mathcal{B}$, $\mathcal{B} \preceq \mathcal{A}$. As already pointed out in the general case by N. Ollinger in [5], this follows from the semi-decidability of intrinsic universality: with a maximal CCA for non-universal CCA we could semi-decide non-universality and combining the two semi-decision procedures we could finally decide intrinsic universality which contradicts theorem 2.

[1] In [1], the undecidability result is not formulated for a fixed radius, but it is easy to check that the result remains true.

Now let us show that for any pair $\mathcal{A}$, $\mathcal{B}$ of non-universal CCA, there is a non-universal CCA $\mathcal{C}$ such that $\mathcal{A} \preceq \mathcal{C}$ and $\mathcal{B} \preceq \mathcal{C}$. This fact complete the proof since it implies that there cannot be more than one non-universal CCA which is maximal for non-universal CCA. $\mathcal{C}$ can be chosen as follows. Let $\ll$ be a total ordering on $A \cup B$ such that $\forall a \in A$ and $\forall b \in B$: $b \ll a$ (up to renaming of states, we can suppose $A \cap B = \emptyset$). Then for any $u \in (A \cup B)^{2r+1}$, $\mathcal{C}$ is defined by:

$$\mathcal{C}(u) = \begin{cases} \mathcal{A}(u) & \text{if } u \in A^{2r+1}, \\ \mathcal{B}(u) & \text{if } u \in B^{2r+1}, \\ \max \Sigma(u) & \text{otherwise,} \end{cases}$$

where max is relative to the order $\ll$. Clearly $\mathcal{C}$ is a CCA and $\mathcal{A} \preceq \mathcal{C}$ and $\mathcal{B} \preceq \mathcal{C}$. Moreover if $\mathcal{C}$ is universal, then $\mathcal{A}$ or $\mathcal{B}$ must also be universal. To see this consider a universal CA $\mathcal{U}$ such that for any pair a, b of states the periodic configuration of period ab is a fixed point (to be convinced of the existence of $\mathcal{U}$ notice that any behaviour can be specified on "non-coding" configurations in the construction of a universal CA). Hence, if $\mathcal{C}$ is universal then $\mathcal{U} \sqsubseteq \mathcal{C}^{<m,n>}$ for some $m, n \in \mathbb{N}$. And, denoting by ϕ the injection involved in this relation, we must have $\phi(U) \subseteq A^m$ or $\phi(U) \subseteq B^m$ because, otherwise, there would exists u_1 and u_2 in U such that the word $\phi(u_1)\phi(u_2)$ contains both a state from A and a state from B. From the definition of $\mathcal{C}$, the periodic configuration of period $\phi(u_1)\phi(u_2)$ cannot be a fixed point (because any state of B ultimately disappears) which contradicts the property of $\mathcal{U}$. Therefore we must have either $\mathcal{U} \sqsubseteq \mathcal{A}^{<m,n>}$ or $\mathcal{U} \sqsubseteq \mathcal{B}^{<m,n>}$. □

3 What About the General Case?

There is no zero-one law for monotonic properties on CA in general. Indeed, let $\mathcal{P}$ be the property of possessing at least one quiescent state ($q \in A$ is quiescent for $\mathcal{A}$ if $\mathcal{A}(\overline{q}) = \overline{q}$). Then $\mathcal{P}$ is an increasing property but $0 < \mu(\mathcal{P}) < 1$. To be precise, it is not difficult to show that

$$\mu(\neg\mathcal{P}) = \lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e}.$$

Actually, there are also increasing properties with density 0. This is what we are going to show with the property $\mathcal{P}_{\sqsubseteq}$ of possessing a non trivial sub-automaton: $\mathcal{P}_{\sqsubseteq} = \{\mathcal{A} : \exists \mathcal{B}, 1 < |B| < |A| \text{ and} \mathcal{B} \sqsubseteq \mathcal{A}\}$.

Before giving the proof, notice that this shows at least that arguments similar to those of theorem 1 cannot be used in the general case. The following lemma establish a useful upper bound.

Lemma 4. $\exists n_0 \forall n \geq n_0 \forall k, 2 \leq k \leq n-1 : \left(\frac{k}{n}\right)^{k^{2r+1}} \leq n^{-2k}$.

Proof. Taking the log of the expression, it is sufficient to show that the following relation eventually holds uniformly for k: $\frac{k^{2r}}{2} \log \frac{n}{k} \geq \log n$. This majoration is obtained by a standard analysis of the real function

$$f(k,n) : k, n \mapsto \frac{k^{2r}}{2} \log \frac{n}{k}$$

as follows. First, $f(2,n) \geq \log n$ and $f(n-1,n) \geq \log n$ are eventually true. Then $\frac{\partial f}{\partial k}(k,n) = rk^{2r-1} \log \frac{n}{k} - \frac{k^{2r-1}}{2}$ so it equals zero for $k = e^{-\frac{1}{2r}} n$. Finally, for any $0 < \alpha < 1$, $f(\alpha n, n)$ is eventually greater than $\log n$ since it is a monomial in n with a positive coefficient. □

Proposition 1. *For any fixed radius, almost no* CA *possesses a non-trivial sub-automaton:* $\mu(\mathcal{P}_{\sqsubseteq}) = 0$.

Proof. For $2 \leq k \leq n-1$, the probability that $\mathcal{A} \in A_n$ has a sub-automaton with k states is bounded by $C_n^k \left(\frac{k}{n}\right)^{k^{2r+1}}$: for a fixed choice of k states, each transition involving only these k states must lead to one of them. Hence we have:

$$\mu(\mathcal{P}_{\sqsubseteq}) \leq \lim_{n\to\infty} \sum_{k=2}^{n-1} C_n^k \left(\frac{k}{n}\right)^{k^{2r+1}}. \tag{1}$$

Using majoration of lemma 4 in the inequality, we obtain:

$$\begin{aligned}
\mu(\mathcal{P}_{\sqsubseteq}) &\leq \lim_{n\to\infty} \sum_{k=2}^{n-1} C_n^k \left(\frac{1}{n^2}\right)^k \\
&\leq \lim_{n\to\infty} \left(\sum_{k=0}^{n} C_n^k \left(\frac{1}{n^2}\right)^k - \sum_{k\in\{0,1,n\}} C_n^k \left(\frac{1}{n^2}\right)^k \right) \\
&\leq \lim_{n\to\infty} \left(\left(1 + \frac{1}{n^2}\right)^n - 1 - \frac{1}{n} - \frac{1}{n^{2n}} \right).
\end{aligned}$$

Thus $\mu(\mathcal{P}_{\sqsubseteq}) = 0$. □

4 Perspectives and Open Problems

We have shown that intrinsic universality is ubiquitous in CCA. Actually, the result extends to any reasonable notion of universality including Turing universality. We insist however that the set of intrinsically universal CCA is rich because non recursive. Moreover, lemma 3 shows that it is possible given any CA to construct a CCA that is somehow similar with respect to $\preceq$. Can this lemma be extended and more precisely can we characterise $\simeq$-classes containing a CCA? Or at least give a large collection of $\simeq$-classes containing a CCA?

Besides, although the density result for CCA gives a lower bound on the growing rate of intrinsically universal CA, the density problem for CA remains open since CCA constitute a negligible subset of cellular automata (see proposition 1). However, if this density turned out to be non-zero, it would mean that a significant part of CA acquire any sub-structure by simple spatio-temporal

transformation while they are locally totally unstructured (proposition 1). To that extent, a study of the way a CA can or cannot acquire structure by spatio-temporal transformations reveals to be essential to decide the density problem. In [11] it is shown that some CA avoid some sub-automaton size even up to spatio-temporal transformations. Can we extend this kind of result and show that there exists some CA which has no non-trivial sub-automaton at any scale? Or conversely is there for any CA some unavoidable transformation giving him some non-trivial structure even if it is totally unstructured locally?

Acknowledgement

We thank E. JEANDEL for helpful discussions regarding proposition 1.

References

1. Ollinger, N.: The intrinsic universality problem of one-dimensional cellular automata. In: Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science (2003) 632–641
2. Banks, E.R.: Universality in cellular automata. In: Eleventh Annual Symposium on Switching and Automata Theory, Santa Monica, California, IEEE (1970)
3. Ollinger, N.: The quest for small universal cellular automata. In: International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science (2002) 318–330
4. Mazoyer, J., Rapaport, I.: Inducing an Order on Cellular Automata by a Grouping Operation. In: Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science (1998)
5. Ollinger, N.: Automates Cellulaires : structures. PhD thesis, École Normale Supérieure de Lyon (2002)
6. Albert, J., Čulik, II, K.: A simple universal cellular automaton and its one-way and totalistic version. Complex Systems **1** (1987) 1–16
7. Theyssier, G.: Captive cellular automata. In: Mathematical Foundations of Computer Science, Lecture Notes in Computer Science (2004) 427–438
8. Kůrka, P.: Languages, equicontinuity and attractors in cellular automata. Ergodic Theory and Dynamical Systems **17** (1997) 417–433
9. Hedlund, G.A.: Endomorphisms and Automorphisms of the Shift Dynamical Systems. Mathematical Systems Theory **3** (1969) 320–375
10. Mazoyer, J., Rapaport, I.: Global fixed point attractors of circular cellular automata and periodic tilings of the plane: undecidability results. Discrete Mathematics **199** (1999) 103–122
11. Mazoyer, J., Rapaport, I.: Additive cellular automata over Z_p and the bottom of $(CA, \leq)$. In: Mathematical Foundations of Computer Science, Lecture Notes in Computer Science (1998) 834–843

Cellular Automata: Real-Time Equivalence Between One-Dimensional Neighborhoods

Victor Poupet

LIP (UMR CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1),
École Normale Supérieure de Lyon,
46 allée d'Italie 69364 LYON cedex 07 France

Abstract. It is well known that one-dimensional cellular automata working on the usual neighborhood are Turing complete, and many acceleration theorems are known. However very little is known about the other neighborhoods. In this article, we prove that every one-dimensional neighborhood that is sufficient to recognize every Turing language is equivalent (in terms of real-time recognition) either to the usual neighborhood $\{-1, 0, 1\}$ or to the one-way neighborhood $\{0, 1\}$.

Keywords: Cellular automata, neighborhoods, real-time.

1 Introduction

Cellular automata (CA) are a computation model that is simple at microscopic scale (local transitions) but can have very complex behaviours at macroscopic scale (global transitions). As a complex systems, it is natural to wonder what their compuational capabilities are. It is known that they can simulate any Turing Machine [15, 10, 1] (Turing universality) and that there exist "universal" CA that can simulate the evolution of any other [8] (intrinsic universality).

Once their computational power is known, we investigate the complexity of such computations (in time and memory). Many famous algorithms [6, 5] have shown that the massively parallel structure of the CA enables complex computations to be realized in a very short time. However, because of the local communication between cells, it is possible to show that information cannot be transmitted faster than some "maximal speed".

In this article, we will work on the problem of language recognition by CA, and more specifically the recognition of languages in "real time", which is the fastest possible time according to the "maximal speed" restriction. Many results are already known on this topic concerning one-dimensional CA working on the standard neighborhood [11, 7, 4, 9, 12, 13], but very little is known about CA working on different neighborhoods.

Concerning Turing machines, it is easy to show that any computation can be accelerated by a constant time (as long as the time stays greater than the minimal time necessary to read the entry). As for cellular automata, C. Choffrut and K. Čulik in [2] first show that with the neighborhood $\{-1, 0, 1\}$, any computation

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 133–144, 2005.

on a CA can be accelerated by a constant time. In [9], J. Mazoyer and N. Reimen show that for any positive integers r and q, the neighborhoods $[-r, r]$ and $[-q, q]$ are time wise equivalent. Moreover, S. Cole [3] showed that on every neighborhood "complete enough" (such that every cell can, after some finite time, be affected by the state of any other) it is possible to simulate the behaviour of any CA working on any other neighborhood. This last result gives a first computational equivalence between neighborhoods.

Here, a strong "real-time equivalence" between neighborhoods will be proved that shows that the languages recognized in real-time by CA working on a given neighborhood "complete enough" (similarly to Cole's definition) are either exactly the same as the ones recognized in real-time by CA working on the usual neighborhood or exactly the ones recognized in real-time by CA working on the one-way neighborhood (one-way cellular automata).

2 Preliminary Study: Neighborhoods Growth

In all this section, a *neighborhood* will be a finite subset of $\mathbb{Z}$. We will here study an algebraic property of neighborhoods that is independent of the notion of cellular automaton.

Given a neighborhood V, we will use the notations $V^0 = \{0\}$ and for all $k \in \mathbb{N}$, $V^{k+1} = \{x + y | x \in V, y \in V^k\}$. Moreover, we will always consider that the neighborhoods contain 0.

Definition 21. *Let V be a neighborhood. V is*

- r-complete *if* $\forall n \in \mathbb{N}, \exists k \in \mathbb{N}, n \in V^k$*;*
- complete *if* $\forall n \in \mathbb{Z}, \exists k \in \mathbb{N}, n \in V^k$*;*
- r-incomplete *if it is* r-complete *but not* complete*.*

Proposition 21. *A neighborhood V is r-incomplete if and only if it contains 1 and has no negative element. It is complete if and only if its non-zero elements are prime altogether (their gcd is 1) and has both a positive and a negative element.*

The proof is a trivial combination of the previous definitions and Bezout's theorem.

Remark. We will see later that the r-complete neighborhoods are the neighborhoods on which a CA can correctly recognize a language.

From now on, if V is an r-complete neighborhood, we will use the following notations

$$\begin{aligned} x_p &= \max V \\ -x_n &= \min V \\ t_c &= \min\{t \in \mathbb{N} | [\![-x_n, x_p]\!] \subseteq V^{t+1}\} \end{aligned}$$

Remark. Proposition 21 ensures that t_c is correctly defined.

Proposition 22. *Let V be an r-complete neighborhood. For all $t \in \mathbb{N}$, we have*

$$[\![-tx_n, tx_p]\!] \subseteq V^{t+t_c} \subseteq [\![-(t+t_c)x_n, (t+t_c)x_p]\!]$$

Proof. The rightmost inclusion is trivial. As for the leftmost one, we prove it inductively. For $t = 0$ and $t = 1$, the property is obvious. For $t \geq 1$, we assume that $[\![-tx_n, tx_p]\!] \subseteq V^{t+t_c}$. So

$$[\![-(t+1)x_n, (t+1)x_p]\!] = [\![-tx_n, tx_p]\!] + \{-x_n, 0, x_p\} \subseteq V^{t+t_c} + V = V^{t+t_c+1}$$

□

Definition 22. *Let V be an r-complete neighborhood. We will call* r-shadow *of V at time $(t + t_c)$ the set*

$$S^+_{t+t_c}(V) = \left(V^{t+t_c} \cap [\![tx_p, (t+t_c)x_p]\!]\right) - tx_p$$

Similarly, we'll call l-shadow *of V at time $(t + t_c)$ the set*

$$S^-_{t+t_c}(V) = \left(V^{t+t_c} \cap [\![-(t+t_c)x_n, -tx_n]\!]\right) + tx_n$$

Remark. According to the proposition 22,

$$V^{t+t_c} = \left[S^-_{t+t_c}(V) - tx_n\right] \cup [\![-tx_n, tx_p]\!] \cup \left[S^+_{t+t_c}(V) + tx_p\right]$$

Proposition 23. *Let V be an r-complete neighborhood. The sequences*

$$\left(S^+_{t+t_c}(V)\right)_{t\in\mathbb{N}} \text{ and } \left(S^-_{t+t_c}(V)\right)_{t\in\mathbb{N}}$$

are ultimately constant.

Proof. Let $x \in S^+_{t+t_c}(V)$, then $(x+tx_p) \in V^{t+t_c}$ and so $(x+(t+1)x_p) \in V^{t+t_c+1}$ and finally $x \in S^+_{t+1+t_c}(V)$. This proves that $\left(S^+_{t+t_c}(V)\right)_{t\in\mathbb{N}}$ is an increasing sequence (where the considered order is the inclusion). Moreover, this sequence's elements are all in the subsets of $[\![0, t_c x_p]\!]$, which is a finite set. This implies that the sequence is ultimately constant. The proof is similar for $\left(S^-_{t+t_c}(V)\right)_{t\in\mathbb{N}}$. □

Definition 23. *Let V be an r-complete neighborhood. The* stabilization time *t_s of V is the smallest integer from which the sequences $\left(S^+_{t+t_c}(V)\right)_{t\in\mathbb{N}}$ and $\left(S^-_{t+t_c}(V)\right)_{t\in\mathbb{N}}$ are constant.*

We also define $S^+(V) = S^+_{t_c+t_s}$, $S^-(V) = S^-_{t_c+t_s}$ and $x_0 = \min\{x \in \mathbb{N} | x \notin S^+(V)\}$.

Proposition 23 ensures that t_s is correctly defined for all r-complete neighborhoods. We have proven the following theorem

Theorem 21. *For all r-complete neighborhood V, there exists an integer t_s, and two sets $S^-(V) \subseteq [\![-t_c x_n, 0]\!]$ and $S^+(V) \subseteq [\![0, t_c x_p]\!]$ such that for all $t \geq t_s$,*

$$V^{t+t_c} = \left(S^-(V) - tx_n\right) \cup [\![-tx_n, tx_p]\!] \cup \left(S^+(V) + tx_p\right)$$

3 Language Recognition by Cellular Automata

3.1 Cellular Automata

Definition 31. *A* cellular automaton *(CA) is a triple* $\mathcal{A} = (\mathcal{Q}, V, f)$ *where*

- $\mathcal{Q}$ *is a finite set called* set of states *containing a special* quiescent *state* #;
- $V = \{v_1, \dots, v_{|V|}\} \subseteq \mathbb{Z}$ *is a finite set called* neighborhood *that contains 0.*
- $f : \mathcal{Q}^{|V|} \to \mathcal{Q}$ *is the* transition function. *We have* $f(\#, \dots, \#) = \#$.

For a given automaton $\mathcal{A}$, we call *configuration* of $\mathcal{A}$ any function $\mathfrak{C}$ from $\mathbb{Z}$ into $\mathcal{Q}$. The set of all configurations is therefore $\mathcal{Q}^{\mathbb{Z}}$. From the local function f we can define a global function F

$$\begin{array}{l} F : \mathcal{Q}^{\mathbb{Z}} \to \mathcal{Q}^{\mathbb{Z}} \\ \qquad \mathfrak{C} \mapsto \mathfrak{C}' \quad | \quad \forall x \in \mathbb{Z}, \mathfrak{C}'(x) = f(\mathfrak{C}(x + v_1), \dots, \mathfrak{C}(x + v_{|V|})) \end{array}$$

Elements of $\mathbb{Z}$ are called *cells*. Given a configuration $\mathfrak{C}$, we'll say that a cell c is in state q if $\mathfrak{C}(c) = q$.

If at time $t \in \mathbb{N}$ the CA is in a configuration $\mathfrak{C}$, we'll consider that at time $(t + 1)$ it is in the configuration $F(\mathfrak{C})$. This enables to define the *evolution* of a CA from a configuration. This evolution is completely determined by $\mathfrak{C}$.

3.2 Language Recognition

Definition 32. *We consider a CA* $\mathcal{A} = (\mathcal{Q}, V, f)$ *and a set* $\mathcal{Q}_{acc} \subseteq \mathcal{Q}$ *of* accepting states. *Let* $w = w_0 w_1 \dots w_{l-1}$ *be a word on a finite alphabet* $A \subseteq \mathcal{Q}$. *We define the configuration* $\mathfrak{C}_w$ *as follows.*

$$\mathfrak{C}_w : \quad \begin{array}{l} \mathbb{Z} \to \mathcal{Q} \\ \begin{cases} x \mapsto w_x \text{ if } 0 \leq x < l \\ x \mapsto \# \quad \text{otherwise} \end{cases} \end{array}$$

We'll say that the CA $\mathcal{A}$ recognizes *the word* w *with accepting states* $\mathcal{Q}_{acc}$ *in time* t_w *if, starting from the configuration* $\mathfrak{C}_w$ *at time* 0, *the cell 0 is in a state in* $\mathcal{Q}_{acc}$ *at time* t_w.

Definition 33. *Let* $\mathcal{A} = (\mathcal{Q}, V, f)$ *be a CA and* $L \subseteq A^*$ *a language on the alphabet* $A \subseteq \mathcal{Q}$. *For a given function* T *from* $\mathbb{N}$ *into* $\mathbb{N}$, *we'll say that the language* L *is* recognized by $\mathcal{A}$ in time T *if there exists a set* $\mathcal{Q}_{acc} \subseteq \mathcal{Q}$ *such that, for all word* w *of length* l *in* A^*, *the CA* $\mathcal{A}$ *recognizes* w *with accepting states* $\mathcal{Q}_{acc}$ *in time* $T(l)$ *if and only if* $w \in L$.

3.3 Real-Time

Definition 34. *Given an r-complete neighborhood* V, *we define the* real-time function *on* V

$$\begin{array}{l} TR_V : \mathbb{N} \to \mathbb{N} \\ \qquad l \mapsto \min\{t \in \mathbb{N} | [\![0, l-1]\!] \subseteq V^t\} \end{array}$$

Definition 35. *Let $\mathcal{A} = (\mathcal{Q}, V, f)$ be a CA where V is r-complete and L a language on $A \subseteq \mathcal{Q}$. We'll say that the CA $\mathcal{A}$ recognizes L in* real-time *if it recognizes L in time TR_V.*

4 Constant Speed-Up Theorem

In this section we will prove the following theorem

Theorem 41. *Let V be an r-complete neighborhood. For all $k \in \mathbb{N}$, if a language L can be recognized by a CA working on the neighborhood V in time $TR_V + k$ then it is recognized by a CA working on V in real-time.*

To prove this theorem, let's consider an r-complete neighborhood V (and all the corresponding notations from section 2). Let L be a language on a given alphabet recognized by a CA $\mathcal{A}$ working on V in time $TR_V + k$. Let $\mathcal{Q}$ be the set of states of $\mathcal{A}$.

We will construct a CA $\mathcal{A}'$ working on V that will simulate the evolution of $\mathcal{A}$ but with a constant speed-up (of k generations).

In all the following proof, we will consider the evolution of the automaton $\mathcal{A}$ from the initial configuration corresponding to a word $w = w_0 \dots w_{l-1}$ of length l. The initial configuration is

$$\dots ###w_0w_1 \dots w_{l-1}### \dots$$

The state of the cell $c \in \mathbb{Z}$ at time t in the evolution of $\mathcal{A}$ will be noted $\langle c \rangle_t$. The states of $\mathcal{A}'$ will be tuples of elements of $\mathcal{Q}$.

4.1 The Evolution of $\mathcal{A}'$

Time 0. The initial configuration is the same than the one of $\mathcal{A}$. Each cell c is in the state $\langle c \rangle_0$.

Time $0 \to (t_c + t_s)$. The cells will gather information, meaning that they will only read the state of their neighbors (the transition rule of $\mathcal{A}$ is not applied) and memorize the state of the cells that are further away from them.

Time $(t_c + t_s)$. Each cell c knows exactly the states $\{\langle c + x \rangle_0 | x \in V^{t_c+t_s}\}$, and will "assume" that all states $\{\langle c + x \rangle_0 | x \in [\![t_s x_p, (t_c + t_s + k)x_p]\!]\}$ that it doesn't already know (the ones not in $c + V^{t_c+t_s}$) are #. Note that this assumption is true for the cells that are close enough to the end of the word w, and false for the others (we'll see this more in details later). Also each cell will do the symmetrical assumption that all states in $\{\langle c - x \rangle_0 | x \in [\![t_s x_n, (t_c + t_s + k)x_n]\!]\}$ that it doesn't know are #.

¿From now on we'll make a difference between the information that a cell "knows" and the information it "assumes" on its right and on its left.

Time $(t_c + t_s) \to \infty$. Here, each cell will apply the transition rule of $\mathcal{A}$ to all the information it knows. This way, at time $(t_c + t_s + t)$ each cell knows the states $\{\langle c + x \rangle_t | x \in V^{t_c+t_s}\}$. Also, it will apply the transition rule to the assumed information. In this computation, the cell might have incompatible information

(what it has assumed so far and what the cell $(c + x_p)$ knows or assumes for example). In this case, it will always give the priority to the information held by the cell $(c+x_p)$ when computing assumptions on its right, and to the information held by $(c - x_n)$ when computing assumptions on its left.

4.2 Why the Simulation Works

Here we will prove that the automaton $\mathcal{A}'$ described in the previous subsection recognizes the language L in real-time. We will still focus on the evolution from the word w and keep the notations of the states in the evolution of $\mathcal{A}$.

Claim 41. *At time $(t_c + t_s)$ the cell c knows the states $\{\langle c + x\rangle_0 | x \in V^{t_c+t_s}\}$.*

Proof. By induction, if at time t the cell knows $\{\langle c + x\rangle_0 | x \in V^t\}$ then it can see its neighbors $(V + c)$ and their stored information. At time $(t + 1)$ it can therefore have the information

$$\{\langle c + v + x\rangle_0 | v \in V, x \in V^t\} = \{\langle c + x\rangle_0 | x \in V^{t+1}\}$$

□

Claim 42. *At time (t_c+t_s+t) the cell c knows the states $\{\langle c + x\rangle_t | x \in V^{t_c+t_s}\}$.*

Proof. Induction again. To compute the states

$$\{\langle c + x\rangle_{t+1} | x \in V^{t_c+t_s}\}$$

the cell c needs to see the states

$$\{\langle c + x\rangle_t | x \in (V^{t_c+t_s} + V)\}$$

which it does by looking at its neighbors in $(V + c)$. □

Definition 41. *At a given time t, a cell c will be called* r-correct *if all the assumptions it does on its right are correct. The cell will be called* r-incorrect *otherwise.*

We have similar definitions of l-correct *and* l-incorrect *for the left side.*

Remark. If at time $(t_c + t_s + t)$ the cell c is r-correct, then it knows or assumes correctly all the states $\{\langle c + x\rangle_t | x \in [\![-t_s x_n, (t_c + t_s + k)x_p]\!]\}$.

Claim 43. *At time $(t_c + t_s)$ all the cells $c \geq (l - t_s x_p - x_0)$ are r-correct.*

Proof. These cells assume that the initial states of the cells $c \geq l$ are # which is true according to the initial configuration (by definition $(t_s x_p + x_0)$ is the smallest positive integer not in $V^{t_c+t_s}$). □

Claim 44. *If the cell $(c + x_p)$ is r-correct at time $(t_c + t_s + t)$ then the cell c is r-correct at time $(t_c + t_s + t + 1)$.*

Proof. We have seen that when the cell c applies the transition rule to the assumed information, it considers the information held by $(c + x_p)$ with a higher priority. To compute correctly the states

$$\{\langle c + x\rangle_{t+1} | x \in [\![t_s x_p, (t_c + t_s + k)x_p]\!]\}$$

the cell c needs to see the states

$$\{\langle c + x + v\rangle_t | v \in V, x \in [\![t_s x_p, (t_c + t_s + k)x_p]\!]\}$$

that are all included in

$$\{\langle c + x_p + x\rangle_t | x \in [\![-t_s x_n, (t_c + t_s + k)x_p]\!]\}$$

so c sees all the correct information at time $(t_c + t_s + t)$ and is therefore r-correct at time $(t_c + t_s + t + 1)$. □

Claim 45. *At time $(t_c + t_s + t)$, all cells $c \geq l - (t_s + t)x_p - x_0$ are r-correct.*

Proof. Immediate consequence of claims 43 and 44. □

Claim 46. *At every time, the origin and all negative cells are l-correct.*

Proof. The proof is easy and similar to the previous ones □

Claim 47. *If the length of the word w is such that $l \geq (t_s + 1)x_p + x_0 + 1$ then the real-time (for this word) is $TR_V(l) \geq t_c + \left\lfloor \frac{l-1-x_0}{x_p} \right\rfloor + 1$.*

Proof. Let's define $\alpha = \left\lfloor \frac{l-1-x_0}{x_p} \right\rfloor$. We have $\alpha x_p + x_0 \leq l - 1$. Moreover, since $l \geq (t_s + 1)x_p + x_0 + 1$ we have $\alpha \geq t_s$ and so by theorem 21 we have

$$(\alpha x_p + x_0) \notin V^{t_c+\alpha}$$

which means that $TR_V(l) \geq t_c + \alpha + 1$. □

Claim 48. *If the length of w is such that $l \geq (t_s + 1)x_p + x_0 + 1$ then at time $TR_V(l)$ the automaton $\mathcal{A}'$ can determine whether or not the word w is in L.*

Proof. From claim 45, we know that the origin is r-correct at times

$$t \geq t_c + \left\lceil \frac{l - x_0}{x_p} \right\rceil = t_c + \left\lfloor \frac{l - 1 - x_0}{x_p} \right\rfloor + 1$$

This means that the origin is r-correct at time $TR_V(l)$ (from claim 47) and, since it is always l-correct (claim 46), at time $TR_V(l)$ the origin knows (or assumes correctly) all the states in

$$\{\langle c\rangle_{TR_V(l)-t_c-t_s} | c \in [\![-(t_c+t_s+k)x_n, (t_c+t_s+k)x_p]\!]\}$$

which contain all the necessary information to compute the state $\langle 0\rangle_{TR+k}$. The automaton can therefore determine if w is in L. □

This last proposition and the fact that there is only a finite number of words of length $l \leq (t_s+1)x_p + x_0$ (so the automaton $\mathcal{A}'$ can handle these exceptions directly) end the proof of theorem 41.

5 Real-Time Equivalences

Definition 51. *For every neighborhood V and every natural number n, we will define $L^n_{TR}(V)$ the set of languages on the alphabet $[\![0, n-1]\!]$ that can be recognized in real-time by a CA working on V. We will also define*

$$L_{TR}(V) = \bigcup_{n\in\mathbb{N}} L^n_{TR}(V)$$

This section is dedicated to the proof of the following theorem.

Theorem 51. *Let V and V' be two neighborhoods, we have the following relations:*

1. *if V is r-complete and V' is complete then $L_{TR}(V) \subseteq L_{TR}(V')$;*
2. *if V and V' are complete then $L_{TR}(V) = L_{TR}(V')$;*
3. *if V and V' are r-incomplete then $L_{TR}(V) = L_{TR}(V')$.*

Proposition 51. *For all r-complete neighborhood V, if $-x_n = \min V$ and $x_p = \max V$ then $L_{TR}(V) = L_{TR}([\![-x_n, x_p]\!])$.*

Proof. It is obvious that any automaton on V can be simulated by an automaton on $[\![-x_n, x_p]\!]$ without any loss in time. Proposition 22 shows that the difference between the real-time on V and the real-time on $[\![-x_n, x_p]\!]$ is at most t_c for any word. Therefore, from theorem 41, we have $L_{TR}(V) \subseteq L_{TR}([\![-x_n, x_p]\!])$.

Now if a language L is recognized in real-time by an automaton working on $[\![-x_n, x_p]\!]$ we can construct an automaton working on V that will gather information during the first t_c steps. From there each cell sees enough information at each step to apply the transition rule of the automaton working on $[\![-x_n, x_p]\!]$ and update its information. The resulting automaton recognizes L in time $TR_{[\![-x_n,x_p]\!]} + t_c$ which is obviously smaller than $TR_V + t_c$, and so by theorem 41 we have the converse inclusion. □

Proposition 52. *For all r-complete neighborhood V and all $k \in \mathbb{N}^*$, we have $L_{TR}(V) = L_{TR}(V^k)$.*

Proof. By proposition 51 we can restrict ourself to the connex neighborhoods. It is clear that for a connex neighborhood V, $TR_V \geq k(TR_{V^k} - 1)$. We know that one step of a CA working on V^k can be simulated by a CA working on V in k steps, so every language L that can be recognized in time TR_{V^k} on V^k can be recognized in time $kTR_{V^k} \leq TR_V + k$. From theorem 41 we have $L_{TR}(V^k) \subseteq L_{TR}(V)$.

Reciprocally, it is easy to see that $TR_{V^k} \geq \lfloor TR_V/k \rfloor$. Since we know that a CA working on V^k can simulate k steps of a CA working on V in only one generation, every language that can be recognized on V in real-time can be recognized on V^k in time $\lceil TR_V/k \rceil \leq TR_{V^k} + 1$. Conclusion comes from theorem 41. □

Proposition 53. *For all $x_n \leq x'_n \in \mathbb{N}$ and $x_p \in \mathbb{N}^*$, we have*

$$L_{TR}([\![-x_n, x_p]\!]) \subseteq L_{TR}([\![-x'_n, x_p]\!])$$

Proof. It is enough to observe that the real-time functions for both of these neighborhoods are the same, and since it is obvious that every CA working on the smaller can be simulated without loss of time by a CA working on the bigger we have this trivial inclusion. □

Proposition 54. *For all x_n and x_p in $\mathbb{N}^*$ we have*

$$L_{TR}([\![-x_n, x_p]\!]) = L_{TR}([\![-2x_n, x_p]\!])$$

Proof. It is well known that if there is a CA working on $[\![-2x_n, x_p]\!]$ that recognizes L in real-time, then there exists another CA working on the same neighborhood that recognizes L in real-time on limited space (meaning that a cell in state # never changes its state). This technique is very similar to the technique used to prove that Turing machines working on a semi-infinite tape are equivalent to the ones working on a bi-infinite tape.

To prove the inclusion $L_{TR}([\![-2x_n, x_p]\!]) \subseteq L_{TR}([\![-x_n, x_p]\!])$, consider an automaton $\mathcal{A}$ working on $V_2 = [\![-2x_n, x_p]\!]$ that recognizes a language L in real-time and limited space. Let $w = w_0 w_1 \dots w_{l-1}$ be a word, and let's consider the evolution of $\mathcal{A}$ from the initial configuration corresponding to w. Like previously, we will denote as $\langle c \rangle_t$ the state of the cell c at time t in the evolution of $\mathcal{A}$.

Now we will explain the behaviour of a CA working on the neighborhood $V_1 = [\![-x_n, x_p]\!]$ starting from the same initial configuration, and recognizing the language L in exactly the same time as $\mathcal{A}$ (the evolution of $\mathcal{A}'$ on an initial configuration corresponding to a word of length 7 for $V_1 = [\![-1, 1]\!]$ is illustrated on figure 1).

- Time 0. Each cell c has the information $\langle c \rangle_0$.
- Time 1. Cell c has the information $\langle c \rangle_0$ and $\langle c + 1 \rangle_0$. This is possible because $\{0, 1\} \subseteq V$.

- When $1 \leq t \leq c$, the cell c has the information $\langle c+t-1\rangle_0$ and $\langle c+t\rangle_0$. This again is possible simply by looking at the cell $(c+1)$.
- When $t = c+k$ with $0 \leq k \leq x_n$, the cell c has information $\langle 2c-1\rangle_0$, $\langle 2c-1\rangle_1$, ..., $\langle 2c-1\rangle_k$ and $\langle 2c\rangle_0$, $\langle 2c\rangle_1$, ..., $\langle 2c\rangle_k$. The information up to time $(k-1)$ is already on c at the previous time (only memory), and the information of time k can be computed from the information c sees because by construction it can now see "twice as far" to the left, from the compression.
- At time $(c+x_n+t)$, the cell c has the information $\langle 2c-1\rangle_t$, $\langle 2c-1\rangle_{t+1}$, ..., $\langle 2c-1\rangle_{t+x_n}$ and $\langle 2c\rangle_t$, $\langle 2c\rangle_{t+1}$, ..., $\langle 2c\rangle_{t+x_n}$. Again, some information was already on c at the previous time, the rest can be computed.

With such an evolution, we can see that the origin 0 has at each step t the information $\langle 0\rangle_t$ so the automaton $\mathcal{A}'$ can decide whether or not w is in L at time $TR_{V_1}(l) = TR_{V_2}(l)$, which proves the inclusion. The converse inclusion is covered by proposition 53. □

...	#	$\langle 0\rangle_6\langle 0\rangle_5$	$\langle 1\rangle_5\langle 2\rangle_5$ $\langle 1\rangle_4\langle 2\rangle_4$	$\langle 3\rangle_4\langle 4\rangle_4$ $\langle 3\rangle_3\langle 4\rangle_3$	$\langle 5\rangle_3\langle 6\rangle_3$ $\langle 5\rangle_2\langle 6\rangle_2$	#	...			
...	#	$\langle 0\rangle_5\langle 0\rangle_4$	$\langle 1\rangle_4\langle 2\rangle_4$ $\langle 1\rangle_3\langle 2\rangle_3$	$\langle 3\rangle_3\langle 4\rangle_3$ $\langle 3\rangle_2\langle 4\rangle_2$	$\langle 5\rangle_2\langle 6\rangle_2$ $\langle 5\rangle_1\langle 6\rangle_1$	#	...			
...	#	$\langle 0\rangle_4\langle 0\rangle_3$	$\langle 1\rangle_3\langle 2\rangle_3$ $\langle 1\rangle_2\langle 2\rangle_2$	$\langle 3\rangle_2\langle 4\rangle_2$ $\langle 3\rangle_1\langle 4\rangle_1$	$\langle 5\rangle_1\langle 6\rangle_1$ $\langle 5\rangle_0\langle 6\rangle_0$	#	...			
...	#	$\langle 0\rangle_3\langle 0\rangle_2$	$\langle 1\rangle_2\langle 2\rangle_2$ $\langle 1\rangle_1\langle 2\rangle_1$	$\langle 3\rangle_1\langle 4\rangle_1$ $\langle 3\rangle_0\langle 4\rangle_0$	$\langle 5\rangle_0\langle 6\rangle_0$	$\langle 6\rangle_0$#	#	...		
...	#	$\langle 0\rangle_2\langle 0\rangle_1$	$\langle 1\rangle_1\langle 2\rangle_1$ $\langle 1\rangle_0\langle 2\rangle_0$	$\langle 3\rangle_0\langle 4\rangle_0$	$\langle 4\rangle_0\langle 5\rangle_0$	$\langle 5\rangle_0\langle 6\rangle_0$	$\langle 6\rangle_0$#	#	...	
...	#	$\langle 0\rangle_1\langle 0\rangle_0$	$\langle 1\rangle_0\langle 2\rangle_0$	$\langle 2\rangle_0\langle 3\rangle_0$	$\langle 3\rangle_0\langle 4\rangle_0$	$\langle 4\rangle_0\langle 5\rangle_0$	$\langle 5\rangle_0\langle 6\rangle_0$	$\langle 6\rangle_0$#	#	...
...	#	$\langle 0\rangle_0$	$\langle 1\rangle_0$	$\langle 2\rangle_0$	$\langle 3\rangle_0$	$\langle 4\rangle_0$	$\langle 5\rangle_0$	$\langle 6\rangle_0$	#	...

Fig. 1. The evolution of $\mathcal{A}'$ (time goes from bottom to top)

To prove the 3 propositions of theorem 51, let us consider two r-complete neighborhoods V and V', with $-x_n = \min V$, $-x'_n = \min V'$, $x_p = \max V$ and $x'_p = \max V'$.

1. If V' is complete then $x'_n > 0$. From propositions 51, 52, 53 and 54 we have, for some k big enough

$$L_{TR}(V) = L_{TR}([\![-x_n, x_p]\!]) = L_{TR}([\![-x'_p x_n, x'_p x_p]\!]) \subseteq L_{TR}([\![-2^k x_p x'_n, x_p x'_p]\!])$$
$$\subseteq L_{TR}([\![-x_p x'_n, x_p x'_p]\!]) \subseteq L_{TR}([\![-x'_n, x'_p]\!]) \subseteq L_{TR}(V')$$

2. This equality is a direct consequence of the previous inclusion.
3. If both are r-incomplete then $x_n = x'_n = 0$ and

$$L_{TR}(V) = L_{TR}([\![0, x_p]\!]) = L_{TR}([\![0, x'_p x_p]\!]) = L_{TR}([\![0, x'_p]\!]) = L_{TR}(V')$$

This ends the proof of theorem 51.

6 Conclusion

We have here proven an extension of Cole's equivalence [3] for one-dimensional neighborhoods. Since the usual simulation between neighborhoods is linear (in time), it was already known that linear, polynomial and exponential capabilities of the complete neighborhoods are all equal. We now know that the real-time capabilities are the same.

We have also studied an extension of one-way cellular automata: the CA working on r-incomplete neighborhoods. For these neighborhoods too we have obtained a strong equivalence. These two kinds of neighborhoods (complete and r-incomplete) are the only ones that enable a computation power equivalent to Turing machines. On the other neighborhoods (not r-complete) it is possible to prove that some cells will never interact with the origin (no matter what their state is) and that the position of these "invisible cells" is ultimately periodic. From this observation, it is easy to show that the languages that can be recognized on these neighborhoods are exactly the sub-class of Turing languages that do not depend on the letters on the "invisible cells". We can prove a constant speed-up theorem for this sub-class too.

We have therefore shown that language recognition by one-dimensional cellular automata can always be reduced to a recognition on either the usual neighborhood $\{-1, 0, 1\}$ or the one-way neighborhood $\{0, 1\}$.

In dimension 2 and above, it is known that the different neighborhoods aren't equivalent in terms of real-time recognition (of two-dimensional shapes) [14]. Although we can prove a theorem similar to theorem 21 that gives an exact description of any iteration of a two-dimensional neighborhood, it is unknown if we have a constant speed-up theorem.

References

1. Albert, J., Čulik II, K.: A simple universal cellular automaton and its one-way and totalistic version. Complex Systems **1** (1987) 1–16
2. Choffrut, C., Čulik II, K.: On real-time cellular automata and trellis automata. Acta Informatica **21** (1984) 393–407
3. Cole, S.N.: Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Transactions on Computers **C-18** (1969) 349–365
4. Čulik, K., Hurd, L.P., Yu, S.: Computation theoretic aspects of cellular automata. Phys. D **45** (1990) 357–378
5. Delorme, M., Mazoyer, J.: Reconnaissance parallèle des langages rationnels sur automates cellulaires plans. Theor. Comput. Sci. **281** (2002) 251–289

6. Fischer, P.C.: Generation of primes by one-dimensional real-time iterative array. Journal of the Assoc. Comput. Mach. **12** (1965) 388–394
7. Ibarra, O., Jiang, I.: Relating the power of cellular arrays to their closure properties. Theoretical Computer Science **57** (1988) 225–238
8. Martin, B.: A universal automaton in quasi-linear time with its s-n-m form. Theoretical Computer Science **124** (1994) 199–237
9. Mazoyer, J., Reimen, N.: A linear speed-up theorem for cellular automata. Theor. Comput. Sci. **101** (1992) 59–98
10. Smith III, A.R.: Simple computation-universal cellular spaces. J. ACM **18** (1971) 339–353
11. Smith III, A.R.: Real-time language recognition by one-dimensional cellular automata. Journal of the Assoc. Comput. Mach. **6** (1972) 233–253
12. Terrier, V.: Language recognizable in real time by cellular automata. Complex Systems **8** (1994) 325–336
13. Terrier, V.: Language not recognizable in real time by one-way cellular automata. Theoretical Computer Science **156** (1996) 281–287
14. Terrier, V.: Two-dimensional cellular automata and their neighborhoods. Theor. Comput. Sci. **312** (2004) 203–222
15. von Neumann, J.: Theory of Self-Reproducing Automata. University of Illinois Press, Urbana, IL, USA (1966)

On the Decidability of Temporal Properties of Probabilistic Pushdown Automata

Tomáš Brázdil*, Antonín Kučera**, and Oldřich Stražovský

Faculty of Informatics, Masaryk University,
Botanická 68a, 60200 Brno, Czech Republic
{brazdil, kucera, strazovsky}@fi.muni.cz

Abstract. We consider qualitative and quantitative model-checking problems for probabilistic pushdown automata (pPDA) and various temporal logics. We prove that the qualitative and quantitative model-checking problem for ω-regular properties and pPDA is in **2-EXPSPACE** and **3-EXPTIME**, respectively. We also prove that model-checking the qualitative fragment of the logic PECTL* for pPDA is in **2-EXPSPACE**, and model-checking the qualitative fragment of PCTL for pPDA is in **EXPSPACE**. Furthermore, model-checking the qualitative fragment of PCTL is shown to be **EXPTIME**-hard even for stateless pPDA. Finally, we show that PCTL model-checking is undecidable for pPDA, and PCTL$^+$ model-checking is undecidable even for stateless pPDA.

1 Introduction

In this paper we concentrate on a subclass of discrete probabilistic systems (see, e.g., [22]) that correspond to probabilistic sequential programs with recursive procedure calls. Such programs can conveniently be modeled by probabilistic pushdown automata (pPDA), where the stack symbols correspond to procedures and global data is stored in the finite control. This model is equivalent to probabilistic recursive state machines, or recursive Markov chains (see, e.g., [3, 16, 15]). An important subclass of pPDA are stateless pPDA, denoted pBPA[1]. In the non-probabilistic setting, BPA are often easier to analyze than general PDA (i.e., the corresponding algorithms are more efficient), but they still retain a reasonable expressive power which is sufficient, e.g., for modelling some problems of interprocedural dataflow analysis [12]. There is a close relationship between pBPA and stochastic context-free grammars. In fact, pBPA *are* stochastic context-free grammars, but they are seen from a different perspective in the setting of our

* Supported by the Grant Agency of the Czech Republic, grant No. 201/03/1161.
** Supported by the Alexander von Humboldt Foundation and by the 1M National Research Centre "Institute for Theoretical Computer Science (ITI)".

[1] This notation is borrowed from process algebra; stateless PDA correspond (in a well-defined sense) to processes of the so-called Basic Process Algebra.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 145–157, 2005.

paper. We consider the model-checking problem for pPDA/pBPA systems and properties expressible in probabilistic extensions of various temporal logics.

The State of the Art. Methods for automatic verification of probabilistic systems have so far been examined mainly for finite-state probabilistic systems. Model-checking algorithms for various (probabilistic) temporal logics like LTL, PCTL, PCTL*, probabilistic μ-calculus, etc., have been presented in [23, 19, 26, 18, 4, 10, 20, 11]. As for infinite-state systems, most works so far considered probabilistic lossy channel systems [21] which model asynchronous communication through unreliable channels [5, 1, 2, 6, 25]. The problem of deciding probabilistic bisimilarity over various classes of infinite-state probabilistic systems has recently been considered in [7]. Model-checking problems for pPDA and pBPA processes have been studied in [13]. In [13], it has been shown that the qualitative/quantitative random walk problem for pPDA is in **EXPTIME**, that the qualitative fragment of the logic PCTL is decidable for pPDA (but no upper complexity bound was given), and that the qualitative/quantitative model-checking problem for pPDA and a subclass of ω-regular properties definable by deterministic Büchi automata is also decidable. The reachability problem for pPDA and pBPA processes is studied in greater depth in [16], where it is shown that the qualitative reachability problem for pBPA is solvable in polynomial time, and a fast-converging algorithm for quantitative pPDA reachability is given.

Our Contribution. In this paper we continue the study initiated in [13]. We still concentrate mainly on clarifying the decidability/undecidability border for model-checking problems, but we also pay attention to complexity issues. Basic definitions together with some useful existing results are recalled in Section 2. As a warm-up, in Section 3 we show that both qualitative and quantitative model-checking problem for ω-regular properties and pPDA is decidable. More precisely, if ω-regular properties are encoded by Büchi automata, then the qualitative variant of the problem is in **2-EXPSPACE**, and the quantitative one is in **3-EXPTIME**. The proof is obtained by extending and modifying the construction for deterministic Büchi automata given in [13] so that it works for Muller automata. Note that the considered problems are known to be **PSPACE**-hard even for finite-state systems [26]. The core of the paper is Section 4. First we prove that model-checking general PCTL is undecidable for pPDA, and model-checking PCTL$^+$ is undecidable even for pBPA. Since the structure of formulae which are constructed in our proofs is relatively simple, our undecidability results hold even for fragments of these logics. From a certain point of view, these results are tight (see Section 4). Note that in the non-probabilistic case, the model-checking problems for logics like CTL, CTL*, or even the modal μ-calculus, are decidable for PDA. Our undecidability proofs are based on a careful arrangement of transition probabilities in the constructed pPDA so that various nontrivial properties can be encoded by specifying probabilities of certain events (which are expressible in PCTL or PCTL$^+$). We believe that these tricks might be applicable to other problems and possibly also to other models. In the light of

these undecidability results, it is sensible to ask if the model-checking problem is decidable at least for some natural fragments of probabilistic branching-time logics. We show that model-checking the *qualitative fragment* of the logic PECTL* is decidable for pPDA, and we give the **2-EXPSPACE** upper bound. For the qualitative fragment of PCTL we give the **EXPSPACE** upper bound. We also show that model-checking the qualitative fragment of PCTL is **EXPTIME**-hard even for pBPA processes. Our proof is a simple modification of the one given in [27] which shows **EXPTIME**-hardness of the model-checking problem for (non-probabilistic) CTL and PDA. Due to space constraints, formal proofs are omitted. We refer to [8] for technical details.

2 Preliminaries

For every alphabet Σ, the symbols Σ^* and Σ^ω denote the sets of all finite and infinite words over the alphabet Σ, respectively. The length of a given $w \in \Sigma^* \cup \Sigma^\omega$ is denoted $|w|$ (if $w \in \Sigma^\omega$ then we put $|w| = \omega$). For every $w \in \Sigma^* \cup \Sigma^\omega$ and every $0 \leq i < |w|$, the symbols $w(i)$ and w_i denote the i+1-th letter of w and the suffix of w which starts with $w(i)$, respectively. By writing $w(i)$ or w_i we implicitly impose the condition that the object exists.

Definition 1. *A* Büchi *automaton is a tuple* $\mathcal{B} = (\Sigma, B, \varrho, b_I, Acc)$*, where* Σ *is a finite* alphabet*,* B *is a finite set of* states*,* $\varrho \subseteq B \times \Sigma \times B$ *is a* transition relation *(we write* $b \xrightarrow{a} b'$ *instead of* $(b, a, b') \in \varrho$*),* b_I *is the* initial state*, and* $Acc \subseteq B$ *is a set of* accepting states*.*

A word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{B}$ if there is a run of $\mathcal{B}$ on w which visits some accepting state infinitely often. The set of all $w \in \Sigma^\omega$ which are accepted by $\mathcal{B}$ is denoted $L(\mathcal{B})$.

Definition 2. *A* probabilistic transition system *is a triple* $\mathcal{T} = (S, \rightarrow, Prob)$ *where* S *is a finite or countably infinite set of* states*,* $\rightarrow \subseteq S \times S$ *is a* transition relation*, and* $Prob$ *is a function which to each transition* $s \rightarrow t$ *of* $\mathcal{T}$ *assigns its probability* $Prob(s \rightarrow t) \in (0, 1]$ *so that for every* $s \in S$ *we have that* $\sum_{s \rightarrow t} Prob(s \rightarrow t) \in \{0, 1\}$*. (The sum above can be* 0 *if* s *does not have any outgoing transitions.)*

In the rest of this paper we write $s \xrightarrow{x} t$ instead of $Prob(s \rightarrow t) = x$. A *path* in $\mathcal{T}$ is a word $w \in S^* \cup S^\omega$ such that $w(i-1) \rightarrow w(i)$ for every $1 \leq i < |w|$. A *run* is a maximal path, i.e., a path which cannot be prolonged. The sets of all finite paths, all runs, and all infinite runs of $\mathcal{T}$ are denoted $FPath$, Run, and $IRun$, respectively[2]. Similarly, the sets of all finite paths, runs, and infinite runs that start in a given $s \in S$ are denoted $FPath(s)$, $Run(s)$, and $IRun(s)$, respectively.

Each $w \in FPath$ determines a *basic cylinder* $Run(w)$ which consists of all runs that start with w. To every $s \in S$ we associate the probabilistic space $(Run(s), \mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the σ-field generated by all basic cylinders $Run(w)$

[2] In this paper, $\mathcal{T}$ is always clear from the context.

where w starts with s, and $\mathcal{P} : \mathcal{F} \to [0,1]$ is the unique probability function such that $\mathcal{P}(Run(w)) = \Pi_{i=1}^{|w|-1} x_i$ where $w(i-1) \xrightarrow{x_i} w(i)$ for every $1 \leq i < |w|$ (if $|w| = 1$, we put $\mathcal{P}(Run(w)) = 1$).

The Logics PCTL, PCTL$^+$, PCTL*, PECTL*, and Their Qualitative Fragments. Let $Ap = \{a, b, c, \dots\}$ be a countably infinite set of *atomic propositions*. The syntax of PCTL* *state* and *path* formulae is given by the following abstract syntax equations (for simplicity, we omit the bounded 'until' operator from the syntax of path formulae).

$$\Phi ::= \mathtt{tt} \mid a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}^{\sim\varrho}\varphi$$
$$\varphi ::= \Phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathcal{X}\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

Here a ranges over Ap, $\varrho \in [0,1]$, and $\sim \in \{\leq, <, \geq, >\}$. The logic PCTL is a fragment of PCTL* where state formulae are defined as for PCTL* and path formulae are given by the equation $\varphi ::= \mathcal{X}\Phi \mid \Phi_1 \,\mathcal{U}\, \Phi_2$. The logic PCTL$^+$ is a fragment of PCTL* where the $\mathcal{X}$ and $\mathcal{U}$ operators in path formulae can be combined using Boolean connectives, but they cannot be nested. Finally, the logic PECTL* is an extension of PCTL* where only state formulae are introduced and have the following syntax:

$$\Phi ::= \mathtt{tt} \mid a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}^{\sim\varrho}\mathcal{B}$$

Here $\mathcal{B}$ is a Büchi automaton over an alphabet $2^{\{\Phi_1,\cdots,\Phi_n\}}$, where each Φ_i is a PECTL* formula.

Let $\mathcal{T} = (S, \to, Prob)$ be a probabilistic transition system, and let $\nu : Ap \to 2^S$ be a *valuation*. The semantics of PCTL* is defined below. State formulae are interpreted over S, and path formulae are interpreted over *IRun*. (Alternatively, path formulae could also be interpreted over *Run*. This would not lead to any problems, and our model-checking algorithms would still work after some minor modifications. We stick to infinite runs mainly for the sake of simplicity.)

$s \models^\nu \mathtt{tt}$		$w \models^\nu \Phi$	iff $w(0) \models^\nu \Phi$
$s \models^\nu a$	iff $s \in \nu(a)$	$w \models^\nu \neg\varphi$	iff $w \not\models^\nu \varphi$
$s \models^\nu \neg\Phi$	iff $s \not\models^\nu \Phi$	$w \models^\nu \varphi_1 \wedge \varphi_2$	iff $w \models^\nu \varphi_1$ and $w \models^\nu \varphi_2$
$s \models^\nu \Phi_1 \wedge \Phi_2$	iff $s \models^\nu \Phi_1$ and $s \models^\nu \Phi_2$	$w \models^\nu \mathcal{X}\varphi$	iff $w_1 \models^\nu \varphi$
$s \models^\nu \mathcal{P}^{\sim\varrho}\varphi$	iff $\mathcal{P}(\{w \in IRun(s) \mid w \models^\nu \varphi\}) \sim \varrho$	$w \models^\nu \varphi_1 \,\mathcal{U}\, \varphi_2$	iff $\exists j \geq 0 : w_j \models^\nu \varphi_2$ and $w_i \models^\nu \varphi_1$ for all $0 \leq i < j$

For PCTL, the semantics of path formulae is redefined to

$$w \models^\nu \mathcal{X}\Phi \quad \text{iff} \quad w(1) \models^\nu \Phi$$
$$w \models^\nu \Phi_1 \,\mathcal{U}\, \Phi_2 \quad \text{iff} \quad \exists j \geq 0 : w(j) \models^\nu \Phi_2 \text{ and } w(i) \models^\nu \Phi_1 \text{ for all } 0 \leq i < j$$

The semantics of a PECTL* formula $\mathcal{P}^{\sim\varrho}\mathcal{B}$, where $\mathcal{B}$ is a Büchi automaton over an alphabet $2^{\{\Phi_1,\cdots,\Phi_n\}}$, is defined as follows. First, we can assume that the semantics of the PECTL* formulae $\Phi_1, \cdots, \Phi_n$ has already been defined.

This means that for each $w \in IRun$ we can define an infinite word $w_{\mathcal{B}}$ over the alphabet $2^{\{\Phi_1,\cdots,\Phi_n\}}$ by $w_{\mathcal{B}}(i) = \{\Phi \in \{\Phi_1,\cdots,\Phi_n\} \mid w(i) \models^{\nu} \Phi\}$. For every state s, let $Run(s,\mathcal{B}) = \{w \in IRun(s) \mid w_{\mathcal{B}} \in L(\mathcal{B})\}$. We stipulate that $s \models^{\nu} \mathcal{P}^{\sim\varrho}\mathcal{B}$ iff $\mathcal{P}(Run(s,\mathcal{B})) \sim \varrho$.

The *qualitative fragments* of PCTL, PCTL*, and PECTL*, denoted qPCTL, qPCTL*, and qPECTL*, resp., are obtained by restricting the allowed operator/number combinations in $\mathcal{P}^{\sim\varrho}\varphi$ and $\mathcal{P}^{\sim\varrho}\mathcal{B}$ subformulae to '≤ 0' and '≥ 1', which can also be written as '$= 0$' and '$= 1$', resp. (Observe that '< 1', '> 0' are definable from '≤ 0', '≥ 1', and negation).

Probabilistic PDA. A *probabilistic pushdown automaton (pPDA)* is a tuple $\Delta = (Q, \Gamma, \delta, Prob)$ where Q is a finite set of *control states*, Γ is a finite *stack alphabet*, $\delta \subseteq Q \times \Gamma \times Q \times \Gamma^*$ is a finite *transition relation* (we write $pX \to q\alpha$ instead of $(p, X, q, \alpha) \in \delta$), and $Prob$ is a function which to each transition $pX \to q\alpha$ assigns its probability $Prob(pX \to q\alpha) \in (0, 1]$ so that for all $p \in Q$ and $X \in \Gamma$ we have that $\sum_{pX \to q\alpha} Prob(pX \to q\alpha) \in \{0, 1\}$.

A *pBPA* is a pPDA with just one control state. Formally, a pBPA is understood as a triple $\Delta = (\Gamma, \delta, Prob)$ where $\delta \subseteq \Gamma \times \Gamma^*$.

In the rest of this paper we adopt a more intuitive notation, writing $pX \xrightarrow{x} q\alpha$ instead of $Prob(pX \to q\alpha) = x$. The set $Q \times \Gamma^*$ of all configurations of Δ is denoted by $\mathcal{C}(\Delta)$. We also assume (w.l.o.g.) that if $pX \to q\alpha \in \delta$, then $|\alpha| \leq 2$. Given a configuration $pX\alpha$ of Δ, we call pX the *head* and α the *tail* of $pX\alpha$. To Δ we associate the probabilistic transition system $\mathcal{T}_{\Delta}$ where $\mathcal{C}(\Delta)$ is the set of states and the probabilistic transition relation is determined by $pX\beta \xrightarrow{x} q\alpha\beta$ iff $pX \xrightarrow{x} q\alpha$.

The model checking problem for pPDA configurations and any nontrivial class of properties is clearly undecidable for general valuations. Therefore, we restrict ourselves to *simple* valuations where the (in)validity of atomic propositions depends just on the current control state and the current symbol on top of the stack. Alternatively, we could consider *regular* valuations where the set of all configurations that satisfy a given atomic proposition is encoded by a finite-state automaton. However, regular valuations can be "encoded" into simple valuations by simulating the finite-state automata in the stack (see, e.g., [14]), and therefore they do not bring any extra expressive power.

Definition 3. *A valuation ν is* simple *if there is a function f_ν which assigns to every atomic proposition a subset of $Q \times \Gamma$ such that for every configuration $p\alpha$ and every $a \in Ap$ we have that $p\alpha \models^{\nu} a$ iff $\alpha = X\alpha'$ and $pX \in f_\nu(a)$.*

Random Walks on pPDA Graphs. Let $\mathcal{T} = (S, \to, Prob)$ be a probabilistic transition system. For all $s \in S$, $\mathcal{C}_1, \mathcal{C}_2 \subseteq S$, let $Run(s, \mathcal{C}_1 \mathcal{U} \mathcal{C}_2) = \{w \in Run(s) \mid \exists j \geq 0 : w(j) \in \mathcal{C}_2 \text{ and } w(i) \in \mathcal{C}_1 \text{ for all } 0 \leq i < j\}$. An instance of the *random walk problem* is a tuple $(s, \mathcal{C}_1, \mathcal{C}_2, \sim, \varrho)$, where $s \in S$, $\mathcal{C}_1, \mathcal{C}_2 \subseteq S$, $\sim \in \{\leq, <, \geq, >, =\}$, and $\varrho \in [0, 1]$. The question is if $\mathcal{P}(Run(s, \mathcal{C}_1 \mathcal{U} \mathcal{C}_2)) \sim \varrho$. In [13], it was shown that the random walk problem for pPDA processes and simple sets of configurations is decidable (a simple set is a set of the form $\bigcup_{pX \in \mathcal{H}} \{pX\alpha \mid \alpha \in \Gamma^*\}$ where $\mathcal{H}$ is a subset of $Q \times \Gamma$). More precisely, it was

shown that for a given tuple $(pX, \mathcal{C}_1, \mathcal{C}_2, \sim, \varrho)$, where $\mathcal{C}_1, \mathcal{C}_2$ are simple sets of configurations of a given pPDA system Δ, there is an efficiently constructible system of recursive quadratic equations such that the probability $\mathcal{P}(Run(pX, \mathcal{C}_1 \,\mathcal{U}\, \mathcal{C}_2))$ is the first component in the tuple of non-negative real values which form the least solution of the system. Thus, the relation $\mathcal{P}(Run(pX, \mathcal{C}_1 \,\mathcal{U}\, \mathcal{C}_2)) \sim \varrho$ can effectively be expressed in $(\mathbb{R}, +, *, \leq)$ by constructing a formula Φ saying that a given vector $\boldsymbol{x}$ is the least solution of the system and $\boldsymbol{x}(1) \sim \varrho$. Since the quantifier alternation depth in the constructed formula is fixed, it was concluded in [13] that the random walk problem for pPDA and simple sets of configurations is in **EXPTIME** by applying the result of [17]. Later, it was observed in [16] that the existential fragment of $(\mathbb{R}, +, *, \leq)$ is sufficient to decide the quantitative reachability problem for pPDA. This observation applies also to the random walk problem. Actually, it follows easily from the results of [13] just by observing that the existential (or universal) fragment of $(\mathbb{R}, +, *, \leq)$ is sufficient to decide whether $\mathcal{P}(Run(pX, \mathcal{C}_1 \,\mathcal{U}\, \mathcal{C}_2)) \sim \varrho$ when $\sim \in \{<, \leq\}$ (or $\sim \in \{>, \geq\}$, resp.). Since the existential and universal fragments of $(\mathbb{R}, +, *, \leq)$ are decidable in polynomial space [9], we obtain the following result which is used in our complexity estimations:

Lemma 1. *The random walk problem for pPDA processes and simple sets of configurations is in* ***PSPACE****.*

3 Model-Checking ω-Regular Properties

In this section we show that the qualitative and quantitative model-checking problems for pPDA and ω-regular properties represented by Büchi automata are in **2-EXPSPACE** and **3-EXPTIME**, respectively. For both of these problems there is a **PSPACE** lower complexity bound due to [26]. Our proof is a generalization of the construction for *deterministic* Büchi automata presented in [13]. We show that this construction can be extended to (deterministic) Muller automata, which have the same expressive power as general Büchi automata.

Definition 4. *A* Muller automaton *is a tuple $\mathcal{M} = (\Sigma, M, \varrho, m_I, \mathcal{F})$, where Σ is a finite* alphabet*, M is a finite set of* states*, $\varrho\colon M \times \Sigma \to M$ is a (total)* transition function *(we write $m \xrightarrow{a} m'$ instead of $\varrho(m, a) = m'$), m_I is the* initial state*, and $\mathcal{F} \subseteq 2^M$ is a set of* accepting sets*.*

For every infinite run v of $\mathcal{M}$, let $\inf(v)$ *be the set of all states which appear in v infinitely often. A word $w \in \Sigma^\omega$ is* accepted *by $\mathcal{M}$ if* $\inf(v) \in \mathcal{F}$*, where v is the (unique) run of $\mathcal{M}$ on w.*

For the rest of this section, we fix a pPDA $\Delta = (Q, \Gamma, \delta, Prob)$. We consider specifications given by Muller automata $\mathcal{M}$ having $Q \times \Gamma$ as their alphabet. Each infinite run w of Δ determines a unique word $v \in (Q \times \Gamma)^\omega$, where $v(i)$ is the head of $w(i)$ for every $i \in \mathbb{N}_0$. A run w of Δ is *accepted* by $\mathcal{M}$ if its associated word v is accepted by $\mathcal{M}$. For a given configuration pX, let $Run(pX, \mathcal{M})$ be the set of all runs of $IRun(pX)$ that are accepted by $\mathcal{M}$. Our aim is to show that the problem

if $\mathcal{P}(Run(pX, \mathcal{M})) \sim \varrho$ for given Δ, pX, $\mathcal{M}$, $\sim \in \{\leq, <, \geq, >\}$, and $\varrho \in [0, 1]$, is in **2-EXPTIME**. In the qualitative case, we derive the **EXPSPACE** upper bound.

Theorem 1. *The quantitative model-checking problem for pPDA processes and ω-regular properties represented by Muller automata is in **2-EXPTIME**, and the qualitative variant of this problem is in **EXPSPACE**.*

Corollary 1. *The quantitative model-checking problem for pPDA processes and ω-regular properties represented by Büchi automata is in **3-EXPTIME**, and the qualitative variant of this problem is in **2-EXPSPACE**.*

4 Model-Checking PCTL, PCTL*, and PECTL* Properties

We start by proving that model-checking PCTL is undecidable for pPDA processes, and model-checking PCTL$^+$ is undecidable for pBPA processes.

A *Minsky machine* with two counters is a finite sequence $\mathcal{C}$ of labeled instructions $\ell_1{:}inst_1, \cdots, \ell_n{:}inst_n$, where $n \geq 1$, $inst_n = \texttt{halt}$, and for every $1 \leq i < n$, the instruction $inst_i$ is of one of the following two types:

Type I. $c_r := c_r + 1;$ `goto` ℓ_j
Type II. if $c_r = 0$ `then goto` ℓ_j `else` $c_r := c_r - 1;$ `goto` ℓ_k

Here $r \in \{1, 2\}$ is a counter index. A *configuration* of $\mathcal{C}$ is a triple (ℓ_i, v_1, v_2), where $1 \leq i \leq n$ and $v_1, v_2 \in \mathbb{N}_0$ are counter values. Each configuration (ℓ_i, v_1, v_2) has a unique *successor* which is the configuration obtained by performing $inst_i$ on (ℓ_i, v_1, v_2). The *halting problem* for Minsky machines with two counters initialized to zero, i.e., the question whether $(\ell_1, 0, 0)$ eventually reaches a configuration of the form (ℓ_n, v_1, v_2), where $v_1, v_2 \in \mathbb{N}_0$, is undecidable [24].

Our aim is to reduce the halting problem for Minsky machines to the PCTL model checking problem for pPDA. Since a full proof is somewhat technical, we give just an intuitive explanation and refer to [8] for missing details.

Let $\mathcal{C}$ be a Minsky machine. We construct a pPDA system Δ, a process $p\alpha$ of Δ, and a PCTL formula ψ such that $\mathcal{C}$ halts iff $p\alpha \models \psi$. The formula ψ looks as follows:

$$\psi \;\equiv\; \mathcal{P}^{>0}((\mathit{check} \Rightarrow (\varphi_{state} \wedge \varphi_{zero} \wedge \varphi_{count}))\ \mathcal{U}\ \mathit{halt})$$

Here *check* and *halt* are atomic propositions, φ_{state} and φ_{zero} are qualitative formulae with just one $\mathcal{U}$ operator, and φ_{count} is a quantitative formula with just one $\mathcal{U}$ operator. So, φ_{count} is the only non-qualitative subformula in ψ. The stack content of the initial process $p\alpha$ corresponds to the initial configuration of $\mathcal{C}$. In general, a configuration (ℓ_i, v_1, v_2) is represented by the sequence $\ell_i A^{v_1} B^{v_2}$ of stack symbols, and individual configurations are separated by the # marker.

Starting from $p\alpha$, Δ tries to "guess" the successor configuration of $\mathcal{C}$ by pushing a sequence of stack symbols of the form $\ell_j A^{v_1} B^{v_2} \#$. The transitions

of Δ are arranged so that only strings of this syntactical form can be pushed. Transition probabilities do not matter here, the only important thing is that the "right" configuration can be guessed with a non-zero probability. After guessing the configuration (i.e., after pushing the symbol ℓ_j), Δ inevitably pushes one of the special "checking" symbols of the form (ℓ_i, ℓ_j, r, d), where $1 \leq i \leq n$, $r \in \{1, 2\}$ is a counter index, and $d \in \{-1, 0, 1\}$ a counter change (note that the previously pushed ℓ_j is in the second component of the checking symbol). An intuitive meaning of checking symbols is explained later. Let us just note that checking symbols correspond to instructions of $\mathcal{C}$ and hence not all tuples of the form (ℓ_i, ℓ_j, r, d) are necessarily checking symbols. Still, there can be several checking symbols with the same ℓ_j in the second component, and Δ can freely choose among them. Actually, the checking symbol is pushed *together* with ℓ_j, and hence the guessing phase ends in a "checking configuration" where the stack looks as follows: $(\ell_i, \ell_j, r, d)\ell_j A^{v_1} B^{v_2} \# \ldots$. The atomic proposition *check* is valid in exactly all checking configurations (i.e., configurations with a checking symbol on top of the stack), and the proposition *halt* is valid in exactly those configurations where ℓ_n (i.e., the label of `halt`) is on top of the stack.

From a checking configuration, Δ can either pop the checking symbol (note that the symbol ℓ_j appears at the top of the stack at this moment) and go on with guessing another configuration of $\mathcal{C}$, or perform other transitions so that the subformulae φ_{state}, φ_{zero}, and φ_{count} are (possibly) satisfied. Hence, the formula ψ says that there is a finite sequence of transitions from $p\alpha$ leading to a "halting" configuration along which all checking configurations satisfy the formulae φ_{state}, φ_{zero}, and φ_{count}. As can be expected, these three subformulae together say that the configuration of $\mathcal{C}$ just pushed to the stack is the successor of the configuration which was pushed previously. Let us discuss this part in greater detail.

First, let us clarify the meaning of checking symbols. Intuitively, each checking symbol corresponds to some computational step of $\mathcal{C}$. More precisely, the set of all checking symbols is the least set $\mathcal{T}$ such that for every $1 \leq i \leq n$ we have that

- if $inst_i \equiv c_r := c_r + 1;$ `goto` ℓ_j, then $(\ell_i, \ell_j, r, 1) \in \mathcal{T}$;
- if $inst_i \equiv$ if $c_r = 0$ `then goto` ℓ_j `else` $c_r := c_r - 1;$ `goto` ℓ_k, then $(\ell_i, \ell_j, r, 0), (\ell_i, \ell_k, r, -1) \in \mathcal{T}$.

Note that the checking symbol (ℓ_i, ℓ_j, r, d) which is pushed together with ℓ_j at the end of guessing phase is chosen freely. So, this symbol can also be chosen "badly" in the sense that ℓ_i is not the label of the previously pushed configuration, or the wrong branch of a Type II instruction is selected.

The formula φ_{state} intuitively says that we have chosen the "right" ℓ_i, and the subformula φ_{zero} says that if the checking symbol (ℓ_i, ℓ_j, r, d) claims the use of a Type II instruction and the counter c_r was supposed to be zero (i.e., $d = 0$), then the previously pushed configuration of $\mathcal{C}$ indeed has zero in the respective counter. In other words, φ_{zero} verifies that the right branch of a Type II instruction was selected.

The most interesting part is the subformula φ_{count}, which says that the counter values in the current and the previous configuration have changed accordingly to (ℓ_i, ℓ_j, r, d). For example, if $r = 0$ and $d = -1$, then the subformula φ_{count} is valid in the considered checking configuration iff the first counter was changed by -1 and the second counter remained unchanged.

To get some intuition on how this can be implemented, let us consider a simplified version of this problem. Let us assume that we have a configuration of the form $pA^m\#A^n\#$. Our aim is to set up the transitions of $pA^m\#A^n\#$ and to construct a PCTL formula φ so that $pA^m\#A^n\# \models \varphi$ iff $m = n$ (this indicates how to check if a counter remains unchanged). Let

$$\begin{array}{llllll} pA \xrightarrow{1/2} qA, & qA \xrightarrow{1} q\varepsilon, & rA \xrightarrow{1/2} sA, & tA \xrightarrow{1/2} t\varepsilon, & sA \xrightarrow{1} sA, \\ pA \xrightarrow{1/2} tA, & q\# \xrightarrow{1} r\varepsilon, & rA \xrightarrow{1/2} r\varepsilon, & tA \xrightarrow{1/2} uA, & uA \xrightarrow{1} uA \\ & & & t\# \xrightarrow{1} sA, & \end{array}$$

By inspecting possible runs of $pA^m\#A^n\#$, one can easily confirm that the probability that a run of $pA^m\#A^n\#$ hits a configuration having sA as its head is exactly

$$\frac{1}{2} \cdot (1 - \frac{1}{2^n}) \; + \; \frac{1}{2} \cdot \frac{1}{2^m} \quad = \quad \frac{1}{2} - \frac{1}{2^{n+1}} + \frac{1}{2^{m+1}}$$

Let p_{sA} be an atomic proposition which is valid in (exactly) all configurations having sA as their head. Then $pA^m\#A^n\# \models \mathcal{P}^{=\frac{1}{2}}(\mathtt{tt}\,\mathcal{U}\,p_{sA})$ iff $m = n$.

One can argue that formulae where some probability is required to be *equal* to some value are seldom used in practice. However, it is easy to modify the proof so that for every subformula of the form $\mathcal{P}^{\sim\varrho}\varphi$ which is employed in the proof we have that $\sim$ is $>$ and ϱ is a "simple" rational like $1/2$ or $1/4$. We refer to [8] for details.

Finally, let us note that our undecidability result is tight with respect to the nesting depth of $\mathcal{U}$. The fragment of PCTL where the $\mathcal{U}$ operators are not nested (and the $\mathcal{X}$ operators can be nested to an arbitrary depth) is decidable by applying the results of [13]. In our undecidability proof we use a PCTL formula where the nesting depth of $\mathcal{U}$ is 2 (PCTL formulae where the $\mathcal{U}$ operators are not nested have the nesting depth 1).

Theorem 2. *The model-checking problem for pPDA processes and the logic PCTL is undecidable. Moreover, the undecidability result holds even for the fragment of PCTL where the nesting depth of $\mathcal{U}$ is at most two, and for all subformulae of the form $\mathcal{P}^{\sim\varrho}\varphi$ we have that $\sim$ is $>$.*

The proof of Theorem 2 does not carry over to pBPA processes. The decidability of PCTL for pBPA processes is one of the challenges which are left open for future work. Nevertheless, we were able to show that model-checking PCTL$^+$ (and in fact a simple fragment of this logic) is undecidable even for pBPA. The structure of the construction is similar as in Theorem 2, but the proof contains new tricks invented specifically for pBPA. In particular, the consistency of counter values in consecutive configurations is verified somewhat differently. This is the only place where we use the expressive power of PCTL$^+$.

Theorem 3. *The model-checking problem for pBPA processes and the logic PCTL$^+$ is undecidable. More precisely, the undecidability result holds even for a fragment of PCTL$^+$ where the nesting depth of $\mathcal{U}$ is at most two, and for all subformulae of the form $\mathcal{P}^{\sim\varrho}\varphi$ we have that $\sim$ is $>$.*

Now we prove that the model-checking problem for pPDA and the logic qPECTL* is decidable and belongs to **2-EXPSPACE**. For the logic qPCTL, our algorithm only needs singly exponential space.

Let us fix a pPDA $\Delta = (Q, \Gamma, \delta, Prob)$, qPECTL* formula τ, and a simple valuation ν. The symbol $Cl(\tau)$ denotes the set of all subformulae of τ, and $Acl(\tau) \subseteq Cl(\tau)$ is the subset of all "automata subformulae" of the form $\mathcal{P}^{=x}\mathcal{B}$.

Let $\varphi \equiv \mathcal{P}^{=x}\mathcal{B} \in Acl(\tau)$ where $\mathcal{B}$ is a Büchi automaton over an alphabet $\Sigma_\varphi = 2^{\{\Phi_1,\ldots,\Phi_n\}}$. Then there is a (deterministic) Muller automaton $\mathcal{M}_\varphi = (\Sigma_\varphi, M_\varphi, \varrho_\varphi, m^I_\varphi, \mathcal{F}_\varphi)$ whose size is at most exponential in the size of $\mathcal{B}$ such that $L(\mathcal{M}_\varphi) = L(\mathcal{B})$. In our constructions we use $\mathcal{M}_\varphi$ instead of $\mathcal{B}$.

The intuition behind our proof is that we extend each configuration of Δ with some additional information that allows to determine the (in)validity of each subformula of τ in a given configuration just by inspecting the head of the configuration. Our algorithm computes a sequence of *extensions of* Δ that are obtained from Δ by augmenting stack symbols and transition rules with some information about subformulae of τ. These extensions are formally introduced in our next definition. For notation convenience, we define $St = \Pi_{\varphi\in Acl(\tau)} 2^{Q\times M_\varphi}$. For every $v \in St$, the projection of v onto a given $\varphi \in Acl(\tau)$ is denoted $v(\varphi)$. Note that $v(\varphi)$ is a set of pairs of the form (q, m), where $q \in Q$ and $m \in M_\varphi$.

Definition 5. *We say that a pPDA $\Delta' = (Q, \Gamma', \delta', Prob')$ is an* extension of *Δ if and only if $\Gamma' = St \times \Gamma \times St$ (elements of Γ' are written as (uXv), where $u, v \in St$ and $X \in \Gamma$), and the outgoing transitions of every $p(uXv) \in Q \times \Gamma'$ satisfy the following:*

1. *if $pX \xrightarrow{x} q\varepsilon$, then $p(uXv) \xrightarrow{x} q\varepsilon$;*
2. *if $pX \xrightarrow{x} qY$, then there is a* unique *$z \in St$ such that $p(uXv) \xrightarrow{x} q(zYv)$;*
3. *if $pX \xrightarrow{x} qYZ$, then there are* unique *$z, w \in St$ such that $p(uXv) \xrightarrow{x} q(zYw)(wZv)$;*
4. *$p(uXv)$ has no other outgoing transitions.*

Note that due to 2. and 3., a given Δ can have many extensions. However, all of these extensions have the same set of control states and the same stack alphabet. Moreover, the part of $\mathcal{T}_{\Delta'}$ which is reachable from a configuration $p(u_1X_1v_1)\cdots(u_nX_nv_n)$ is isomorphic to the part of $\mathcal{T}_\Delta$ reachable from the configuration $pX_1\cdots X_n$.

Definition 6. *Let $\Delta' = (Q, \Gamma', \delta', Prob')$ be an extension of Δ. For each $\varphi \in Cl(\tau)$ we define a set $\mathcal{C}_\varphi \subseteq Q \times \Gamma'$ inductively as follows:*

- *if $\varphi = a$ where $a \in Ap$, then $\mathcal{C}_\varphi = \{p(uXv) \mid pX \in f_\nu(a)$ and $u, v \in St\}$*
- *if $\varphi = \psi \wedge \xi$, then $\mathcal{C}_\varphi = \mathcal{C}_\psi \cap \mathcal{C}_\xi$*

– *if* $\varphi = \neg\psi$, *then* $\mathcal{C}_\varphi = (Q \times \Gamma') \setminus \mathcal{C}_\psi$
– *if* $\varphi = \mathcal{P}^{=x}\mathcal{B}$, *then* $\mathcal{C}_\varphi = \{p(uXv) \mid u, v \in St \text{ and } (p, m^I_\varphi) \in u(\varphi)\}$

For each $\varphi \in Acl(\tau)$ *we define a Muller automaton* $\mathcal{M}'_\varphi = (\Sigma'_\varphi, M_\varphi, \varrho'_\varphi, m^I_\varphi, \mathcal{F}_\varphi)$, *which is a modification of the automaton* $\mathcal{M}_\varphi$, *as follows:* $\Sigma'_\varphi = Q \times \Gamma'$, *and* $m \xrightarrow{h} m'$ *is a transition of* ϱ'_φ *iff there is* $A \in \Sigma_\varphi$ *such that* $m \xrightarrow{A} m'$ *is a transition of* ϱ_φ *and* $h \in (\bigcap_{\psi \in A} \mathcal{C}_\psi) \setminus \bigcup_{\psi \notin A} \mathcal{C}_\psi$. *Note that* $\mathcal{M}'_\varphi$ *is again deterministic.*

Let Δ' be an extension of Δ. The symbol $[s, p(uXv)\bullet]_\varphi$ denotes the probability that a run of $Run(p(uXv))$ is accepted by $\mathcal{M}'_\varphi$ where the initial state of $\mathcal{M}'_\varphi$ is changed to s. Furthermore, the symbol $[s, p(uXv)q, t]_\varphi$ denotes the probability that a run w of $Run(p(uXv))$ hits the configuration $q\varepsilon$, i.e., w is of the form $w'\, q\varepsilon$, so that $\mathcal{M}'_\varphi$ initiated in s moves to t after reading the heads of all configurations in w'.

Intuitively, the sets $\mathcal{C}_\varphi$ are supposed to encode exactly those configurations where φ holds (the information which is relevant for the (in)validity of φ should have been accumulated in the symbol at the top of the stack). However, this works only under some "consistency" assumptions, which are formalized in our next definition (see also Lemma 2 below).

Definition 7. *Let* $\varphi \in Acl(\tau)$ *and let* Δ' *be an extension of* Δ. *We say that a symbol* $(uXv) \in \Gamma'$ *is* φ-consistent *in* Δ' *iff the following conditions are satisfied:*

– *if* $\varphi \equiv \mathcal{P}^{=1}\mathcal{B}$, *then* $u(\varphi) = \{(p, s) \mid [s, p(uXv)\bullet]_\varphi + \sum_{(q,t) \in v(\varphi)} [s, p(uXv)q, t]_\varphi = 1\}$
– *if* $\varphi \equiv \mathcal{P}^{=0}\mathcal{B}$, *then* $u(\varphi) = \{(p, s) \mid [s, p(uXv)\bullet]_\varphi + \sum_{(q,t) \notin v(\varphi)} [s, p(uXv)q, t]_\varphi = 0\}$

We say that a configuration $p(u_1X_1v_1)\cdots(u_nX_nv_n)$ *is* φ-*consistent in* Δ' *iff* $(u_iX_iv_i)$ *is* φ-*consistent in* Δ' *for every* $1 \leq i \leq n$, *and* $v_i = u_{i+1}$ *for every* $1 \leq i < n$.

An extension Δ' *of* Δ *is* φ-*consistent iff for all transitions of the form* $p(uXv) \xrightarrow{x} q(zYv)$ *and* $p(uXv) \xrightarrow{x} q(zYw)(wZv)$ *of* Δ' *we have that* $q(zYv)$ *and* $q(zYw)(wZv)$ *are* φ-*consistent in* Δ', *respectively.*

It is important to realize that the conditions of Definition 7 are effectively verifiable, because, e.g., the condition $[s, p(uXv)\bullet]_\varphi + \sum_{(q,t) \in v(\varphi)} [s, p(uXv)q, t]_\varphi = 1$ can effectively be translated into $(\mathbb{R}, +, *, \leq)$ using the construction of Theorem 1 and the results on random walks of [13] which were recalled in Section 2. We refer to [8] for details and complexity estimations.

A $v \in St$ is *terminal* iff for each $\varphi \in Acl(\tau)$ we have that if $\varphi = \mathcal{P}^{=1}\mathcal{B}$ then $v(\varphi) = \emptyset$, and if $\varphi = \mathcal{P}^{=0}\mathcal{B}$ then $v(\varphi) = Q \times M_\varphi$.

Lemma 2. *Let* $\varphi \in Cl(\tau)$, *and let* Δ' *be an extension of* Δ *which is* ψ-*consistent for all* $\psi \in Acl(\varphi)$. *Let* $p(u_1X_1v_1)\cdots(u_nX_nv_n)$ *(where* $n \geq 1$*) be a configuration of* Δ' *which is* ψ-*consistent in* Δ' *for each* $\psi \in Acl(\varphi)$, *and where* v_n *is terminal. Then* $pX_1 \cdots X_n \models \varphi$ *iff* $p(u_1X_1v_1) \in \mathcal{C}_\varphi$.

Lemma 3. *Let pX be a configuration of Δ. Then there exists an extension Δ^τ of Δ which is φ-consistent for each $\varphi \in Acl(\tau)$, and a configuration $p(uXv)$ which is φ-consistent in Δ^τ for each $\varphi \in Acl(\tau)$. Moreover, Δ^τ and $p(uXv)$ are effectively constructible is space which is* doubly exponential *in the size of τ (if τ is a PCTL formula, then the space complexity is only* singly exponential *in the size of τ) and* singly exponential *in the size of Δ.*

An immediate corollary to Lemma 2 and Lemma 3 is the following:

Theorem 4. *The model-checking problems for pPDA processes and the logics qPECTL* and qPCTL are in **2-EXPSPACE** and **EXPSPACE**, respectively.*

Finally, let us note that the construction presented in [27] which shows **EXPTIME**-hardness of the model-checking problem for the logic CTL and PDA processes can be adapted so that it works for (non-probabilistic) BPA[3]. This idea carries over to the probabilistic case after some trivial modifications. Thus, we obtain the following:

Theorem 5. *The model-checking problem for pBPA processes and the logic qPCTL is **EXPTIME**-hard.*

References

1. P.A. Abdulla, C. Baier, S.P. Iyer, and B. Jonsson. Reasoning about probabilistic channel systems. In *Proceedings of CONCUR 2000*, vol. 1877 of *LNCS*, pp. 320–330. Springer, 2000.
2. P.A. Abdulla and A. Rabinovich. Verification of probabilistic systems with faulty communication. In *Proceedings of FoSSaCS 2003*, vol. 2620 of *LNCS*, pp. 39–53. Springer, 2003.
3. R. Alur, K. Etessami, and M. Yannakakis. Analysis of recursive state machines. In *Proceedings of CAV 2001*, vol. 2102 of *LNCS*, pp. 207–220. Springer, 2001.
4. A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Proceedings of CAV'95*, vol. 939 of *LNCS*, pp. 155–165. Springer, 1995.
5. C. Baier and B. Engelen. Establishing qualitative properties for probabilistic lossy channel systems: an algorithmic approach. In *Proceedings of 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, vol. 1601 of *LNCS*, pp. 34–52. Springer, 1999.
6. N. Bertrand and Ph. Schnoebelen. Model checking lossy channel systems is probably decidable. In *Proceedings of FoSSaCS 2003*, vol. 2620 of *LNCS*, pp. 120–135. Springer, 2003.
7. T. Brázdil, A. Kučera, and O. Stražovský. Deciding probabilistic bisimilarity over infinite-state probabilistic systems. In *Proceedings of CONCUR 2004*, vol. 3170 of *LNCS*, pp. 193–208. Springer, 2004.
8. T. Brázdil, A. Kučera, and O. Stražovský. On the decidability of temporal properties of probabilistic pushdown automata. Technical report FIMU-RS-2005-01, Faculty of Informatics, Masaryk University, 2005.

[3] This observation is due to Mayr (Private communication, July 2004).

9. J. Canny. Some algebraic and geometric computations in PSPACE. In *Proceedings of STOC'88*, pp. 460–467. ACM Press, 1988.
10. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *JACM*, 42(4):857–907, 1995.
11. J.M. Couvreur, N. Saheb, and G. Sutre. An optimal automata approach to LTL model checking of probabilistic systems. In *Proceedings of LPAR 2003*, vol. 2850 of *LNCS*, pp. 361–375. Springer, 2003.
12. J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proceedings of FoSSaCS'99*, vol. 1578 of *LNCS*, pp. 14–30. Springer, 1999.
13. J. Esparza, A. Kučera, and R. Mayr. Model-checking probabilistic pushdown automata. In *Proceedings of LICS 2004*, pp. 12–21. IEEE, 2004.
14. J. Esparza, A. Kučera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. *I&C*, 186(2):355–376, 2003.
15. K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic systems. Technical Report, School of Informatics, U. of Edinburgh, 2005.
16. K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations. In *Proceedings of STACS'2005*, LNCS. Springer, 2005. To Appear.
17. D. Grigoriev. Complexity of deciding Tarski algebra. *Journal of Symbolic Computation*, 5(1–2):65–108, 1988.
18. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
19. S. Hart and M. Sharir. Probabilistic temporal logic for finite and bounded models. In *Proceedings of POPL'84*, pp. 1–13. ACM Press, 1984.
20. M. Huth and M.Z. Kwiatkowska. Quantitative analysis and model checking. In *Proceedings of LICS'97*, pp. 111–122. IEEE, 1997.
21. S.P. Iyer and M. Narasimha. Probabilistic lossy channel systems. In *Proceedings of TAPSOFT'97*, vol. 1214 of *LNCS*, pp. 667–681. Springer, 1997.
22. M.Z. Kwiatkowska. Model checking for probability and time: from theory to practice. In *Proceedings of LICS 2003*, pp. 351–360. IEEE, 2003.
23. D. Lehman and S. Shelah. Reasoning with time and chance. *I&C*, 53:165–198, 1982.
24. M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
25. A. Rabinovich. Quantitative analysis of probabilistic lossy channel systems. In *Proceedings of ICALP 2003*, vol. 2719 of *LNCS*, pp. 1008–1021. Springer, 2003.
26. M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of FOCS'85*, pp. 327–338. IEEE, 1985.
27. I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proceedings of FST&TCS'2000*, vol. 1974 of *LNCS*, pp. 127–138. Springer, 2000.

Deciding Properties of Contract-Signing Protocols

Detlef Kähler, Ralf Küsters, and Thomas Wilke

Institut für Informatik und Praktische Mathematik,
Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
{kaehler, kuesters, wilke}@ti.informatik.uni-kiel.de

Abstract. We show that for infinite transition systems induced by cryptographic protocols in the Rusinowitch/Turuani style certain fundamental branching properties are decidable. As a consequence, we obtain that crucial properties of contract-signing protocols such as balance are decidable.

1 Introduction

There has been intensive research on the automatic analysis of cryptographic protocols in the recent past (see, e.g., [8, 13] for an overview) which led to industrial-strength debugging tools (see, e.g., [2]). One of the central results of the area is that security of cryptographic protocols is decidable when analyzed w.r.t. a finite number of sessions, without a bound on the message size, and in presence of the so-called Dolev-Yao intruder (see, e.g., [15, 1, 5, 14]). This result (and all the related ones) is, however, restricted to security properties such as authenticity and secrecy, which are reachability properties of the transition system associated with a given protocol: Is a state, in which the intruder possesses a certain secret, such as a session key, reachable? In contrast, crucial properties required of contract-signing and related protocols [10, 3, 4, 17], for instance abuse-freeness [10] and balance [6], are properties of the structure of the transition system associated with a protocol. Balance, for instance, requires that in no stage of a protocol run, the intruder or a dishonest party has both a strategy to abort the run and a strategy to successfully complete the run and thus obtain a valid contract.

The main goal of this paper is to show that the central result mentioned above extends to branching properties, such as balance, and similar properties of contract-signing protocols. In other words, we want to show that these branching properties are decidable w.r.t. a finite number of sessions, without a bound on the message size, and in presence of the so-called Dolev-Yao intruder. This can potentially lead to fully automatic analyzers for contract-signing protocols that are much more precise than the existing ones, which consider only drastically scaled down finite-state versions of the protocols in question.

The protocol and intruder model that we suggest to use is a "finite session" version of a model proposed in [6].[1] It contains different features im-

[1] Our technical exposition though is closer to the term-rewriting approach [15] than to the multi-set rewriting framework employed in [6].

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 158–169, 2005.

portant for contract-signing protocols which are absent in the models for authentication and key exchange protocols referred to above. First, as in [6], we include private contract signatures in our model as an important example of a cryptographic primitive used in contract-signing protocols, such as the protocol proposed in [10]. Second, as in [6], we model write-protected channels which are *not* under the control of the intruder. In this paper, we call these channels *secure channels* for simplicity, although this notion is also used in cryptography with a different meaning. Third, for protocols expressed in our model we explicitly define the induced transition systems. These transition systems have infinitely many states and are infinitely branching, but have paths of bounded length, and allow us to state crucial properties of contract-signing properties.

Our main technical result is that for the transition systems induced by cryptographic protocols certain game-theoretic properties—expressing the existence of certain strategies of the intruder—are decidable. From this we obtain that balance is decidable for contract-signing protocols. We also show that reachability properties slightly more general than those mentioned above are decidable and conclude that other important properties of contract signing protocols, namely effectiveness and fairness, are decidable as well.

The basic technique used in our proofs is the same as the one first introduced in [15], where they show that to find an attack on a protocol only reasonably small substitutions have to be considered. For our framework, we extend and modify this idea appropriately.

In several papers, contract-signing and related protocols have been analyzed using finite-state model checkers (see, e.g., [16, 12]). Due to the restriction to a finite state set, the Dolev-Yao intruder is, however, only approximated. A much more detailed model has been considered by Chadha et al. [6], who analyzed the contract-signing protocol proposed by Garay, Jakobsson, and MacKenzie [10], even taking into account an unbounded number of sessions. However, the analysis was carried out by hand and without tool support. As mentioned, our model is the "finite session" version of the model by Chadha et al. Hence, our results show that when restricted to a finite number of sessions, the analysis carried out in [6] can be fully automated (given a specification of the protocols) without resorting to finite-state models as in [16, 12]. Drielsma and Mödersheim [9] apply an automatic tool originally intended for authentication and key exchange protocols in the analysis of the Asokan-Shoup-Waidner (ASW) protocol [3]. Their analysis is, however, restricted to reachability properties as branching properties cannot be handled by their tool. Also, secure channels are not modeled explicitly in that paper.

Structure of This Paper. In Section 2, we introduce our protocol and intruder model, with an example provided in Section 3. In Section 4, we present our main technical result, stating that certain game-theoretic properties of transition systems induced by cryptographic protocols are decidable. In Section 5, this result is applied to contract-signing protocols. Due to the page limit, private contract signatures and reachability properties are only dealt with in our technical report [11], which also includes the full proofs of our results.

2 The Protocol and Intruder Model

As mentioned in the introduction, in essence, our model is the "finite session" version of the model proposed in [6]. When it comes to the technical exposition, our approach is, however, inspired by the term-rewriting approach of [15] rather than the multi-set rewriting approach of [6].

In our model, a protocol is a finite set of principals and every principal is a finite tree, which represents all possible behaviours of the principal. Each edge of such a tree is labeled by a rewrite rule, which describes the receive-send action that is performed when the principal takes this edge in a run of the protocol.

When a principal carries out a protocol, it traverses its tree, starting at the root. In every node, the principal takes its current input, chooses one of the edges leaving the node, matches the current input with the left-hand side of the rule the edge is labeled with, sends out the message which is determined by the right-hand side of the rule, and moves to the node the chosen edge leads to. Whether or not a principal gets an input and which input it gets is determined by the secure channel and the intruder (see below), who receives every message sent by a principal, can use all the messages he has received to construct new messages, and can provide input messages to any principal he wants—this is the usual Dolev-Yao model (see, e.g., [15]).

The above is very similar to the approach in [15]. There are, however, three important ingredients that are not present in [15]: secure channels, explicit branching structure, and certain cryptographic primitives relevant to contract-signing protocols, such as private contract signatures. In the following, we deal with secure channels and the branching structure. Private contract signatures are only dealt with in the technical report [11].

Secure Channels. Unlike in the standard Dolev-Yao model, in our model the input of a principal may not only come from the intruder but also from a so-called secure channel. While a secure channel is not read-protected (the intruder can read the messages written onto this channel), the intruder does not control this channel. That is, he cannot delay, duplicate, or remove messages, or write messages onto this channel under a fake identity (unless he has corrupted a party).

Branching Structure. As mentioned in the introduction, unlike authentication and key-exchange protocols, properties of contract-signing and related protocols cannot be stated as reachability properties, i.e., in terms of single runs of a protocol alone. One rather has to consider branching properties. We therefore describe the behavior of a protocol as an infinite-state transition system which comprises all runs of a protocol. To be able to express properties of contract-signing protocols we distinguish several types of transitions: there are intruder transitions (just as in [15]), there are ε-transitions, which can be used to model that a subprotocol is spawned without waiting for input from the in-

truder, and secure channel transitions, which model communication via secure channels.

2.1 Terms and Messages

We have a finite set $\mathcal{V}$ of variables, a finite set $\mathcal{A}$ of atoms, a finite set $\mathcal{K}$ of public and private keys, an infinite set $\mathcal{A}_I$ of intruder atoms, and a finite set $\mathcal{N}$ of principal addresses. All of them are assumed to be disjoint.

The set $\mathcal{K}$ is partitioned into a set $\mathcal{K}_{pub}$ of public keys and a set $\mathcal{K}_{priv}$ of private keys. There is a bijective mapping $\cdot^{-1}\colon \mathcal{K} \to \mathcal{K}$ which assigns to every public key the corresponding private key and to every private key its corresponding public key.

Typically, the set $\mathcal{A}$ contains names of principals, atomic symmetric keys, and nonces (i.e., random numbers generated by principals). We note that we will allow non-atomic symmetric keys as well. The atoms in $\mathcal{A}_I$ are the nonces, symmetric keys, etc. the intruder may generate. The elements of $\mathcal{N}$ are used as addresses of principals in secure channels.

We define two kinds of terms by the following grammar, namely *plain terms* and *secure channel terms*:

$$\begin{aligned} \textit{plain-terms} ::= {} & \mathcal{V} \mid \mathcal{A} \mid \mathcal{A}_I \mid \langle \textit{plain-terms}, \textit{plain-terms} \rangle \mid \{\textit{plain-terms}\}^s_{\textit{plain-terms}} \mid \\ & \{\textit{plain-terms}\}^a_{\mathcal{K}} \mid \mathsf{hash}(\textit{plain-terms}) \mid \mathsf{sig}_{\mathcal{K}}(\textit{plain-terms}) \\ \textit{sec-terms} ::= {} & \mathsf{sc}(\mathcal{N}, \mathcal{N}, \textit{plain-terms}) \\ \textit{terms} ::= {} & \textit{plain-terms} \mid \textit{sec-terms} \mid \mathcal{N} \end{aligned}$$

Plain terms, secure channel terms, and terms without variables (i.e., ground terms) are called *plain messages*, *secure channel messages*, and *messages*, respectively. As usual, $\langle t, t' \rangle$ is the pairing of t and t', the term $\{t\}^s_{t'}$ stands for the symmetric encryption of t by t' (note that the key t' may be any plain term), $\{t\}^a_k$ is the asymmetric encryption of t by k, the term $\mathsf{hash}(t)$ stands for the hash of t, and $\mathsf{sig}_k(t)$ is the signature on t which can be verified with the public key k.

A secure channel term of the form $\mathsf{sc}(n, n', t)$ stands for feeding the secure channel from n to n' with t. A principal may only generate such a term if he knows n and t (but not necessarily n'). This guarantees that a principal cannot impersonate other principals on the secure channel. Knowing n grants access to secure channels with sender address n.

A *substitution* assigns terms to variables. The *domain* of a substitution is denoted by $\mathsf{dom}(\sigma)$ and defined by $\mathsf{dom}(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$. Substitutions are required to have finite domains and it is required that $\sigma(x)$ is a ground term for each $x \in \mathsf{dom}(\sigma)$. Given two substitutions σ and σ' with disjoint domains, their union $\sigma \cup \sigma'$ is defined in the obvious way. Given a term t, the term $t\sigma$ is obtained from t by simultaneously substituting each variable x occurring in t by $\sigma(x)$.

2.2 Principals and Protocols

Principal rules are of the form $R \Rightarrow S$ where R is a term or ε and S is a term.

A *rule tree* $\Pi = (V, E, r, \ell)$ is a finite tree rooted at $r \in V$ where ℓ maps every edge $(v, v') \in E$ of Π to a principal rule $\ell(v, v')$. A rule tree $\Pi = (V, E, r, \ell)$ is called a *principal* if every variable occurring on the right-hand side of a principal rule $\ell(v, v')$ also occurs on the left-hand side of $\ell(v, v')$ or on the left-hand side of a principal rule on the path from r to v.

For $v \in V$, we write $\Pi{\downarrow}v$ to denote the subtree of Π rooted at v. For a substitution σ, we write $\Pi\sigma$ for the principal obtained from Π by substituting all variables x occurring in the principal rules of Π by $\sigma(x)$.

A *protocol* $P = ((\Pi_1, \ldots, \Pi_n), \mathcal{I})$ consists of a finite set of principals and a finite set $\mathcal{I}$ of messages, the *initial intruder knowledge*. We require that each variable occurs in the rules of only one principal, i.e., different principals must have disjoint sets of variables. We assume that intruder atoms, i.e., elements of $\mathcal{A}_I$, do not occur in P.

2.3 Intruder

Given a set $\mathcal{I}$ of general messages, the (infinite) set $d(\mathcal{I})$ of general messages the intruder can derive from $\mathcal{I}$ is the smallest set satisfying the following conditions:

1. $\mathcal{I} \subseteq d(\mathcal{I})$.
2. *Composition and decomposition*: If $m, m' \in d(\mathcal{I})$, then $\langle m, m' \rangle \in d(\mathcal{I})$. Conversely, if $\langle m, m' \rangle \in d(\mathcal{I})$, then $m \in d(\mathcal{I})$ and $m' \in d(\mathcal{I})$.
3. *Symmetric encryption and decryption*: If $m, m' \in d(\mathcal{I})$, then $\{m\}^s_{m'} \in d(\mathcal{I})$. Conversely, if $\{m\}^s_{m'} \in d(\mathcal{I})$ and $m' \in d(\mathcal{I})$, then $m \in d(\mathcal{I})$.
4. *Asymmetric encryption and decryption*: If $m \in d(\mathcal{I})$ and $k \in d(\mathcal{I}) \cap \mathcal{K}$, then $\{m\}^a_k \in d(\mathcal{I})$. Conversely, if $\{m\}^a_k \in d(\mathcal{I})$ and $k^{-1} \in d(\mathcal{I})$, then $m \in d(\mathcal{I})$.
5. *Hashing*: If $m \in d(\mathcal{I})$, then $\mathsf{hash}(m) \in d(\mathcal{I})$.
6. *Signing*: If $m \in d(\mathcal{I}), k^{-1} \in d(\mathcal{I}) \cap \mathcal{K}$, then $\mathsf{sig}_k(m)$. (The signature contains the public key but can only be generated if the corresponding private key is known.)
7. *Writing onto and reading from the secure channel:* If $m \in d(\mathcal{I})$, $n \in d(\mathcal{I}) \cap \mathcal{N}$, and $n' \in \mathcal{N}$, then $\mathsf{sc}(n, n', m) \in d(\mathcal{I})$. If $\mathsf{sc}(n, n', m) \in d(\mathcal{I})$, then $m \in d(\mathcal{I})$.
8. *Generating fresh constants*: $\mathcal{A}_I \subseteq d(\mathcal{I})$.

Each of the above rules only applies when the resulting expression is a term according to the grammar stated above. For instance, a hash of a secure channel term is not a term, so rule 5 does not apply when m is of the form $\mathsf{sc}(n, n', m')$.

Intuitively, $n \in d(\mathcal{I}) \cap \mathcal{N}$ means that the intruder has corrupted the principal with address n and therefore can impersonate this principal when writing onto the secure channel. Also, the intruder can extract m from $\mathsf{sc}(n, n', m)$ since, just as in [6], the secure channel is not read-protected. (However, our results hold independent of whether or not the secure channel is read-protected.)

2.4 The Transition Graph Induced by a Protocol

We define the transition graph $\mathcal{G}_P$ induced by a protocol P and start with the definition of the states and the transitions between these states.

A *state* is of the form $((\Pi_1, \ldots, \Pi_n), \sigma, \mathcal{I}, \mathcal{S})$ where

1. σ is a substitution,
2. for each i, Π_i is a rule tree such that $\Pi_i\sigma$ is a principal,
3. $\mathcal{I}$ is a finite set of messages, the *intruder knowledge*, and
4. $\mathcal{S}$ is a finite multi-set of secure channel messages, the *secure channel*.

The idea is that when the transition system gets to such a state, then the substitution σ has been performed, the accumulated intruder knowledge is what can be derived from $\mathcal{I}$, the secure channels hold the messages in $\mathcal{S}$, and for each i, Π_i is the "remaining protocol" to be carried out by principal i. This also explains why $\mathcal{S}$ is a multi-set: messages sent several times should be delivered several times.

Given a protocol $P = ((\Pi_1, \ldots, \Pi_n), \mathcal{I})$ the *initial state of* P is set to be $((\Pi_1, \ldots, \Pi_n), \sigma, \mathcal{I}, \emptyset)$ where σ is the substitution with empty domain.

We have three kinds of transitions: intruder, secure channel, and ε-transitions. In what follows, let $\Pi_i = (V_i, E_i, r_i, \ell_i)$ and $\Pi_i' = (V_i', E_i', r_i', \ell_i')$ denote rule trees. We define under which circumstances there is a transition

$$((\Pi_1, \ldots, \Pi_n), \sigma, \mathcal{I}, \mathcal{S}) \xrightarrow{\tau} ((\Pi_1', \ldots, \Pi_n'), \sigma', \mathcal{I}', \mathcal{S}')$$

with τ an appropriate label.

1. *Intruder transitions:* The above transition with label i, m, I exists if there exists $v \in V_i$ with $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = R \Rightarrow S$, and a substitution σ'' of the variables in $R\sigma$ such that
   (a) $m \in d(\mathcal{I})$,
   (b) $\sigma' = \sigma \cup \sigma''$,
   (c) $R\sigma' = m$,
   (d) $\Pi_j' = \Pi_j$ for every $j \neq i$, $\Pi_i' = \Pi_i{\downarrow}v$,
   (e) $\mathcal{I}' = \mathcal{I} \cup \{S\sigma'\}$,
   (f) $\mathcal{S}' = \mathcal{S}$ if $S \neq \mathsf{sc}(\cdot, \cdot, \cdot)$, and $\mathcal{S}' = \mathcal{S} \cup \{S\sigma'\}$ otherwise.
   This transition models that principal i reads the message m from the intruder (i.e., the public network).
2. *Secure channel transitions:* The above transition with label i, m, sc exists if there exists $v \in V_i$ with $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = R \Rightarrow S$, and a substitution σ'' of the variables in $R\sigma$ such that $m \in \mathcal{S}$, (b)–(e) from 1., and $\mathcal{S}' = \mathcal{S} \setminus \{m\}$ if $S \neq \mathsf{sc}(\cdot, \cdot, \cdot)$, and $\mathcal{S}' = (\mathcal{S} \setminus \{m\}) \cup \{S\sigma'\}$ otherwise.
   This transition models that principal i reads message m from the secure channel.
3. *ε-transitions:* The above transition with label i exists if there exists $v \in V_i$ with $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = \varepsilon \Rightarrow S$ such that $\sigma' = \sigma$ and (d), (e), (f) from above.
   This transition models that i performs a step where neither a message is read from the intruder nor from the secure channel.

If $q \xrightarrow{\tau} q'$ is a transition where the first component of the label τ is i, then the transition is called an *i-transition* and q' an *i-successor* of q.

Given a protocol P, the *transition graph $\mathcal{G}_P$ induced by P* is the tuple (S_P, E_P, q_P) where q_P is the initial state of P, S_P is the set of states reach-

able from q_P by a sequence of transitions, and E_P is the set of all transitions among states in S_P. Formally, a transition $q \xrightarrow{\tau} q'$ is a tuple (q, τ, q').

We write $q \in \mathcal{G}_P$ if q is a state in $\mathcal{G}_P$ and $q \xrightarrow{\tau} q' \in \mathcal{G}_P$ if $q \xrightarrow{\tau} q'$ is a transition in $\mathcal{G}_P$.

Remark 1. The transition graph $\mathcal{G}_P$ of P is a DAG since by performing a transition the size of the first component of a state decreases. While the graph may be infinite branching, the maximal length of a path in this graph is bounded by the total number of edges in the principals Π_i of P.

For a state q, we denote the subgraph of $\mathcal{G}_P$ consisting of all states reachable from q by $\mathcal{G}_{P,q}$.

3 Modeling the Originator of the ASW Protocol

To demonstrate that our framework can actually be used to analyze contract-signing protocols, we show how the originator of the Asokan-Shoup-Waidner (ASW) protocol [3] can be modeled. In a similar fashion, other contract-signing protocols, such as the Garay-Jakobsson-MacKenzie protocol [10], can be dealt with.

3.1 Overview of the Protocol

Our informal description of the ASW protocol follows [16] (see this work or [3] for more details). For ease in notation, we will write $\mathsf{sig}[m, k]$ instead of $\langle m, \mathsf{sig}_k(m)\rangle$.

The ASW protocol enables two principals O (originator) and R (responder) to obtain each other's commitment on a previously agreed contractual text, say text, with the help of a trusted third party T, which, however, is only invoked in case of problems. In other words, the ASW protocol is an optimistic two-party contract-signing protocol.

There are two kinds of valid contracts specified in the ASW protocol: the standard contract, $\langle \mathsf{sig}[m_O, k_O], N_O, \mathsf{sig}[m_R, k_R], N_R\rangle$, and the replacement contract, $\mathsf{sig}[\langle \mathsf{sig}[m_O, k_O], \mathsf{sig}[m_R, k_R]\rangle, k_T]$, where k_T is the key of the trusted third party, $m_O = \langle k_O, k_R, k_T, \mathsf{text}, \mathsf{hash}(N_O)\rangle$, and $m_R = \langle m_O, \mathsf{hash}(N_R)\rangle$. The keys k_O, k_R, and k_T are used for identifying the principals. Note that a signed contractual text ($\mathsf{sig}[\mathsf{text}, k_O]$ or $\mathsf{sig}[\mathsf{text}, k_R]$) is not considered a valid contract.

The ASW protocol consists of three subprotocols: the exchange, abort, and resolve protocol. However, we can describe every principal—O, R, and T—in terms of a single tree as introduced in Section 2.2.

The basic idea of the exchange protocol is that O first indicates his/her interest to sign the contract. To this end, O hashes a nonce N_O and signs it together with text and the keys of the principals involved. The resulting message is the message m_O from above. By sending it to R, O commits to the contract. Then, similarly, R indicates his/her interest to sign the contract by hashing a nonce N_R and signing it together with text and the keys of the involved principals. This is the message m_R from above. By sending it to O, R commits to the contract. Finally, first O and then R reveal N_O and N_R, respectively. This is why a standard contract is only valid if N_O and N_R are included.

If, after O has sent the first message, R does not respond, O may contact T to abort. At any point, if one of O and R does not respond, the other may contact T to resolve. In case the protocol is successfully resolved, the replacement contract $\mathsf{sig}[\langle m_O, m_R \rangle, k_T]$ is issued. While this version of the contract only contains the message indicating O's and R's intention to sign the contract (and neither N_O nor N_R), the signature of T validates the contract.

In the next subsection, the model of O is presented. The models for R and T as well as the security properties for the ASW protocol can be found in the technical report [11].

3.2 The Principal *O*

The principal O is defined by the tree Π_O depicted in Figure 1 where the numbers stand for the principal rules defined below. Rules 1, 2, and 3 belong to the exchange protocol, rules 4, 5, and 6 belong to the abort protocol, and rules 7, 8, and 9 belong to the resolve protocol.

Exchange Protocol. The actions performed in the exchange protocol have informally been discussed above.

Abort Protocol. If, after the first step of the exchange protocol, O does not get an answer back from R, the principal O may start the abort protocol, i.e., send an abort request via a secure channel to T (rule 4). Then, T will either confirm the abort of the protocol by returning an abort token—in this case O will continue with rule 5—or send a resolve token—in this case O will continue with rule 6. (The trusted third party T sends a resolve token if R previously contacted T to resolve the protocol run.)

Resolve Protocol. If after rule 2, i. e., after sending N_O, the principal O does not get an answer back from R, then O can start the resolve protocol by sending a resolve request to T via the secure channel (rule 7). After that, depending on the answer returned from T (which again will return an abort or resolve token), one of the rules 8 or 9 is performed.

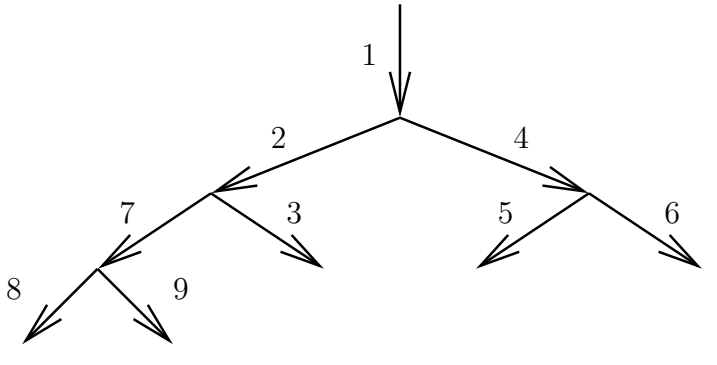


Fig. 1. The Originator O

We now present the principal rules for O where the numbering corresponds to the one in Figure 1. Any occurrence of _ should be substituted by a new fresh variable, that is, the term which is matched is not used afterwards.

1. $\varepsilon \Rightarrow me_1$ where
$$me_1 = \mathsf{sig}[me_2, k_O] \quad \text{and} \quad me_2 = \langle k_O, k_R, k_T, \mathsf{text}, \mathsf{hash}(N_O)\rangle\,.$$
2. $\mathsf{sig}[me_3, k_R] \Rightarrow N_O$ where $me_3 = \langle me_1, \mathsf{hash}(x)\rangle\,.$
3. $x \Rightarrow \mathsf{OHasValidContract}$.
4. $\varepsilon \Rightarrow \mathsf{sc}(O, T, ma_1)$ where $ma_1 = \mathsf{sig}[\langle \mathsf{aborted}, me_1\rangle\,, k_O]$.
5. $\mathsf{sc}(T, O, ma_2) \Rightarrow \mathsf{OHasValidContract}$ where
$$ma_2 = \mathsf{sig}[\langle me_1, me_4\rangle\,, k_T] \quad \text{and} \quad me_4 = \mathsf{sig}[\langle me_1, \_\rangle\,, k_R].$$
6. $\mathsf{sc}(T, O, \mathsf{sig}[\langle \mathsf{aborted}, ma_1\rangle\,, k_T]) \Rightarrow \mathsf{OHasAbortToken}$.
7. $\varepsilon \Rightarrow \mathsf{sc}(O, T, \langle me_1, \mathsf{sig}[me_3, k_R]\rangle)$.
8. $\mathsf{sc}(T, O, \mathsf{sig}[\langle \mathsf{aborted}, ma_1\rangle\,, k_T]) \Rightarrow \mathsf{OHasAbortToken}$.
9. $\mathsf{sc}(T, O, mr_1) \Rightarrow \mathsf{OHasValidContract}$ where
$$mr_1 = \mathsf{sig}[\langle me_1, mr_2\rangle\,, k_T] \quad \text{and} \quad mr_2 = \mathsf{sig}[\langle me_1, \_\rangle\,, k_R].$$

4 Main Result

As indicated in the introduction, our main result states that for the transition graphs induced by cryptographic protocols certain game-theoretic properties—expressing the existence of certain strategies of the intruder—are decidable. In what follows we formalize this.

We first define the notion of a strategy graph, which captures that the intruder has a way of acting such that regardless of how the other principals act, he achieves a certain goal, where goal in our context means that a state will be reached where the intruder can derive certain constants and cannot derive others.

A q-strategy graph $\mathcal{G}_q$ is a sub-transition system of $\mathcal{G}_P$ where q is the initial state of $\mathcal{G}_q$ and such that for all states q' in $\mathcal{G}_q$, the following conditions, which are explained below, are satisfied.

1. If $q' \xrightarrow{j} q'' \in \mathcal{G}_P$, then $q' \xrightarrow{j} q'' \in \mathcal{G}_q$ for every j and q''.
2. If $q' \xrightarrow{j,m,\mathsf{sc}} q'' \in \mathcal{G}_P$, then $q' \xrightarrow{j,m,\mathsf{sc}} q'' \in \mathcal{G}_q$ for every m, j, and q''.
3. If $q' \xrightarrow{j,m,I} q'' \in \mathcal{G}_q$ and $q' \xrightarrow{j,m,I} q''' \in \mathcal{G}_P$, then $q' \xrightarrow{j,m,I} q''' \in \mathcal{G}_q$ for every m, j, q'', and q'''.

The first condition says that every ε-transition of the original transition system must be present in the strategy graph; this is because the intruder should not be able to prevent a principal from performing an ε-rule. The second condition is similar: the intruder should not be able to block secure channels. The third condition says that although the intruder can choose to send a particular message to a particular principal, he cannot decide which transition this principal uses (if the message matches two rules).

A *strategy property* is a tuple $((C_1, C_1'), \ldots, (C_s, C_s'))$, where $C_i, C_i' \subseteq \mathcal{A} \cup \mathcal{K} \cup \mathcal{N}$. A state q satisfies $((C_1, C_1'), \ldots, (C_s, C_s'))$ if there exist q-strategy graphs $\mathcal{G}_1, \ldots, \mathcal{G}_s$ such that every $\mathcal{G}_i$ satisfies (C_i, C_i'), where $\mathcal{G}_i$ satisfies (C_i, C_i') if for all leaves v_i of $\mathcal{G}_i$ all elements from C_i can be derived by the intruder and all elements from C_i' cannot.

The decision problem STRATEGY asks, given a protocol P and a strategy property $((C_1, C_1'), \ldots, (C_s, C_s'))$, whether there exists a state q that satisfies the property.

Theorem 1. STRATEGY *is decidable.*

To prove this theorem, we show that given a possibly large state q and large q-strategy graphs satisfying the properties, we can reduce the sizes of the state and the strategy graphs, i.e., the sizes of the substitutions in the state and the graphs. For this purpose, we need to deal with all substitutions in all of the strategy graphs at the same time. The challenge is then to guarantee that the reduced strategy graphs are in fact strategy graphs, i.e., satisfy the required conditions. We make use of the fact that the intruder can generate new constants.

5 Balance

In this section, we formalize a fundamental property of contract-signing protocols, namely balance, and explain why this property is decidable in our framework.

As in [6] and most other works on the formal analysis of contract-signing protocols we formulate balance for two-party optimistic contract-signing protocols (see, however, [7]). Besides the two parties (the contractual partners), say A and B, who want to sign the contract, a trusted third party T is involved in the protocol, and is consulted in case a problem occurs. We assume in what follows that the protocols are modeled in such a way that if A has a valid contract, it writes the atom AHasValidContract into the network, similarly for B.

We formulate balance under the assumption that one of the contractual parties is dishonest and the other party is honest, i.e., follows the protocol. The actions of the dishonest party are performed by the intruder, and hence, are arbitrary. All private information the dishonest party has, such as private keys and addresses for the secure channel, are given to the intruder as part of his initial knowledge. The trusted third party is assumed to be honest. We denote the honest party by H and the dishonest party by I.

Informally speaking, balance for H means that at no stage of the protocol run I has both a strategy to prevent H from obtaining a valid contract with I on the previously agreed contractual text, say text, and a strategy to obtain a valid contract with H on text.

In order to formulate that I has a strategy to obtain a valid contract, we assume that the protocol description contains a specific principal, a *watch dog*, which when receiving a valid contract for I outputs IHasValidContract. Such

a watch dog can easily be derived from a protocol specification. For the ASW protocol, for example, the watch dog would be a principal with two edges leaving the root of the tree. The first edge would be labeled with the rule

$$\langle \mathsf{sig}[m_O, k_H], N_H, \mathsf{sig}[m_I, k_I], x\rangle \Rightarrow \mathsf{IHasValidContract}$$

where $m_H = \langle k_H, k_I, k_T, \mathsf{text}, \mathsf{hash}(N_H)\rangle$ and $m_I = \langle m_H, \mathsf{hash}(x)\rangle$. The nonce N_H could be replaced by a variable y in case no specific instance of H is considered, and hence, no specific nonce generated in such an instance. This rule indicates that I has a standard contract with H. Analogously, the rule for the second edge would check whether I has a replacement contract with H.

Now, formally a protocol P is *not* balanced for H if P satisfies the strategy property $((\{\mathsf{IHasValidContract}\}, \emptyset), (\emptyset, \{\mathsf{HHasValidContract}\}))$. By Theorem 1, this can be decided. In [11], following [6], we also consider a formulation of balance involving abort tokens.

6 Conclusion

In this paper we have shown that effectiveness, fairness, and balance, a branching property of contract-signing protocols, is decidable when there is no bound on the message size for a Dolev-Yao intruder and when there are only a finite number of sessions. This extends known results on the decidability of reachability problems for cryptographic protocols in a natural way. Our approach is fairly generic; it should therefore be a good starting point for analyzing other game-theoretic properties of cryptographic protocols. From a practical point of view, our result may also be a good starting point for developing more precise analyzers for contract-signing protocols.

Acknowledgment. We thank the anonymous referees for their comments, which helped, in particular, to improve the presentation in Section 5.

References

1. R.M. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *TCS*, 290(1):695–740, 2002.
2. A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In CAV 2002, volume 2404 of *LNCS*, pages 349–353. Springer, 2002.
3. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.
4. M. Ben-Or, O. Goldreich, S. Micali, and R.L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.
5. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 667–681. Springer-Verlag, 2001.

6. R. Chadha, M.I. Kanovich, and A.Scedrov. Inductive methods and contract-signing protocols. In *CCS 2001*, pages 176–185. ACM Press, 2001.
7. R. Chadha, S. Kremer, and A. Scedrov. Formal analysis of multi-party contract signing. In *CSFW-17*, pages 266–279. IEEE Computer Society Press, 2004.
8. H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technolog*, 2002.
9. P. H. Drielsma and S. Mödersheim. The ASW protocol revisited: A unified view. In *Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA)*, 2004.
10. J.A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO'99*, volume 1666 of *LNCS*, pages 449–466. Springer-Verlag, 1999.
11. D. Kähler, R. Küster, T. Wilke. Deciding Properties of Contract-Signing Protocols. Technical Report IFI 0409, CAU Kiel, 2004. Available from http://www.informatik.uni-kiel.de/reports/2004/0409.html
12. S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *Computer Security Foundations Workshop 2002 (CSFW 2002)*. IEEE Computer Society, 2002.
13. C. Meadows. Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends. *IEEE Journal on Selected Areas in Communication*, 21(1):44–54, January 2003.
14. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001*, pages 166–175. ACM Press, 2001.
15. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science*, 299(1–3):451–475, 2003.
16. V. Shmatikov and J.C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.
17. J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 55–61. IEEE Computer Society Press, 1996.

Polylog-Time Reductions Decrease Dot-Depth

Christian Glaßer

University at Würzburg, Am Hubland, 97074 Würzburg, Germany

Abstract. We study (i) regular languages that are polylog-time reducible to languages in dot-depth 1/2 and (ii) regular languages that are polylog-time decidable. For both classes we provide

- forbidden-pattern characterizations, and
- characterizations in terms of regular expressions.

Both classes are decidable. A language is in class (ii) if and only if the language and its complement are in class (i). We apply our observations and obtain three consequences.

1. Gap theorems for balanced regular-leaf-language definable classes $\mathcal{C}$ and $\mathcal{D}$:
   (a) Either $\mathcal{C}$ is contained in NP, or $\mathcal{C}$ contains coUP.
   (b) Either $\mathcal{D}$ is contained in P, or $\mathcal{D}$ contains UP or coUP.
   We extend both theorems such that no promise classes are involved. Formerly, such gap theorems were known only for the unbalanced approach.
2. Polylog-time reductions can tremendously decrease dot-depth complexity (despite that these reductions cannot count). We construct languages of arbitrary dot-depth that are reducible to languages in dot-depth 1/2.
3. Unbalanced starfree leaf-languages can be much stronger than balanced ones. We construct starfree regular languages L_n such that the balanced leaf-language class of L_n is NP, but the unbalanced leaf-language class of L_n contains level n of the unambiguous alternation hierarchy. This demonstrates the power of unbalanced computations.

1 Introduction

Regular languages are described by regular expressions. These consist of single letters which are combined by three types of operations: Boolean operations, concatenation, and iteration. If we forbid iteration, then these restricted regular expressions define *starfree regular languages* (starfree languages for short). The class of these languages is denoted by SF. It is a subclass of REG, the class of regular languages.

Throughout the paper all automata are deterministic. Call a minimal finite automaton *permutationfree* if the following holds for all $n \geq 2$, all words w, and all states s: If s on input w^n leads to s, then already s on input w leads to s. While regular languages are accepted by finite automata, starfree languages are accepted by *permutationfree finite automata* [21, 16] (permutationfree automata

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 170–181, 2005.

for short). By definition, if any state of a permutationfree automaton has a loop w^n, then it already has a loop w. So permutationfree automata cannot count the number of certain events modulo some $m > 1$ (e.g., number of letters a modulo 2). We say that permutationfree automata cannot do *modulo-counting*. For this reason, McNaughton and Papert [16] call such automata *counterfree*. However, permutationfree automata can do *threshold-counting*. This means to count the number of events up to some threshold (e.g., counting whether the number of a's is 0, 1, 2, or ≥ 3).

Dot-Depth Hierarchy. The most interesting aspect of starfree languages is the *dot-depth hierarchy* which was introduced by Brzozowski and Cohen [9, 6]. The dot-depth measures the complexity of starfree languages. It counts the minimal number of nested alternations between Boolean operations and concatenation that is needed to define a language. Classes of the dot-depth hierarchy consist of languages that have the same dot-depth. Fix a finite alphabet that has at least two letters (the hierarchy collapses for unary alphabets). Define $\mathrm{Pol}(\mathcal{C})$, the polynomial closure of $\mathcal{C}$, to be $\mathcal{C}$'s closure under finite (possibly empty) union and finite (possibly empty) concatenation. Let $\mathrm{BC}(\mathcal{C})$ be the Boolean closure of $\mathcal{C}$. For $n \geq 0$ define the classes (levels) of the dot-depth hierarchy:

$$\begin{aligned} \mathcal{B}_0 &\stackrel{df}{=} \{L \mid L \text{ is finite or cofinite}\} \\ \mathcal{B}_{n+\frac{1}{2}} &\stackrel{df}{=} \mathrm{Pol}(\mathcal{B}_n) \\ \mathcal{B}_{n+1} &\stackrel{df}{=} \mathrm{BC}(\mathcal{B}_{n+\frac{1}{2}}) \end{aligned}$$

The dot-depth of a language L is defined as the minimal m such that $L \in \mathcal{B}_m$ where $m = n/2$ for some integer n. At first glance, the definition of levels $n+1/2$ looks a bit artificial. The reason for this definition is of historic nature: Originally, Brzozowski and Cohen were interested in the full levels $\mathcal{B}_n$ and therefore, defined the dot-depth hierarchy in this way. Later Pin and Weil [20] considered both, the levels $\mathcal{B}_n$ and their polynomial closure. To be consistent with Brzozowski and Cohen, they extended the dot-depth hierarchy by half levels $\mathcal{B}_{n+1/2}$.

By definition, all levels of the dot-depth hierarchy are closed under union and it is known that they are closed under intersection, under taking inverse morphisms, and under taking residuals [19, 1, 20]. The dot-depth hierarchy is strict [7, 23] and it exhausts the class of starfree languages [10].

Does there exist an algorithm that computes the dot-depth of a starfree language? This question is known as the *dot-depth problem*. Today, more than 30 years after it was discovered by Brzozowski and Cohen, it is still an open problem. Most researchers consider the dot-depth problem as one of the most difficult problems in automata theory.

The problem remains hard, if we ask for decidability of single classes of the dot-depth hierarchy. However, we know that the 4 lowest classes are decidable. $\mathcal{B}_0$ is decidable for trivial reasons. Pin and Weil [20] proved decidability of $\mathcal{B}_{1/2}$, Knast [14] proved decidability of $\mathcal{B}_1$, and Glaßer and Schmitz [11] proved decidability of $\mathcal{B}_{3/2}$. Other levels are not known to be decidable, but it is widely believed that they are. The decidability results for $\mathcal{B}_{1/2}$ and $\mathcal{B}_{3/2}$ share an inter-

esting property: Both classes, $\mathcal{B}_{1/2}$ and $\mathcal{B}_{3/2}$, have *forbidden-pattern characterizations*. This means that a language belongs to $\mathcal{B}_{1/2}$ if and only if its minimal automaton does *not* have a certain pattern. This implies decidability of $\mathcal{B}_{1/2}$.

Restricted Modulo-Counting. We come back to the result by Schützenberger [21] and McNaughton and Papert [16]: The class of languages accepted by permutationfree automata (or counterfree automata) is exactly the class of starfree languages. This tells us that starfree languages cannot do modulo-counting. For instance the set of even-length words is not starfree. However, there do exist starfree subsets of all even-length words. This is possible, since sometimes counting can be reformulated as local properties. For instance, $L = (ab)^*$ is starfree, since a word belongs to L if and only if it starts with a, ends with b, and neither has aa nor has bb as factor. This example adjusts our intuition: Starfree languages cannot do arbitrary modulo-counting, but modulo-counting in a restricted sense is possible. We will exploit this phenomenon.

Leaf Languages. The concept of leaf languages was independently introduced by Bovet, Crescenzi, and Silvestri [5] and Vereshchagin [24]. Let M be any nondeterministic polynomial-time Turing machine such that every computation path stops and outputs one letter. $M(x)$ denotes the computation tree on input x. Call a nondeterministic polynomial-time Turing machine M *balanced* if there exists a polynomial-time computable function that on input (x, n) computes the n-th path of $M(x)$. Let $\text{leafstring}^M(x)$ be the concatenation of all outputs of $M(x)$. For any language B, let $\text{Leaf}_u^p(B)$ be the class of languages L such that there exists an (unbalanced) nondeterministic polynomial-time Turing machine M as above such that for all x,

$$x \in L \iff \text{leafstring}^M(x) \in B.$$

If we assume M to be a balanced, nondeterministic polynomial-time Turing machine, then this defines the class $\text{Leaf}_b^p(B)$. For any class $\mathcal{C}$ let $\text{Leaf}_u^p(\mathcal{C}) = \bigcup_{B\in\mathcal{C}} \text{Leaf}_u^p(B)$ and $\text{Leaf}_b^p(\mathcal{C}) = \bigcup_{B\in\mathcal{C}} \text{Leaf}_b^p(B)$. If $\mathcal{C} \subseteq \text{REG}$ and $\mathcal{D} = \text{Leaf}_u^p(\mathcal{C})$, then we say that $\mathcal{D}$ is an *unbalanced regular-leaf-language definable class*. Analogously define *balanced regular-leaf-language definable classes*. Since in this paper, $\mathcal{C}$ will always be a subclass of REG, we will use the term (un)balanced leaf-language definable class as abbreviation.

Connection Between Hierarchies. Starfree languages have a very nice connection with complexity theory. In the concept of leaf languages, classes of the dot-depth hierarchy correspond exactly to classes of the polynomial-time hierarchy. For $n \geq 1$,

$$\text{Leaf}_b^p(\mathcal{B}_{n-1/2}) = \text{Leaf}_u^p(\mathcal{B}_{n-1/2}) = \Sigma_{\text{n}}^{\text{P}}.$$

This connection allows a translation of knowledge about dot-depth classes into knowledge about complexity classes. Here the forbidden-pattern characterizations come into play. They allow us to identify gaps between leaf-language definable complexity classes. We sketch this elegant technique with help of an example which goes back to Pin and Weil [20] and Borchert, Kuske, and Stephan [4].

Consider $\mathcal{B}_{1/2}$. If B belongs to $\mathcal{B}_{1/2}$, then, by the mentioned correspondence, B's leaf-language class is contained in NP. Otherwise, B does not belong to $\mathcal{B}_{1/2}$. So B's minimal automaton contains the forbidden pattern [20]. This can be exploited to show that B's leaf-language class is powerful enough to solve coUP [4]. Therefore, between NP and coUP there are no unbalanced leaf-language definable classes. We call this a *gap theorem*.

Another gap theorem is known for P. Borchert [3][1] showed that the following holds for any unbalanced leaf-language definable class $\mathcal{C}$: Either $\mathcal{C}$ is in P, or $\mathcal{C}$ contains at least one of the following classes: NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ for p prime.

Balanced vs. Unbalanced. We are interested in gap theorems similar to the ones showed by Borchert [3] and Borchert, Kuske, and Stephan [4]. However, this time we consider *balanced* leaf languages which show a new situation.

For the unbalanced case the following holds: For any regular B in dot-depth 1/2, $\mathrm{Leaf}_u^p(B)$ is contained in NP; for any regular B not in dot-depth 1/2, $\mathrm{Leaf}_u^p(B)$ is not contained in NP (unless coUP $\subseteq$ NP). This does not hold anymore for the balanced case. It is possible to construct a starfree language C (Example 1) such that C is outside dot-depth 1/2, but $\mathrm{Leaf}_b^p(C) \subseteq$ NP. Even more, there is a regular D that is not starfree, but still $\mathrm{Leaf}_b^p(D) \subseteq$ NP (e.g., $D = (AA)^*$ for any alphabet A). In this sense, the classes of the dot-depth hierarchy do not fit to balanced leaf languages. The reason for this becomes clear with help of a theorem discovered by Bovet, Crescenzi, and Silvestri [5] and Vereshchagin [24].

$$B \leq^{plt} C \;\Leftrightarrow\; \text{for all oracles } O,\ \mathrm{Leaf}_b^p(B)^O \subseteq \mathrm{Leaf}_b^p(C)^O$$

So $\mathrm{Leaf}_b^p(B) \subseteq$ NP not only for all B in dot-depth 1/2, but also for all B that are polylog-time reducible to a language in dot-depth 1/2.

Our Contribution. We start the paper with a study of the power of polylog-time reductions restricted to regular languages. More precisely, we study

1. $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$, the class of regular languages that are polylog-time reducible to a language in dot-depth 1/2, and
2. $\mathrm{PLT} \cap \mathrm{REG}$, the class of regular languages that are polylog-time decidable.

For both classes we give forbidden-pattern characterizations, and characterizations in terms of regular expressions. This immediately implies decidability of the classes. Moreover, we show that both classes are strongly connected:

$$\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{co}\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG} = \mathrm{PLT} \cap \mathrm{REG}.$$

We derive three consequences from the characterizations above.

Consequence 1: Two Gap Theorems. We obtain gap theorems for *balanced* leaf-language definable classes $\mathcal{C}$ and $\mathcal{D}$:

[1] In contrast to chronological order, we first mention the result by Borchert, Kuske, and Stephan [4] and then that by Borchert [3]. The reason is that our paper first proves a result similar to Borchert, Kuske, and Stephan, and then derives a result similar to Borchert.

1. Either $\mathcal{C}$ is contained in NP, or $\mathcal{C}$ contains coUP.
2. Either $\mathcal{D}$ is contained in P, or $\mathcal{D}$ contains UP or coUP.

We translate this into gap theorems that do not involve promise classes.

1. Either $\mathcal{C}$ is contained in NP, or $\mathcal{C}$ contains at least one of the following classes: coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ for p prime.
2. Either $\mathcal{D}$ is contained in P, or $\mathcal{D}$ contains at least one of the following classes: NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ for p prime.

Formerly, such gap theorems were known only for the unbalanced case [3, 4].

Consequence 2: Polylog-Time Reductions Can Decrease Dot-Depth Complexity. Here it is important to note that polylog-time machines neither can do threshold-counting nor can do modulo-counting. (Of course these machines can find out whether the length of the input is even, but they cannot find out whether this holds for the number of a's in the input.) However, there exists a starfree language outside dot-depth 1/2 that is polylog-time reducible to a language in dot-depth 1/2 (Example 1). This shows that polylog-time machines (unable to count by their own) can help permutationfree automata to do threshold-counting. They can decrease dot-depth complexity. It turns out that this decrease can be tremendous: For any $n \geq 1$ there exist starfree languages L_n that are not in $\mathcal{B}_n$ but still in $\mathcal{R}^{plt}(\mathcal{B}_{1/2})$.

We show how polylog-time reductions can exploit restricted modulo-counting possible in $\mathcal{B}_{1/2}$. For this we construct starfree languages with high dot-depth complexity. These languages have the property that words not in the language have a periodical pattern of letters b. We can locally test whether a given word has this pattern. If so, then by looking at the position of the last b we can gain information about the number of a's and b's in the word. This tells us immediately whether the word belongs to the language. Otherwise, if a word does not have the periodical pattern, then it is in the language by definition. All needed computations can be done by polylog-time reductions. In this way, we shift parts of the counting complexity into the polylog-time reduction. This results in a drastic decrease of dot-depth complexity.

Consequence 3: Unbalanced Starfree Leaf-Languages Can Be Much Stronger Than Balanced Ones. Remember that $L_n \notin \mathcal{B}_n$, but still $L_n \in \mathcal{R}^{plt}(\mathcal{B}_{1/2})$. We exploit this to obtain conclusions for leaf-language definable complexity classes. We prove lower bounds for the complexity of $\mathrm{Leaf}_u^p(L_n)$: $\mathrm{Leaf}_b^p(L_n) = \mathrm{NP}$, but $\mathrm{Leaf}_u^p(L_n)$ contains level n of the unambiguous alternation hierarchy. It is expected that level n of the unambiguous alternation hierarchy is not contained in level $n-1$ of the polynomial-time hierarchy. Spakowski and Tripathi [22] construct an oracle such that for every $n \geq 1$, level n of the unambiguous alternation hierarchy is not contained in $\Pi_{\mathrm{n}}^{\mathrm{P}}$. So relative to this oracle, for every $n \geq 1$, $\mathrm{Leaf}_b^p(L_n) = \mathrm{NP}$, yet $\mathrm{Leaf}_u^p(L_n) \not\subseteq \Sigma_{\mathrm{n}-1}^{\mathrm{P}}$. Therefore, our result gives evidence that even for starfree languages, unbalanced leaf-languages are much stronger than balanced ones. This supports the intuition that in general, unbalanced models of computation are stronger than balanced models (e.g., $\mathrm{BPP}_{\mathrm{path}}$ is stronger than BPP unless the polynomial hierarchy collapses [12]).

2 Preliminaries

$\mathbb{N}$ denotes the set of natural numbers. We fix a finite alphabet A such that $|A| \geq 2$ and $a, b \in A$. A^* denotes the set of words (including the empty word ε) over A. Throughout the paper all languages and all classes of the dot-depth hierarchy are considered with respect to A. A language B is *polylog-time reducible* to a language C, $B \leq^{plt} C$, if there exists a polylog-time computable f such that for all x, $x \in B \Leftrightarrow f(x) \in C$. $\mathcal{R}^{plt}(\mathcal{C})$ denotes $\mathcal{C}$'s closure under polylog-time reductions. PLT is the class of languages that have polylog-time computable characteristic functions.

Theorem 1 ([13, 8, 4]). *For $n \geq 1$ and relative to all oracles:*

1. $\mathrm{P} = \mathrm{Leaf}_b^p(\mathrm{PLT}) = \mathrm{Leaf}_b^p(\mathcal{B}_0) = \mathrm{Leaf}_u^p(\mathcal{B}_0)$
2. $\Sigma_{\mathrm{n}}^{\mathrm{P}} = \mathrm{Leaf}_b^p(\mathcal{B}_{n-1/2}) = \mathrm{Leaf}_u^p(\mathcal{B}_{n-1/2})$
3. $\Pi_{\mathrm{n}}^{\mathrm{P}} = \mathrm{Leaf}_b^p(\mathrm{co}\mathcal{B}_{n-1/2}) = \mathrm{Leaf}_u^p(\mathrm{co}\mathcal{B}_{n-1/2})$
4. $\mathrm{BC}(\Sigma_{\mathrm{n}}^{\mathrm{P}}) = \mathrm{Leaf}_b^p(\mathcal{B}_n) = \mathrm{Leaf}_u^p(\mathcal{B}_n)$
5. $\mathrm{NP}(n) = \mathrm{Leaf}_b^p(\mathcal{B}_{1/2}(n)) = \mathrm{Leaf}_u^p(\mathcal{B}_{1/2}(n))$ [2]

Bovet, Crescenzi, and Silvestri [5] and Vereshchagin [24] showed an important connection between polylog-time reducibility and *balanced* leaf languages.

Theorem 2 ([5, 24]). *For all languages B and C,*

$$B \leq^{plt} C \;\Leftrightarrow\; \text{for all oracles } O,\ \mathrm{Leaf}_b^p(B)^O \subseteq \mathrm{Leaf}_b^p(C)^O.$$

3 Regular Languages That Are $\leq^{plt}$-Reducible to $\mathcal{B}_{1/2}$

We characterize the polylog-time closure of $\mathcal{B}_{1/2}$ by:

- a forbidden-pattern characterization, and
- a characterization in terms of regular expressions.

As a consequence, we obtain a gap theorem for balanced leaf-language definable complexity classes $\mathcal{C}$: Either $\mathcal{C}$ is contained in NP, or $\mathcal{C}$ contains coUP. We describe this gap so that no promise classes are involved: Either $\mathcal{C}$ is contained in NP, or $\mathcal{C}$ contains at least one of the following classes: coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ for p prime. Finally, our forbidden-pattern characterization implies decidability of $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$.

Theorem 3. *For every regular L the following are equivalent.*

1. $L \in \mathcal{R}^{plt}(\mathcal{B}_{1/2})$.
2. *The pattern in Figure 1 does not appear in the minimal automaton of L.*
3. *There exists $d \geq 1$ such that L is a finite union of languages of the form $w_0(A^d)^*w_1 \cdots (A^d)^*w_n$ where $n \geq 0$ and $w_i \in A^*$.*

[2] $\mathrm{NP}(n)$ denotes level n of the Boolean hierarchy over NP.

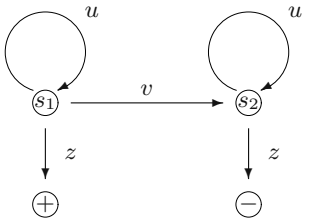


Fig. 1. Forbidden pattern for $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$ where $|v| = |u|$

Barrington et al. [2] give a characterization of $\mathrm{AC}^0 \cap \mathrm{REG}$ for alphabet $\{0,1\}$: The smallest class containing $\{0\}$, $\{1\}$, and all $(\{0,1\}^d)^*$ for $d \geq 1$, that is closed under Boolean operations and concatenation. Interestingly, if we only demand closure under union and concatenation, then, by Theorem 3, this yields $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$. So for alphabet $\{0,1\}$, $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG} \subseteq \mathrm{AC}^0 \cap \mathrm{REG}$. This inclusion is strict, since AC^0 contains all starfree languages over $\{0,1\}$, but $0^* \notin \mathcal{R}^{plt}(\mathcal{B}_{1/2})$. Péladeau, Straubing, and Thérien [18] consider classes of languages that are p-recognized by semigroups of a given variety. For certain varieties $\mathrm{V} * \mathrm{LI}$ they prove a characterization similar to that in Theorem 3. However, we cannot use their characterization, since $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$ cannot be described in terms of varieties $\mathrm{V} * \mathrm{LI}$. Moreover, Maciel, Péladeau, and Thérien [15] characterize the class $\widehat{\mathrm{AC}}^0_1 \cap \mathrm{REG}$ similar to Theorem 3.3. In our notation, $\widehat{\mathrm{AC}}^0_1 \cap \mathrm{REG}$ is the Boolean closure of $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$.

Corollary 1. *Let* $\mathcal{C} = \mathrm{Leaf}^p_b(L)$ *for some regular* L.

1. *If* $L \in \mathcal{R}^{plt}(\mathcal{B}_{1/2})$, *then* $\mathcal{C} \subseteq \mathrm{NP}$.
2. *If* $L \notin \mathcal{R}^{plt}(\mathcal{B}_{1/2})$, *then* $\mathrm{coUP} \subseteq \mathcal{C}$.

Corollary 1 shows a gap for balanced leaf-language definable classes above NP: Any such class higher than NP contains coUP. Since coUP is a promise class, it would be most welcome to show a similar gap that does not involve any promise class. Borchert, Kuske, and Stephan [4] show how to do this. By iterating the coUP pattern they obtain a list of non-promise complexity classes such that every *unbalanced* leaf-language definable class higher than NP contains at least one class from the list.

Corollary 2. *Let* $\mathcal{C} = \mathrm{Leaf}^p_b(L)$ *for some regular* L.

1. *If* $L \in \mathcal{R}^{plt}(\mathcal{B}_{1/2})$, *then* $\mathcal{C} \subseteq \mathrm{NP}$.
2. *If* $L \notin \mathcal{R}^{plt}(\mathcal{B}_{1/2})$, *then* $\mathrm{coNP} \subseteq \mathcal{C}$, *or* $\mathrm{co1NP} \subseteq \mathcal{C}$, *or* $\mathrm{MOD}_p\mathrm{P} \subseteq \mathcal{C}$ *for* p *prime.*

Corollary 3. *It is decidable whether a regular language is* $\leq^{plt}$ *reducible to* $\mathcal{B}_{1/2}$.

Example 1. A starfree $L \notin \mathcal{B}_{1/2}$ that is $\leq^{plt}$ reducible to a language in $\mathcal{B}_{1/2}$.[3]

[3] Some of the following properties of this example were discovered during a discussion with Bernhard Schwarz, Victor Selivanov, and Klaus W. Wagner.

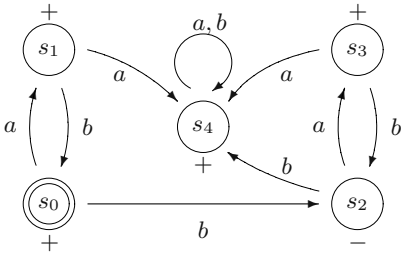


Fig. 2. Automaton $\mathcal{E}$ with initial state s_0

We consider automaton $\mathcal{E}$ (Figure 2). $\mathcal{E}$ is minimal and permutationfree. So $L = L(\mathcal{E})$ is starfree. The automaton contains the forbidden pattern for $\mathcal{B}_{1/2}$ [20]. Therefore, $L \notin \mathcal{B}_{1/2}$. Moreover, $\mathcal{E}$ does not contain the pattern in Figure 1. Therefore, by Theorem 3, $L \in \mathcal{R}^{plt}(\mathcal{B}_{1/2})$ (e.g., L polylog-time reduces to A^*bA^*). L can be characterized in different ways:

$$L = (AA)^* \cup L_0 = (ab)^* \cup L_0 = \overline{(ab)^*b(ab)^*}$$

where $L_0 \stackrel{df}{=} A^*aaA^* \cup A^*bbbA^* \cup A^*bbA^*bbA^* \cup A^*a \cup bbA^* \cup bA^*bbA^*$. It follows that $L \in \mathcal{B}_{1/2} \vee \mathrm{co}\mathcal{B}_{1/2}$ which is the complement of the second level of the Boolean hierarchy over $\mathcal{B}_{1/2}$. In particular, $L \in \mathcal{B}_1$. Moreover, $\mathrm{Leaf}_b^p(L) = \mathrm{NP}$ and $\mathrm{Leaf}_u^p(L) = \mathrm{co1NP}$.

4 Regular Languages That Are Polylog-Time Decidable

This section is similar to Section 3. Here we consider PLT ∩ REG instead of $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$. We provide a characterization of $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{co}\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$ which immediately implies

$$\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{co}\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG} = \mathrm{PLT} \cap \mathrm{REG}.$$

This strong connection between $\mathcal{R}^{plt}(\mathcal{B}_{1/2})$ and PLT allows a translation of results about $\mathcal{R}^{plt}(\mathcal{B}_{1/2})$ (Section 3) to results about PLT. Beside the equation above we obtain two characterizations of regular languages in PLT ∩ REG:

- a forbidden-pattern characterization, and
- a characterization in terms of regular expressions.

While the first characterization is new, the latter one is already known [25]. As a consequence of the forbidden-pattern characterization, we obtain another gap theorem for balanced leaf-language definable complexity classes $\mathcal{C}$: Either $\mathcal{C}$ is contained in P, or $\mathcal{C}$ contains UP or coUP. We describe this gap so that no promise classes are involved: Either $\mathcal{C}$ is contained in P, or $\mathcal{C}$ contains at least one of the following classes: NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ for p prime. Finally, the forbidden-pattern characterization implies decidability of the class PLT ∩ REG.

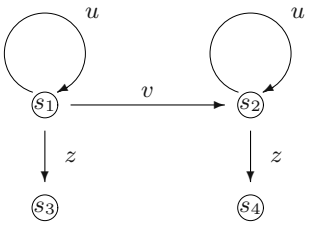


Fig. 3. Forbidden pattern for PLT ∩ REG where $|v| = |u|$ and s_3 accepts $\Leftrightarrow$ s_4 rejects

Theorem 4. *If* $L \in \mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{co}\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$*, then there exists* $d \geq 1$ *such that* L *is a finite union of singletons* $\{u\}$ *and languages* $v(A^d)^*w$ *where* $u, v, w \in A^*$.

This allows the following characterizations of PLT. We point out that the equivalence of statements 1 and 4 in Corollary 4 has been shown by Wagner [25].

Corollary 4. *For every regular* L *the following are equivalent.*

1. $L \in \mathrm{PLT}$.
2. $L \in \mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{co}\mathcal{R}^{plt}(\mathcal{B}_{1/2})$
3. *The pattern in Figure 3 does not appear in the minimal automaton of* L.
4. *There exists* $d \geq 1$ *such that* L *is a finite union of singletons* $\{u\}$ *and languages* $v(A^d)^*w$ *where* $u, v, w \in A^*$.

Corollary 5. *Let* $\mathcal{C} = \mathrm{Leaf}_b^p(L)$ *for some regular* L.

1. *If* $L \in \mathrm{PLT}$*, then* $\mathcal{C} \subseteq \mathrm{P}$.
2. *If* $L \notin \mathrm{PLT}$*, then* $\mathrm{UP} \subseteq \mathcal{C}$ *or* $\mathrm{coUP} \subseteq \mathcal{C}$.

So we have a gap for balanced leaf-language definable classes above P: Any such class higher than P contains UP or coUP. We express this without involving promise classes.

Corollary 6. *Let* $\mathcal{C} = \mathrm{Leaf}_b^p(L)$ *for some regular* L.

1. *If* $L \in \mathrm{PLT}$*, then* $\mathcal{C} \subseteq \mathrm{P}$.
2. *If* $L \notin \mathrm{PLT}$*, then at least one of the following classes is contained in* $\mathcal{C}$*:* NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ *for* p *prime.*

Corollary 7. *It is decidable whether a given regular language belongs to* PLT.

5 Balanced Versus Unbalanced Computations

In Example 1 we have seen that there exist starfree languages L that are not in $\mathcal{B}_{1/2}$, but $\leq^{plt}$ reducible to languages in $\mathcal{B}_{1/2}$. This raises two questions:

1. Does $\mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{SF}$ fall into some level of the dot-depth hierarchy?
2. Can we characterize the complexity of $\mathrm{Leaf}_u^p(L)$ for $L \in \mathcal{R}^{plt}(\mathcal{B}_{1/2}) \cap \mathrm{SF}$?

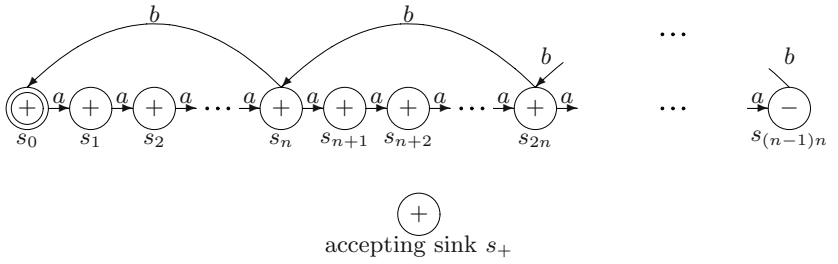


Fig. 4. Automaton $\mathcal{A}_p$ where $p \geq 3$, $n = p - 1$, s_0 is initial state, $s_{(n-1)n}$ is the only rejecting state, and all undefined transitions lead to the accepting sink s_+. All $\mathcal{A}_p$ are minimal. For any prime $p \geq 3$, $L(\mathcal{A}_p) \in \mathrm{SF} \cap \mathcal{R}^{plt}(\mathcal{B}_{1/2})$ but $L(\mathcal{A}_p) \notin \mathcal{B}_{p-3}$

In this section we give a 'no' answer to the first question. For any $n \geq 1$ there exist starfree languages L_n that are not in $\mathcal{B}_n$ but still in $\mathcal{R}^{plt}(\mathcal{B}_{1/2})$. Regarding the second question, we prove lower bounds for the complexity of $\mathrm{Leaf}_u^p(L_n)$. More precisely,

- $\mathrm{Leaf}_b^p(L_n) = \mathrm{NP}$, but
- $\mathrm{Leaf}_u^p(L_n)$ contains level n of the unambiguous alternation hierarchy.

It is expected that level n of the unambiguous alternation hierarchy is not contained in level $n-1$ of the polynomial-time hierarchy. Spakowski and Tripathi [22] construct an oracle such that for every $n \geq 1$, level n of the unambiguous alternation hierarchy is not contained in $\Pi_{\mathrm{n}}^{\mathrm{P}}$. So relative to this oracle, for every $n \geq 1$, $\mathrm{Leaf}_b^p(L_n) = \mathrm{NP}$, yet $\mathrm{Leaf}_u^p(L_n) \not\subseteq \Sigma_{\mathrm{n}-1}^{\mathrm{P}}$. Therefore, our result gives evidence that for starfree languages, unbalanced leaf-languages are much stronger than balanced ones.

We want to give an intuition why it is possible to construct languages of arbitrary dot-depth that are still polylog-time reducible to languages in $\mathcal{B}_{1/2}$. Choose any prime $p \geq 3$. We argue that the language L accepted by automaton $\mathcal{A}_p$ (Figure 4) is not in dot-depth $p-3$, but is polylog-time reducible to A^*aA^*.

Why does L not belong to dot-depth $p-3$? Thomas [23] constructed a family of languages that separate dot-depth classes. ¿From this family we use a language L' that is not in dot-depth $p-3$. It is easy to see that L is the image of L' under the morphism that maps $a \mapsto a^{p-1}$ and $b \mapsto b$. Since dot-depth levels are closed under taking inverse morphisms, we obtain that L is not in dot-depth $p-3$.

Why is L polylog-time reducible to A^*aA^*? Let $n \stackrel{df}{=} p-1$. In $\mathcal{A}_p$, the number of a's between s_{in} and $s_{(i+1)n}$ is $\equiv -1 \pmod p$. All loops that do not go through s_+ are of length $\equiv 0 \pmod p$. So if we reach s_{in}, then the number of letters that has been read so far is $\equiv -i \pmod p$. Call a word *well-formed* if it does not lead from s_0 to s_+.

> In every well-formed word, after $(n-1)n$ consecutive a's there must follow a letter b. $(*)$

Let w be well-formed. Consider any b in w. This b must be read in some state s_{in} where $i \geq 1$. It follows that the number of letters left of this b is $\equiv -i \pmod p$. This shows:

If w is well-formed and $w = w_1 b w_2$, then w_1 leads from s_0 to s_{in} where $i \stackrel{df}{=} (-|w_1| \bmod p)$. (∗∗)

Hence in a well-formed word, the position (modulo p) of some letter b tells us the state in which this letter is read. This shows that we can locally test whether a word is well-formed: Just guess all neighboring b's, make sure that their distance is small (∗), determine the states in which these b's must be read (∗∗), and test whether these states fit to the factor between the b's. This local test shows that the set of words that are not well-formed is polylog-time reducible to A^*aA^*. It remains to argue that the set of words that are in L and that are well-formed is polylog-time reducible to A^*aA^*. This is easy, since by (∗∗), the position of the last b tells us the state in which this b is read. We just have to verify that the remaining part of the word does not lead to $s_{(n-1)n}$.

Theorem 5. *For any prime $p \geq 3$, $L(\mathcal{A}_p) \in \mathrm{SF} \cap \mathcal{R}^{plt}(\mathcal{B}_{1/2})$ but $L(\mathcal{A}_p) \notin \mathcal{B}_{p-3}$.*

Corollary 8. *For every n, there exists a starfree language L_n such that L_n is polylog-time reducible to a language in $\mathcal{B}_{1/2}$, but L_n does not belong to $\mathcal{B}_n$.*

Niedermeier and Rossmanith introduced unambiguous alternating polynomial-time Turing machines and defined the levels $\mathrm{AU}\Sigma_{\mathrm{k}}^{\mathrm{P}}$ and $\mathrm{AU}\Pi_{\mathrm{k}}^{\mathrm{P}}$ of the *unambiguous alternation hierarchy* [17].

Theorem 6. *For every $k \geq 1$ and every $p \geq 4k+2$, $\mathrm{AU}\Sigma_{\mathrm{k}}^{\mathrm{P}} \subseteq \mathrm{Leaf}_u^p(L(\mathcal{A}_p))$.*

Corollary 9. *For every $k \geq 1$ there exists $L \in \mathrm{SF} \cap \mathcal{R}^{plt}(\mathcal{B}_{1/2})$ such that $\mathrm{AU}\Sigma_{\mathrm{k}}^{\mathrm{P}} \subseteq \mathrm{Leaf}_u^p(L)$ but $\mathrm{Leaf}_b^p(L) = \mathrm{NP}$.*

Acknowledgments

The author thanks Bernhard Schwarz, Victor Selivanov, and Klaus W. Wagner for exciting discussions about leaf languages and reducibility notions for starfree languages. In particular, Example 1 was discussed in this group. I would like to thank David A. Mix Barrington, Bernd Borchert, Lane A. Hemaspaandra, Pascal Tesson, and Stephen Travers for drawing my attention to characterizations of $\mathrm{AC}^0 \cap \mathrm{REG}$, for discussions about possible interpretations of Corollary 8, and for other helpful hints.

References

1. M. Arfi. Opérations polynomiales et hiérarchies de concaténation. *Theoretical Computer Science*, 91:71–84, 1991.
2. D. A. Mix Barrington, K. Compton, H. Straubing, and D. Thérien. Regular languages in NC^1. *Journal of Computer and System Sciences*, 44:478–499, 1992.
3. B. Borchert. On the acceptance power of regular languages. *Theoretical Computer Science*, 148:207–225, 1995.

4. B. Borchert, D. Kuske, and F. Stephan. On existentially first-order definable languages and their relation to NP. *Theoretical Informatics and Applications*, 33:259–269, 1999.
5. D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.
6. J. A. Brzozowski. Hierarchies of aperiodic languages. *RAIRO Inform. Theor.*, 10:33–49, 1976.
7. J. A. Brzozowski and R. Knast. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16:37–55, 1978.
8. H.-J. Burtschick and H. Vollmer. Lindström quantifiers and leaf language definability. *International Journal of Foundations of Computer Science*, 9:277–294, 1998.
9. R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5:1–16, 1971.
10. S. Eilenberg. *Automata, languages and machines*, volume B. Academic Press, New York, 1976.
11. C. Glaßer and H. Schmitz. Languages of dot-depth 3/2. In *Proceedings 17th Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 555–566. Springer Verlag, 2000.
12. Y. Han, L. A. Hemaspaandra, and T. Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, 1997.
13. U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.
14. R. Knast. A semigroup characterization of dot-depth one languages. *RAIRO Inform. Theor.*, 17:321–330, 1983.
15. A. Maciel, P. Péladeau, and D. Thérien. Programs over semigroups of dot–depth one. *Theoretical Computer Science*, 245:135–148, 2000.
16. R. McNaughton and S. Papert. *Counterfree Automata*. MIT Press, Cambridge, 1971.
17. R. Niedermeier and P. Rossmanith. Unambiguous computations and locally definable acceptance types. *Theoretical Computer Science*, 194(1-2):137–161, 1998.
18. P. Péladeau, H. Straubing, and D. Thérien. Finite semigroup varieties defined by programs. *Theoretical Computer Science*, 180:325–339, 1997.
19. D. Perrin and J. E. Pin. First-order logic and star-free sets. *Journal of Computer and System Sciences*, 32:393–406, 1986.
20. J. E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory of computing systems*, 30:383–422, 1997.
21. M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.
22. H. Spakowski and R. Tripathi. On the power of unambiguity in alternating machines. Technical Report 851, University of Rochester, 2004.
23. W. Thomas. An application of the Ehrenfeucht–Fraïssé game in formal language theory. *Société Mathématique de France, mémoire 16*, 2:11–21, 1984.
24. N. K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestija Rossijskoj Akademii Nauk*, 57:51–90, 1993. In Russian.
25. K. W. Wagner. A reducibility and complete sets for the dot-depth hierarchy. Manuscript, 2001.

On the Computational Complexity of the Forcing Chromatic Number

Frank Harary[1], Wolfgang Slany[2], and Oleg Verbitsky[3,⋆]

[1] Computer Science Department, New Mexico State University, NM 88003, USA
[2] Institut für Softwaretechnologie, Technische Universität Graz, A-8010 Graz, Austria
[3] Institut für Informatik, Humboldt Universität Berlin, D-10099 Berlin, Germany

Abstract. Let $\chi(G)$ denote the chromatic number of a graph G. A colored set of vertices of G is called forcing if its coloring is extendable to a proper $\chi(G)$-coloring of the whole graph in a unique way. The forcing chromatic number $F_\chi(G)$ is the smallest cardinality of a forcing set of G. We estimate the computational complexity of $F_\chi(G)$ relating it to the complexity class US introduced by Blass and Gurevich. We prove that recognizing if $F_\chi(G) \leq 2$ is US-hard with respect to polynomial-time many-one reductions. Furthermore, this problem is coNP-hard even under the promises that $F_\chi(G) \leq 3$ and G is 3-chromatic. On the other hand, recognizing if $F_\chi(G) \leq k$, for each constant k, is reducible to a problem in US via a disjunctive truth-table reduction. Similar results are obtained also for forcing variants of the clique and the domination numbers of a graph.

1 Introduction

The vertex set of a graph G will be denoted by $V(G)$. An *s-coloring* of G is a map from $V(G)$ to $\{1, 2, \ldots, s\}$. A coloring c is *proper* if $c(u) \neq c(v)$ for any adjacent vertices u and v. A graph G is *s-colorable* if it has a proper s-coloring. The minimum s for which G is s-colorable is called the *chromatic number* of G and denoted by $\chi(G)$. If $\chi(G) = s$, then G is called *s-chromatic*.

A *partial coloring* of G is any map from a subset of $V(G)$ to the set of positive integers. Suppose that G is s-chromatic. Let c be a proper s-coloring and p be a partial coloring of G. We say that p *forces* c if c is a unique extension of p to a proper s-coloring. The domain of p will be called a *defining set for* c. We call $D \subseteq V(G)$ a *forcing set in* G if this set is defining for some proper s-coloring of G. The minimum cardinality of a forcing set is called the *forcing chromatic number* of G and denoted by $F_\chi(G)$.

We study the computational complexity of this graph invariant. To establish the hardness of computing $F_\chi(G)$, we focus on the respective slice decision problems $\textsc{Force}_\chi(k)$: Given a graph G, decide if $F_\chi(G) \leq k$. Here k is a fixed nonnegative constant.

⋆ Supported by an Alexander von Humboldt fellowship.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 182–193, 2005.

The cases of $k = 0$ and $k = 1$ are tractable. It is clear that $F_\chi(G) = 0$ iff $\chi(G) = 1$, that is, G is empty. Furthermore, $F_\chi(G) = 1$ iff $\chi(G) = 2$ and G is connected, that is, G is a connected bipartite graph. Thus, we can pay attention only to $k \geq 2$. Since there is a simple reduction of $\textsc{Force}_\chi(k)$ to $\textsc{Force}_\chi(k+1)$ (see Lemma 5), it would suffice to show that even $\textsc{Force}_\chi(2)$ is computationally hard. This is indeed the case.

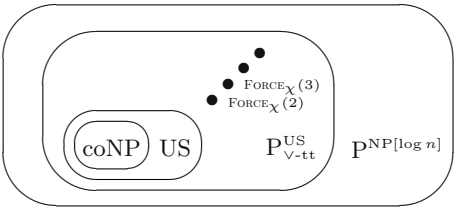
Let 3COL denote the set of 3-colorable graphs and U3COL the set of those graphs in 3COL having a unique, up to renaming colors, proper 3-coloring. First of all, note that a hardness result for $\textsc{Force}_\chi(2)$ is easily derivable from two simple observations:

$$\begin{array}{l} \text{If } F_\chi(G) \leq 2, \text{ then } G \in \text{3COL}; \\ \text{If } G \in \text{U3COL}, \text{ then } F_\chi(G) \leq 2. \end{array} \tag{1}$$

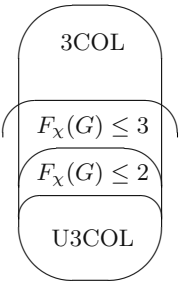
The set 3COL was shown to be NP-complete at the early stage of the NP-completeness theory by reduction from SAT, the set of satisfiable Boolean formulas. It will be benefittable to use a well-known stronger fact: There is a polynomial-time many-one reduction p from SAT to 3COL which is *parsimonious*, that is, any Boolean formula Φ has exactly as many satisfying assignments to variables as the graph $p(\Phi)$ has proper 3-colorings (colorings obtainable from one another by renaming colors are not distinguished). In particular, if Φ has a unique satisfying assignment, then $p(\Phi) \in$ U3COL and hence $F_\chi(p(\Phi)) \leq 2$, while if Φ is unsatisfiable, then $p(\Phi) \notin$ 3COL and hence $F_\chi(p(\Phi)) > 2$.

Valiant and Vazirani [16] designed a polynomial-time computable randomized transformation r of the set of Boolean formulas such that, if Φ is a satisfiable formula, then with a non-negligible probability the formula $r(\Phi)$ has a unique satisfying assignment, while if Φ is unsatisfiable, then $r(\Phi)$ is surely unsatisfiable. Combining r with the parsimonious reduction p of SAT to 3COL, we arrive at the conclusion that $\textsc{Force}_\chi(2)$ is NP-hard with respect to *randomized* polynomial-time many-one reductions. As a consequence, the forcing chromatic number is not computable in polynomial time unless any problem in NP is solvable by a polynomial-time Monte Carlo algorithm with one-sided error.

We aim at determining the computational complexity of $F_\chi(G)$ more precisely. Our first result establishes the hardness of $\textsc{Force}_\chi(2)$ with respect to *deterministic* polynomial-time many-one reductions. This reducibility concept will be the default in what follows. The complexity class US, introduced by Blass and Gurevich [1], consists of languages L for which there is a polynomial-time nondeterministic Turing machine N such that a word x belongs to L iff N on input x has exactly one accepting computation path. Denote the set of Boolean formulas with exactly one satisfying assignment by USAT. This set is complete for US. As easily seen, U3COL belongs to US and, by the parsimonious reduction from SAT to 3COL, U3COL is another US-complete set. By the Valiant-Vazirani reduction, the US-hardness under polynomial-time many-one reductions implies the NP-hardness under randomized reductions and hence the former hardness concept should be considered stronger. It is also known that US includes coNP [1] and this inclusion is proper unless the polynomial time



Location of the slice decision problems for $F_\chi(G)$ in the hierarchy of complexity classes.



The class of graphs with $F_\chi(G)$ at most 2 and surroundings.

Fig. 1. Depicting the main results

hierarchy collapses [14]. Moreover, the US-hardness implies the coNP-hardness [1]. We prove that the problem FORCE$_\chi(2)$ is US-hard.

Note that this result is equivalent to the reducibility of U3COL to FORCE$_\chi(2)$. Such a reduction would follow from the naive hypothesis, which may be suggested by (1), that a 3-chromatic G is in U3COL iff $F_\chi(G) = 2$. It should be stressed that the latter is far from being true in view of Lemma 4.2 below.

On the other hand, we are able to estimate the complexity of every FORCE$_\chi(k)$ from above by putting this family of problems in a complexity class which is a natural extension of US. We show that, for each $k \geq 2$, the problem FORCE$_\chi(k)$ is reducible to a set in US via a polynomial-time disjunctive truth-table reduction (see Sect. 2 for definitions). This improves on the straightforward inclusion of FORCE$_\chi(k)$ in Σ_2^P.

Denote the class of decision problems reducible to US under such reductions by $P^{US}_{V\text{-}tt}$. As shown by Chang, Kadin, and Rohatgi [2], $P^{US}_{V\text{-}tt}$ is strictly larger than US unless the polynomial time hierarchy collapses to its third level. The position of the problems under consideration in the hierarchy of complexity classes is shown in Fig. 1, where $P^{NP[\log n]}$ denotes the class of decision problems solvable by polynomial-time Turing machines with logarithmic number of queries to an NP oracle. The latter class coincides with the class of problems polynomial-time truth-table reducible to NP [10].

Our next result gives a finer information about the hardness of FORCE$_\chi(2)$. Note that, if $\chi(G) = 2$, then $F_\chi(G)$ is equal to the number of connected components of G. It turns out that the knowledge that $\chi(G) = 3$ does not help in computing $F_\chi(G)$. Moreover, it is hard to recognize whether or not $F_\chi(G) = 2$ even if it is also known that $F_\chi(G) \leq 3$. Stating these strengthenings, we relax our hardness concept from the US-hardness to the coNP-hardness. Thus, we prove that the problem FORCE$_\chi(2)$ is coNP-hard even under the promises that $G \in$ 3COL and $F_\chi(G) \leq 3$ (see Fig. 1). Note that the Valiant-Vazirani reduction implies no kind of a hardness result for the promise version of FORCE$_\chi(2)$.

In fact, many other graph characteristics also have natural forcing variants. Recall that a *clique* in a graph is a set of pairwise adjacent vertices. The maxi-

mum number of vertices of G in a clique is denoted by $\omega(G)$ and called the *clique number* of G. A clique is *optimal* if it consists of $\omega(G)$ vertices. A set of vertices is called *forcing* if it is included in a unique optimal clique. We denote the minimum cardinality of a forcing set by $F_\omega(G)$ and call it the *forcing clique number* of G.

Furthermore, we say that a vertex of a graph G *dominates* itself and any adjacent vertex. A set $D \subseteq V(G)$ is called *dominating* if every vertex of G is dominated by a vertex in D. The *domination number* of G, denoted by $\gamma(G)$, is the minimum cardinality of a dominating set of G. Similarly to the above, a *forcing set* of vertices is one included in a unique optimal dominating set. The minimum cardinality of a forcing set is denoted by $F_\gamma(G)$ and called the *forcing domination number* of G. This graph invariant is introduced and studied by Chartrand, Gavlas, Vandell, and Harary [3].

For the forcing clique and domination numbers we consider the respective slice decision problems $\textsc{Force}_\omega(k)$ and $\textsc{Force}_\gamma(k)$ and show the same relation of them to the class US that we have for the forcing chromatic number. Actually, the disjunctive truth-table reducibility to US is proved for all the three numbers by a uniform argument. However, the US-hardness with respect to many-one reductions for ω and γ is proved differently than for χ. The case of ω and γ seems combinatorially simpler because of the following equivalence: A graph G has a unique optimal clique iff $F_\omega(G) = 0$ and similarly with γ. The study of *unique optimum (UO) problems* was initiated by Papadimitriou [13]. Due to the US-hardness of the UO CLIQUE and UO DOMINATING SET problems, we are able to show the US-hardness of $\textsc{Force}_\omega(k)$ and $\textsc{Force}_\gamma(k)$ using only well-known standard reductions, whereas for $\textsc{Force}_\chi(k)$ we use somewhat more elaborate reductions involving graph products.

Overview of Previous Related Work

Forcing Chromatic Number of Particular Graphs. For a few particular families of graphs, the forcing chromatic number is computed in [12, 11]. Our results show that no general approach for efficient computing the forcing chromatic number is possible unless NP = P (and even US = P).

Latin Squares and Complexity of Recognizing a Forcing Coloring. A *Latin square of order* n is an $n \times n$ matrix with entries in $\{1, 2, \ldots, n\}$ such that every row and column contains all the n numbers. In a *partial Latin square* some entries may be empty and every number occurs in any row or column at most once. A partial Latin square is called a *critical set* if it can be completed to a Latin square in a unique way. Colbourn, Colbourn, and Stinson [4] proved that recognition if a given partial Latin square L is a critical set is coNP-hard. As it is observed in [12], there is a natural one-to-one correspondence between Latin squares of order n and proper n-colorings of the Cartesian square $K_n \times K_n$ which matches critical sets and forcing colorings. It follows that it is coNP-hard to recognize if a given partial coloring p of a graph is forcing, even if the problem is restricted to graphs $K_n \times K_n$.

Variety of Combinatorial Forcing Numbers. Critical sets are studied since the seventies. The forcing chromatic number (as well as the forcing domination number) attracted attention of researchers in the mid-nineties. In fact, a number of other problems in diverse areas of combinatorics have a similar forcing nature. Defining sets in block designs (Gray [6]) and forcing matching number (Harary, Klein, and Živković [8]) now have a rather rich bibliography. Other graph invariants whose forcing versions have appeared in the literature are the geodetic, the hull, and the unilateral orientation numbers, and this list is likely inexhaustive.

2 Background

2.1 Basics of Complexity Theory

We write $X \leq^{\mathrm{P}}_{\mathrm{m}} Y$ to say that there is a polynomial-time many-one reduction from a language X to a language Y. A *disjunctive truth-table reduction* of X to Y is a transformation which takes any word x to a set of words $y_1, \ldots, y_m$ so that $x \in X$ iff $y_i \in Y$ for at least one $i \leq m$. We write $X \leq^{\mathrm{P}}_{\vee\text{-tt}} Y$ to say that there is such a polynomial-time reduction. If C is a class of languages and $\leq$ is a reducibility, then $\mathrm{C} \leq X$ means that $Y \leq X$ for all Y in C (i.e. X is C-hard under $\leq$) and $X \leq \mathrm{C}$ means that $X \leq Y$ for some Y in C.

Let Y and Q be languages. Whenever referring to a *decision problem* Y *under the promise* Q, we mean that membership in Y is to be decided only for inputs in Q. A reduction r of an ordinary decision problem X to a problem Y under the promise Q is a normal many-one reduction from X to Y with the additional requirement that $r(x) \in Q$ for all x. This definition allows us to extend the notion of C-hardness to promise problems.

A polynomial-time computable function h is called an *AND*$_2$ *function for a language* Z if for any pair x, y we have both x and y in Z iff $h(x, y)$ is in Z.

2.2 Graph Products

Let $E(G)$ denote the set of edges of a graph G. Given two graphs G_1 and G_2, we define a product graph on the vertex set $V(G_1) \times V(G_2)$ in two ways. Vertices (u_1, u_2) and (v_1, v_2) are adjacent in the *Cartesian product* $G_1 \times G_2$ if either $u_1 = v_1$ and $\{u_2, v_2\} \in E(G_2)$ or $u_2 = v_2$ and $\{u_1, v_1\} \in E(G_1)$. They are adjacent in the *categorical product* $G_1 \cdot G_2$ if both $\{u_1, v_1\} \in E(G_1)$ and $\{u_2, v_2\} \in E(G_2)$.

A set $V(G_1) \times \{v\}$ for each $v \in V(G_2)$ will be called a G_1*-layer of* v and a set $\{u\} \times V(G_2)$ for each $u \in V(G_1)$ will be called a G_2*-layer of* u.

Lemma 1 (Sabidussi). $\chi(G \times H) = \max\{\chi(G), \chi(H)\}$.

If c is a proper coloring of G, it is easy to see that $c^*(x, y) = c(x)$ is a proper coloring of $G \cdot H$. We will say that c *induces* c^*. Similarly, any proper coloring of H induces a proper coloring of $G \cdot H$. This implies the following well-known fact.

Lemma 2. $\chi(G \cdot H) \leq \min\{\chi(G), \chi(H)\}$.

We call two s-colorings *equivalent* if they are obtainable from one another by permutation of colors. Proper s-colorings of a graph G are equivalent if they

determine the same partition of $V(G)$ into s independent sets. Let $N_\chi(G)$ denote the number of such partitions for $s = \chi(G)$. A graph G is *uniquely colorable* if $N_\chi(G) = 1$. In particular, $G \in$ U3COL iff $\chi(G) = 3$ and $N_\chi(G) = 1$.

Lemma 3 (Greenwell-Lovász [7]). *Let G be a connected graph with $\chi(G) > n$. Then $G \cdot K_n$ is uniquely n-colorable.*

Lemma 4. *1. $F_\chi(G) \geq \chi(G) - 1$ and we have equality whenever $N_\chi(G) = 1$.*
2. For any k, there is a 3-chromatic graph G_k on $4k+2$ vertices with $F_\chi(G_k) = 2$ and $N_\chi(G_k) = 2^{k-1} + 1$.

Item 2 shows that the converse of the second part of Item 1 is false.

Proof. Item 1 is obvious. To prove Item 2, consider $H = K_3 \times K_2$. This graph has two inequivalent colorings c_1 and c_2 shown in Fig. 2. Let $u, v, w \in V(H)$ be as in Fig. 2. Note that a partial coloring $p_1(u) \neq p_1(v)$ forces c_1 or its equivalent and that $p_2(u) = p_2(v) \neq p_2(w)$ forces c_2.

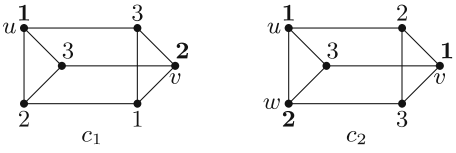


Fig. 2. Proper 3-colorings of $K_3 \times K_2$

Let G_k consist of k copies of H with all u and all v identified, that is, G_k has $4k + 2$ vertices. Since the set $\{u, v\}$ stays forcing in G_k, we have $F_\chi(G_k) = 2$. If u and v are assigned the same color, we are free to assign each copy of w any of the two remaining colors. It follows that $N_\chi(G_k) = 2^{k-1} + 1$.

3 Complexity of $F_\chi(G)$: Lower Bounds

Theorem 1. *For each $k \geq 2$, the problem* Force$_\chi(k)$ *is US-hard. Moreover, this holds true even if we consider only connected graphs.*

We first observe that the family of problems Force$_\chi(k)$ is linearly ordered with respect to the $\leq^{\mathrm{P}}_{\mathrm{m}}$-reducibility. A simple reduction showing this does not preserve connectedness of graphs. However, if we restrict ourselves to connected graphs, we are able to show that Force$_\chi(2)$ remains the minimum element in the $\leq^{\mathrm{P}}_{\mathrm{m}}$ order. We then prove that Force$_\chi(2)$ is US-hard (even for connected graphs).

Lemma 5. Force$_\chi(k) \leq^P_m$ Force$_\chi(k+1)$.

Proof. Given a non-empty graph G, we add one isolated vertex to it. Denoting the result by $G + K_1$, it is enough to notice that $F_\chi(G + K_1) = F_\chi(G) + 1$.

Lemma 6. *Let $k \geq 2$. Then* $\text{FORCE}_\chi(2)$ *reduces to* $\text{FORCE}_\chi(k)$ *even if we consider the problems only for connected graphs.*

The proof is omitted due to space limitation and can be found in the complete version of the paper [9].

Lemma 7. $\text{FORCE}_\chi(2)$ *is US-hard even if restricted to connected graphs.*

To prove the lemma, we describe a reduction from U3COL. Note that U3COL remains US-complete when restricted to connected graphs and that our reduction will preserve connectedness. Since the class of 2-colorable graphs is tractable and can be excluded from consideration, the desired reduction is given by the following lemma.

Lemma 8. *Suppose that $\chi(G) \geq 3$. Then $G \in$ U3COL iff $F_\chi(G \times K_3) = 2$.*

Proof. Case 1: $G \in$ U3COL. We have to show that $F_\chi(G \times K_3) = 2$.

Fix arbitrary $u, v \in V(G)$ whose colors in the proper 3-coloring of G are different, for example, u and v can be any adjacent vertices of G. Let $V(K_3) = \{1, 2, 3\}$. Assign $p(u, 1) = 1$ and $p(v, 2) = 2$ and check that p forces a proper 3-coloring of $G \times K_3$. Assume that c is a proper 3-coloring of $G \times K_3$ consistent with p. Since c on each G-layer coincides with the 3-coloring of G up to permutation of colors, we easily infer that $c(v, 1) = c(u, 2) = 3$. This implies $c(u, 3) = 2$ and $c(v, 3) = 1$. Thus, in each G-layer we have two vertices with distinct colors, which determines colors of all the other vertices. As easily seen, the coloring obtained is really proper.

Case 2: $G \in$ 3COL $\setminus$ U3COL. We have to check that $F_\chi(G \times K_3) \geq 3$.

Given a partial coloring p of two vertices $a, b \in V(G \times K_3)$, we have to show that it is not forcing. The cases that $p(a) = p(b)$ or that a and b are in the same G- or K_3-layer are easy. Without loss of generality we therefore suppose that $p(a) = 1$, $p(b) = 2$, $a = (u, 1)$, and $b = (v, 2)$, where u and v are distinct vertices of G. Define two partial colorings of G by $c_1(u) = c_1(v) = 1$ and by $c_2(u) = 1$, $c_2(v) = 3$.

Subcase 2.1: Both c_1 and c_2 extend to proper 3-colorings of G. Denote the extensions by e_1 and e_2 respectively. Denote the three G-layers of $G \times K_3$ by G_1, G_2, G_3 and consider e_1, e_2 on G_1. For each $i = 1, 2$, e_i and p agree and have a common extension to a proper coloring of $G \times K_3$. Thus, p is not forcing.

Subcase 2.2: Only c_1 extends to a proper 3-coloring of G. Since G is not uniquely colorable, there must be at least two extensions, e_1 and e_2, of c_1 to proper 3-colorings of G_1. As in the preceding case, e_1 and e_2 each agree with p and together with p extend two distinct colorings of $G \times K_3$.

Subcase 2.3: Only c_2 extends to a proper coloring of G. This case is completely similar to Subcase 2.2.

Case 3: $G \notin$ 3COL. We have $\chi(G \times K_3) \geq 4$ by Lemma 1 and $F_\chi(G \times K_3) \geq 3$ by Lemma 4.1.

Theorem 1 immediately follows from Lemmas 7 and 6.

Theorem 2. *The problem* $\textsc{Force}_\chi(2)$ *is coNP-hard even under the promises that* $F_\chi(G) \leq 3$ *and* $\chi(G) \leq 3$ *and even if an input graph* G *is given together with its proper 3-coloring.*

Let us for a while omit the promise that $F_\chi(G) \leq 3$. Then Theorem 2 is provable by combining the Greenwell-Lovász reduction of coNP to US given by Lemma 3 and our reduction of US to $\textsc{Force}_\chi(2)$ given by Lemma 8. Doing so, we easily deduce the following. If $\chi(G) > 3$, then $G \cdot K_3$ is uniquely 3-colorable and hence $F_\chi((G \cdot K_3) \times K_3) = 2$. If $\chi(G) = 3$, then $G \cdot K_3$ is 3-chromatic, has two induced 3-colorings, and hence $F_\chi((G \cdot K_3) \times K_3) \geq 3$. To obtain Theorem 2 (the weakened version without the promise $F_\chi(G) \leq 3$), it now suffices to make the following simple observation.

Lemma 9. $\chi((G \cdot K_3) \times K_3) = 3$ *for any graph* G*. Moreover, a proper 3-coloring is efficiently obtainable from the explicit product representation of* $(G \cdot K_3) \times K_3$*.*

To obtain the full version of Theorem 2, we only slightly modify the reduction: Before transforming G in $(G \cdot K_3) \times K_3$, we add to G a triangle with one vertex in $V(G)$ and two other new vertices. Provided $\chi(G) \geq 3$, this does not change $\chi(G)$ and hence the modified transformation is an equally good reduction. The strengthening (the promise $F_\chi(G) \leq 3$) is given by the following lemma, concluding the proof of Theorem 2.

Lemma 10. *If a graph* G *is connected and contains a triangle, then* $F_\chi((G \cdot K_3) \times K_3) \leq 3$*.*

The proofs of Lemmas 9 and 10 can be found in [9].

4 General Setting

In fact, many other graph characteristics also have natural forcing variants. Taking those into consideration, it will be convenient to use the formal concept of an NP optimization problem (see e.g. [5]).

Let $\{0,1\}^*$ denote the set of binary strings. The length of a string $w \in \{0,1\}^*$ is denoted by $|w|$. We will use notation $[n] = \{1, 2, \ldots, n\}$.

An *NP optimization problem* $\pi = (\mathrm{opt}_\pi, I_\pi, sol_\pi, v_\pi)$ (where subscript π may be omitted) consists of the following components.

- $\mathrm{opt} \in \{\max, \min\}$ is a *type* of the problem.
- $I \subseteq \{0,1\}^*$ is the polynomial-time decidable set of *instances* of π.
- Given $x \in I$, we have $sol(x) \subset \{0,1\}^*$, the set of *feasible solutions* of π on instance x. We suppose that all $y \in sol(x)$ have the same length that

depends only on $|x|$ and is bounded by $|x|^{O(1)}$. Given x and y, it is decidable in polynomial time whether $y \in sol(x)$.
- $v : \{0,1\}^* \times \{0,1\}^* \to \mathbf{N}$ is a polynomial-time computable *objective function* taking positive integer values. If $y \in sol(x)$, then $v(x,y)$ is called the *value* of y.

The problem is, given an instance x, to compute the optimum value

$$\pi(x) = \mathrm{opt}_{y \in sol(x)}\, v(x,y).$$

Such a problem is called *polynomially bounded* if $v(x,y) = |x|^{O(1)}$ for all $x \in I$ and $y \in sol(x)$.

Any $y \in sol(x)$ whose value is optimum is called an *optimum solution* of π on instance x. Let $optsol(x)$ denote the set of all such y. Given an NP optimization problem π, we define

$$\mathrm{UO}_\pi = \{\, x : \, |\, optsol(x)| = 1\}\,.$$

Example 1. The problem of computing the chromating number of a graph is expressible as a quadruple $\chi = (\min, I, sol, v)$ as follows. A graph G with vertex set $V(G) = \{v_1, \ldots, v_n\}$ is represented by its adjacency matrix written down row after row as a binary string x of length n^2. A feasible solution, that is a proper coloring $c : V(G) \to [n]$, is represented by a binary string $y = c(v_1) \ldots c(v_n)$ of length n^2, where a color i is encoded by string $0^{i-1}10^{n-i}$. The value $v(x,y)$ is equal to the actual number of colors occurring in y.

For the problem of computing the clique number it is natural to fix the following representation. An instance graph G is encoded as above. A feasible solution, which is a subset of $V(G)$, is encoded by its characteristic binary string of length n. The problem of computing the domination number is represented in the same way.

Given a non-empty set $U \subseteq \{0,1\}^l$, we define $force(U)$ to be the minimum cardinality of a set $S \subseteq [l]$ such that there is exactly one string in U with 1 at every position from S. Additionally, let $force(\emptyset) = \infty$. With each NP optimization problem π we associate its *forcing number* F_π, an integer-valued function of instances of π defined by

$$F_\pi(x) = force(optsol(x)).$$

Let $\textsc{Force}_\pi(k) = \{\, x : \, F_\pi(x) \le k\}$. It is easy to check that, if χ, ω, and γ are represented as in Example 1, then F_χ, F_ω, and F_γ are precisely those graph invariants introduced in Sect. 1.

Note that $force(U) = 0$ iff U is a singleton. It follows that for $\pi \in \{\omega, \gamma\}$ we have

$$x \in \mathrm{UO}_\pi \;\text{ iff }\; F_\pi(x) = 0. \tag{2}$$

This will be the starting point of our analysis of the decision problems $\textsc{Force}_\omega(k)$ and $\textsc{Force}_\gamma(k)$.

5 Hardness of $\text{FORCE}_\omega(k)$ and $\text{FORCE}_\gamma(k)$

Theorem 3. *Let* $\pi \in \{\omega, \gamma\}$. *Then* $\text{US} \leq_m^P \text{UO}_\pi = \text{FORCE}_\pi(0) \leq_m^P \text{FORCE}_\pi(k) \leq_m^P \text{FORCE}_\pi(k+1)$ *for any* $k \geq 0$.

As observed in [15], the decision problem UO_ω is US-hard. The composition of a few standard reductions shows that $\text{UO}_\omega \leq_{\text{m}}^{\text{P}} \text{UO}_\gamma$ (for details see [9]). Thus, UO_ω and UO_γ are both US-hard and Theorem 3 follows from the following fact.

Lemma 11. *Let* $\pi \in \{\omega, \gamma\}$. *Then* $\text{FORCE}_\pi(k) \leq_m^P \text{FORCE}_\pi(k+1)$ *for any* $k \geq 0$.

Proof. Given a graph G, we have to construct a graph H such that $F_\pi(G) \leq k$ iff $F_\pi(H) \leq k+1$. It suffices to ensure that

$$F_\pi(H) = F_\pi(G) + 1. \tag{3}$$

Let $\pi = \omega$. Let H be the result of adding to G two new vertices u and v and the edges $\{w, u\}$ and $\{w, v\}$ for all $w \in V(G)$. Any optimal clique in H consists of an optimal clique in G and of either u or v. Hence any forcing set in H consists of a forcing set in G and of either u or v (we use the terminology of Sect. 1). This implies (3).

If $\pi = \gamma$, we obtain H from G by adding a new isolated edge.

6 Complexity of $\text{FORCE}_\pi(k)$: An Upper Bound

Theorem 4. *Let* π *be a polynomially bounded NP optimization problem (in particular,* π *can be any of* χ, ω, *and* γ). *Then* $\text{FORCE}_\pi(k) \leq_{\vee\text{-}tt}^P \text{US}$ *for each* $k \geq 0$.

Proof. We will assume that π is a minimization problem (the case of maximization problems is quite similar). Suppose that $v(x, y) \leq |x|^c$ for a constant c. Given $1 \leq m \leq |x|^c$, we define $sol^m(x) = \{ y \in sol(x) : v(x, y) = m \}$ and $F_\pi^m(x) = force(sol^m(x))$. In particular, $F_\pi^m(x) = F_\pi(x)$ if $m = \pi(x)$.

Let k be a fixed integer. Notice that

$$F_\pi(x) \leq k \text{ iff } \bigvee_{m=1}^{|x|^c} (F_\pi^m(x) \leq k \wedge \pi(x) \geq m) \tag{4}$$

(actually, only a disjunction member where $m = \pi(x)$ can be true). The set of pairs (x, m) with $\pi(x) \geq m$ is in coNP and hence in US. Let us now show that the set of (x, m) with $F_\pi^m(x) \leq k$ is disjunctively reducible to US.

Recall that $sol(x) \subseteq \{0,1\}^{l(x)}$, where $l(x) \leq |x|^d$ for a constant d. Define T to be the set of quadruples (x, m, l, D) such that m and l are positive integers, $D \subseteq [l]$, and there is a unique $y \in sol^m(x)$ of length l with all 1's in positions from D. It is easy to see that T is in US and

$$F_\pi^m(x) \leq k \text{ iff } \bigvee_{\substack{l, D: l \leq |x|^d \\ D \subseteq [l],\, |D| \leq k}} (x, m, l, D) \in T.$$

Combining this equivalence with (4), we conclude that $F_\pi(x) \leq k$ iff there are numbers $m \leq |x|^c$ and $l \leq |x|^d$ and a set $D \subseteq [l]$ of size at most k such that

$$(x, m, l, D) \in T \ \wedge \ \pi(x) \geq m.$$

Since every US-complete set has an AND_2 function, this conjunction is expressible as a proposition about membership of the quadruple (x, m, l, D) in a US-complete set. Thus, the condition $F_\pi(x) \leq k$ is equivalent to a disjunction of less than $|x|^{c+d(k+1)}$ propositions each verifiable in US.

7 Open Questions

1. We have considered forcing versions of three popular graph invariants: the chromatic, the clique, and the domination numbers (F_χ, F_ω, and F_γ respectively). We have shown that slice decision problems for each of F_χ, F_ω, and F_γ are as hard as US under the many-one reducibility and as easy as US under the disjunctive truth-table reducibility. The latter upper bound is actually true for the forcing variant of any polynomially bounded NP optimization problem. The lower bound in the case of F_ω and F_γ is provable by using standard reductions on the account of a close connection with the unique optimum problems UO_ω and UO_γ. However, in the case of F_χ we use somewhat more elaborate reductions involving graph products. We point out two simple reasons for the distinction between F_χ and F_ω, F_γ. First, unlike the case of ω and γ, the unique colorability of a graph is apparently inexpressible in terms of F_χ (cf. Lemma 4.2). Second, we currently do not know any relation between F_χ, F_ω, and F_γ as optimization problems (cf. further discussion).

2. As is well known, most NP-complete decision problems, including those for χ, ω, and γ, are rather similar to each other: There are parsimonious many-one reductions between them. However, it is also well known that the total similarity disappears as soon as we get interested in approximation properties of the underlying NP optimization problems or in parametrized complexity variants of the decision problems themselves. In particular, χ, ω, and γ occupy different positions in the two mentioned hierarchies. It would be interesting to compare complexities of F_χ, F_ω, and F_γ introducing appropriate reducibility concepts.

3. Our results imply that the problems $\textsc{Force}_\pi(k)$ for any $\pi \in \{\chi, \omega, \gamma\}$ and $k \geq 0$ (except $k = 0, 1$ if $\pi = \chi$) have the same polynomial-time Turing degree. Let $\textsc{Force}_\pi(*) = \{\, (x, k) : F_\pi(x) \leq k\}$. How related are $\textsc{Force}_\pi(*)$ for $\pi \in \{\chi, \omega, \gamma\}$ under polynomial-time reductions? Note that these three decision problems are US-hard and belong to Σ_2^{P}.

4. Is $\textsc{Force}_\pi(*)$ NP-hard under $\leq_{\mathrm{m}}^{\mathrm{P}}$-reductions for any π under consideration?

5. Given a graph with a perfect matching, Harary, Klein, and Živković [8] define its forcing matching number as the minimum size of a forcing set of edges, where the latter is a set contained in a unique perfect matching. Is this graph invariant computationally hard? Note the well-known fact that, despite computing the number of perfect matchings is #P-complete, uniqueness of a perfect matching is efficiently decidable. Note also that forcing sets of edges are efficiently recognizable

and, as a consequence, for each fixed k the problem of deciding if the forcing matching number does not exceed k is polynomial-time solvable.

Acknowledgement. We thank an anonymous referee for useful comments. This work was initiated when the third author visited the Department of Information Systems at the Vienna University of Technology. He sincerely thanks Georg Gottlob for his hospitality during this visit.

References

1. Blass, A., Gurevich, Y.: On the unique satisfiability problem. Information and Control **55** (1982) 80–88
2. Chang, R., Kadin, J., Rohatgi, P.: On unique satisfiability and the threshold behaviour of randomized reductions. J. Comp. Syst. Sci. **50** (1995) 359–373
3. Chartrand, G., Gavlas, H., Vandell, R.C., Harary, F.: The forcing domination number of a graph. J. Comb. Math. Comb. Comput. **25** (1997) 161–174
4. Colbourn, C.J., Colbourn, M.J., Stinson, D.R.: The computational complexity of recognizing critical sets. In: Graph theory. Proc. of the 1-st Southeast Asian Colloq., Singapore 1983. Lecture Notes in Mathematics, Vol. 1073. Springer-Verlag, Berlin Heidelberg New York (1984) 248–253
5. Crescenzi, P.: A short quide to approximation preserving reductions. In: Proc. of the 12-th Ann. Conference on Computational Complexity (1997) 262–272
6. Gray, K.: On the minimum number of blocks defining a design. Bull. Aust. Math. Soc. **41** (1990) 97–112
7. Greenwell D.L., Lovász, L.: Applications of product colouring. Acta Math. Acad. Sci. Hung. **25** (1974) 335–340
8. Harary, F., Klein, D., Živković, T.: Graphical properties of polyhexes: Perfect matching vector and forcing. J. Math. Chemistry **6** (1991) 295-306
9. Harary, F., Slany, W., Verbitsky, O.: On the computational complexity of the forcing chromatic number. E-print http://arXiv.org/abs/cs.CC/0406044 (2004)
10. Hemachandra, L.: Structure of complexity classes: separations, collapses, and completeness. In: Chytil, M.P, Janiga, L., Koubek, V. (eds.): Proc. of the 13-th Symp. on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science, Vol. 324. Springer-Verlag, Berlin Heidelberg New York (1988) 59–73
11. Mahdian, M., Mahmoodian, E.S., Naserasr, R., Harary, F.: On defining sets of vertex colorings of the Cartesian product of a cycle with a complete graph. In: Alavi, Y., Lick, D.R., Schwenk, A. (eds.): Combinatorics, Graph Theory, and Algorithms. New Issues Press, Kalamazoo (1999) 461–467
12. Mahmoodian, E.S., Naserasr, R., Zaker, M.: Defining sets in vertex colorings of graphs and Latin rectangles. Discrete Math. **167–168** (1997) 451–460
13. Papadimitriou, C.: On the complexity of unique solutions. J. of the ACM **31** (1984) 392–400
14. Ranjan, D., Chari, S., Rohatgi, P.: Improving known solutions is hard. Comput. Complexity **3** (1993) 168–185
15. Thierauf, T.: The computational complexity of equivalence and isomorphism problems. Lecture Notes in Computer Science, Vol. 1852. Springer-Verlag, Berlin Heidelberg New York (2000)
16. Valiant, L.G., Vazirani, V.V.: NP is as easy as detecting unique solution. Theor. Comp. Sci. **47** (1986) 287–300

More Efficient Queries in PCPs for NP and Improved Approximation Hardness of Maximum CSP

Lars Engebretsen and Jonas Holmerin

Department of Numerical Analysis and Computer Science,
Royal Institute of Technology, SE-100 44 Stockholm, Sweden
{enge, joho}@kth.se

Abstract. In the PCP model, a verifier is supposed to probabilistically decide if a given input belongs to some language by posing queries to a purported proof of this fact. The probability that the verifier accepts an input in the language given a correct proof is called the *completeness* c; the probability that the verifier rejects an input not in the language given any proof is called the *soundness* s. For a verifier posing q queries to the proof, the *amortized query complexity* is defined by $q/\log_2(c/s)$ if the proof is coded in binary. It is a measure of the average "efficiency" of the queries in the following sense: An ideal query should preserve the completeness and halve the soundness. If this were the case for all queries, the amortized query complexity would be exactly one.

Samorodnitsky and Trevisan [STOC 2000] gave a q-query PCP for **NP** with amortized query complexity $1 + 2/\sqrt{q} + \varepsilon$ for any constant $\varepsilon > 0$. In this paper, we examine to what extent their result can be sharpened. Using the layered label cover problem recently introduced by Dinur *et al.* [STOC 2003], we devise a new "outer verifier" that allows us to construct an "inner verifier" that uses the query bits more efficiently than earlier verifiers. This enables us to construct a PCP for **NP** that queries q positions in the proof and has amortized query complexity $1 + \sqrt{2/q} + \varepsilon$. As an immediate corollary, we also obtain an improved hardness of approximation result for the Maximum q-CSP problem.

Since the improvement compared to previous work is moderate, we then examine if there is an underlying reason for this. Our construction in this paper follows a paradigm for query efficient PCPs for **NP** outlined by many previous researchers and it combines a state-of-the-art "outer verifier" with a natural candidate for a query efficient "inner verifier". We prove in the full version of this paper that all natural attempts to construct more query efficient versions of our verifier are doomed to fail. This implies that significantly new ideas regarding proof composition and encoding of PCP proofs are required to construct PCPs for **NP** that are more query efficient than the one we propose in his paper.

1 Introduction

For more than a decade, one of the most powerful techniques for proving approximation hardness results for various types of discrete optimization prob-

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 194–205, 2005.

lems, in particular constraint satisfaction problems, has been the use of Probabilistically Checkable Proofs (PCPs) for **NP**. In the PCP model, the verifier is given an input and oracle access to an alleged proof of the fact that the input belongs to some specified language. The verifier also has access to a specified amount of random bits. Based on the random bits and the input, the verifier decides which positions in the proof it should look at. Once it has examined the positions of its choice, it uses all available information to decide if the input should be accepted or rejected. The PCP theorem [1] asserts the startling fact that any language in **NP** can be probabilistically checked by a verifier that uses logarithmic randomness, always accepts a correct proof of an input in the language, accepts proofs of inputs not in the language with probability at most 1/2, and examines a *constant* number of bits of the proof. The probability that the PCP verifier accepts a correct proof of an input in the language is called the *completeness* c, while the probability that the verifier accepts any proof of an input not in the language is called the *soundness* s. It is generally desirable to have $c \approx 1$ and s as small as possible.

PCPs using a logarithmic number of random bits can be used to prove approximation hardness results for many combinatorial optimization problems. In particular, PCPs querying a small number of bits, say q bits, are intimately connected with Boolean q-ary constraint satisfaction problems: Strong approximation hardness results follow immediately from such PCPs with high completeness and low soundness.

Håstad's approximation hardness result for linear equations mod 2 gives such a characterization [12]: The verifier in his PCP for **NP** queries three bits, has completeness $1 - \varepsilon$ and soundness $1/2 + \delta$ for arbitrary ε and δ. Allowing the verifier to make more queries to the proof is a natural way to lower the soundness even further; independent repetition of Håstad's protocol k times gives a PCP that queries $3k$ bits, has completeness at least $1 - k\varepsilon$ and soundness at most $(1/2 + \delta)^k$. Hence the soundness goes down *exponentially fast* with the number of bits read from the proof. The purpose of this paper is to study exactly how fast the soundness can go down. There are several possible measures of "fast" in this context. One is the so called *amortized query complexity*: For a PCP with q queries, the amortized query complexity is defined as $\bar{q} = q/\log(c/s)$. The task of constructing PCPs for **NP** with low amortized query complexity—as well as the related question of testing if a function is linear—has been explored previously, most notably in a sequence of papers by Trevisan with different coauthors [18, 17, 16]. The key idea in those papers is to use *dependent repetitions* of Håstad's basic protocol. The technical part of the argument then boils down to showing that this dependence does not destroy the soundness of the verifier. In this paper, we adapt and extend these previous ideas. In particular, we show that the idea of using dependent repetitions can be combined with the recently introduced *layered label cover problem* [6].

1.1 Query-Efficient Verification of NP

Another important efficiency measure for PCPs is the *free bit complexity*: A PCP has free bit complexity f if there are, for every outcome of the random bits used by the verifier, at most 2^f possible answers to the verifier's queries that make the verifier accept. Using the free bit complexity, our first main result in this paper can be written as follows:

Theorem 1. *For any integer $f \geq 2$, any positive integer $t \leq f(f-1)/2$, and any constant $\varepsilon > 0$, there is a PCP for **NP** with free bit complexity f, query complexity $f + t$, completeness $1 - \varepsilon$, and soundness $2^{-t} + \varepsilon$.*

To compare this with the previously best known result, due to Samorodnitsky and Trevisan [16], it is instructive to cast this result in terms of the amortized query complexity as a function of the number of queries:

Corollary 1. *For any integer $q \geq 3$ and any constant $\varepsilon > 0$, there is a PCP for **NP** with query complexity q and amortized query complexity $1 + \sqrt{2/q} + \varepsilon$.*

Writing the soundness of our PCP as a function of the number of queries, we also get as an immediate corollary of our main result an improved approximation hardness result for the q-CSP problem:

Corollary 2. *For any integer $q \geq 3$ and any constant $\varepsilon > 0$, it is **NP**-hard to approximate the q-CSP problem within $2^{q-\sqrt{2q-2}-1/2} - \varepsilon$.*

The previously best known construction, due to Samorodnitsky and Trevisan [16] gives amortized query complexity $1 + 2/\sqrt{q} + \varepsilon$ and hardness of approximation within $2^{q-2\sqrt{q+1}+1}$. While our improvements might at first seem moderate, we remark that it is possible to approximate the q-CSP problem within $2^{q-1.4}$ in polynomial time and that a PCP for **NP** cannot have amortized query complexity $1+1/(5q/7-1)$ unless $\mathbf{P} = \mathbf{NP}$; this follows from Hast's recent approximation algorithm for q-CSP [11]. Hence the only possible further improvements, unless $\mathbf{P} = \mathbf{NP}$, along this line of research concern the lower order term in $\bar{q}$ and the lower term in the exponent of the approximation factor—where we get an improvement by a factor of $\sqrt{2}$. We also point out that an assignment selected uniformly at random satisfies a fraction 2^{-q} of the constraints in a q-CSP problem. Hence, it is possible to construct, in polynomial time, an algorithm which is a factor of roughly 2.3 better than a random assignment. Prior to our paper, it was known that it is **NP**-hard to beat the random assignment by a factor of $4^{\sqrt{q}}$ [16]; we improve this latter factor to roughly $2.67^{\sqrt{q}}$.

Our second main result is that there seems to be an underlying reason for why our improvement compared to previous work is moderate. Our construction in this paper follows a paradigm for query efficient PCPs for **NP** outlined by many previous researchers: On a high level, it combines a state-of-the-art "outer verifier" corresponding to a so called "layered label cover problem" with a corresponding "inner verifier" that is more query efficient than previously known verifiers. There are natural ways to extend this inner verifier in certain ways to produce what, at first, looks like even more query efficient verifiers. We prove in

the full version [9] of this extended abstract, however, that all such extensions give verifiers that are *less query efficient* than our proposed verifier in the sense that the new verifiers have the same soundness as our verifier but pose more queries. This implies that significantly new ideas regarding proof composition and encoding of PCP proofs are required to construct PCPs for **NP** that are more query efficient than the one we propose in his paper.

Driven by the desire to prove strong inapproximability results for Maximum Clique another measure, the *amortized free bit complexity*, was introduced [4]. Recall that a PCP has free bit complexity f if there are, for every outcome of the random bits used by the verifier, at most 2^f possible answers to the verifier's queries that make the verifier accept. The amortized free bit complexity is then defined as $\bar{f} = f/\log(c/s)$. The intuition behind the term *free bit complexity* is that a typical verifier with free bit complexity f first reads f bits, the so called "free bits", from the proof and then performs a number of consistency checks, each involving some of the free bits and some new positions in the proof.

The PCP constructed by Samorodnitsky and Trevisan [16] has a verifier that first queries $2k$ free bits and then makes k^2 consistency checks, each of which uses two (different) free bits and one unique non-free bit. Hence the verifier queries in total $2k + k^2$ bits and it can be shown that the test has completeness $1 - \varepsilon$ and soundness $2^{-k^2} + \varepsilon$. It follows that the amortized query complexity can be written as $1 + 4/f + \varepsilon$ where f is the number of free bits used by the verifier. Under the assumption that the verifier first queries some free bits and then performs consistency checks involving two free bits where the same two free bits are present in at most one consistency check, the amortized query complexity cannot be lower than $1 + 2/(f - 1)$. Our construction in this paper achieves this goal.

Corollary 3. *For any integer $f \geq 2$ and any constant $\varepsilon > 0$, there is a PCP for **NP** with free bit complexity f and amortized query complexity $1 + 2/(f - 1) + \varepsilon$.*

Hence it is optimal among the PCPs that work as described above. One natural suggestion to improve the soundness is to perform consistency checks not only on pairs of free bits but on all possible subsets of free bits. In their paper [16], Samorodnitsky and Trevisan show that such an approach is futile for the related problem of *linearity testing*. In that setting, the "proof" is a table of a Boolean function and "consistency" simply means that this table respects linearity. For the more complex situation involving PCPs for **NP** with several tables defining different functions that should be correlated in some way, there is no known corresponding result.

As mentioned above, our second main result is an optimality result. In the above language, this result shows that it is impossible to beat the soundness bounds stated above even with a verifier that tests consistency induced not only by pairs of free bits but by *all possible Boolean predicates on the free bits*. Since all hitherto constructed PCPs with low amortized query complexity follow the paradigm outlined above, our result shows that to construct PCPs with even lower amortized query complexity, radically new ideas are needed.

1.2 Main Sources of Our Improvements

Our paper uses the fundamental idea of "recycling" query bits in linearity tests and PCPs, introduced and explored in a sequence of papers authored by Trevisan with different coauthors. For linearity tests, the construction is founded on the basic BLR linearity test [5] that selects random x and y and tests if $f(x)f(y) = f(xy)$. Trevisan [18] introduced the notion of a *graph test* iterating the basic BLR test once for every edge in the graph: Each variable in the graph corresponds to a randomly selected value and for each edge $\{x, y\}$ the test $f(x)f(y) = f(xy)$ is executed. These tests are, of course, dependent, but Trevisan [18] was able to show that they still behave essentially as independent tests for some graphs. He also used similar ideas to construct a PCP for **NP** with $\bar{q} = 2.5+\varepsilon$. Extending the analysis, both for linearity tests and PCPs, Sudan and Trevisan [17] constructed a PCP for **NP** with $\bar{q} = 1.5 + 3/(q-2) + \varepsilon$, and, finally, Samorodnitsky and Trevisan [16] constructed the PCP for **NP** mentioned above, with $\bar{q} = 1 + 2/\sqrt{q} + \varepsilon$.

Our main source of improvement is that we use a layered label cover problem recently introduced by Dinur *et al.* [6] as our underlying **NP**-hard problem. While our construction of the label cover instance is exactly the same as used by Dinur *et al.*, we need in this paper a stronger hardness result than they state in their paper. We also use a new way to connect this layered label cover problem—or "outer verifier" as it is often called in PCP jargon—with the actual PCP—or "inner verifier". Loosely speaking, while previous constructions checked pairwise consistency between pairs (W_0, W_i) of proof tables for some number of W_i:s, our use of the layered label cover problem as an outer verifier enables us to devise an inner verifier that tests pairwise consistency between all pairs (W_i, W_j).

On a slightly more concrete level, the Samorodnitsky-Trevisan verifier queries k free bits in the table W_0 and then one free bit in each of the tables W_i. But it does not use all possible pairs of free bits for consistency checks, only (roughly) half of them. The intuitive reason why our construction is more efficient with respect to the free bits is that we check pairwise consistency between tables (W_i, W_j) by first querying one free bit from each table and then using *all pairs of free bits* for a consistency check.

Håstad and Wigderson [14] have presented a much simplified analysis of the Samorodnitsky-Trevisan PCP; we use the same kind of analysis in this paper.

2 Background

Our construction, as most other PCP constructions, use the powerful technique of *proof composition*, introduced by Arora and Safra [2]. The main idea in such constructions is to combine two verifiers that optimize different type of parameters into one verifier that optimizes both parameters. In particular, it is common to use an "outer verifier" that queries a proof containing values from some large domain and an "inner verifier" whose proof contains purported answers to the outer verifier encoded with some error correcting code.

To get an intuitive understanding of both our construction and earlier related constructions, it is instructive to recall a high-level description of Håstad's PCP for **NP** with three queries—two of which are free—completeness $1-\varepsilon$ and soundness $1/2+\varepsilon$ [12]. The outer verifier used by Håstad can be described in several different ways. While Håstad describes it in terms of a two-prover one-round protocol for a certain class of 3-Sat formulae, we chose one of the alternate formulations: a label cover problem with a certain "gap" property.

Definition 1. *Given two sets R and S, an* instance of the label cover problem on R and S *is a set $\Psi \subseteq V \times \Phi$, where V and Φ are sets of variables with ranges R and S, respectively, with an onto function $\pi_{v\phi}: S \to R$ for every $(v,\phi) \in \Psi$. The instance is* regular *if every $v \in V$ occurs in the same number of pairs in Ψ and every $\phi \in \Phi$ occurs in the same number of pairs in Ψ.*

The following theorem is a consequence of the PCP theorem [1] combined with a certain regularization procedure [10] and the parallel repetition theorem [15].

Theorem 2 ([1, 10, 15]). *There exists a universal constant $\mu > 0$ such that for every large enough constant u there exist sets R and S with $2^u \le |R| \le |S| \le 7^u$ such that it is **NP**-hard to distinguish between the following two cases given a regular instance $\Psi \subseteq V \times \Phi$ of a label cover problem on R and S:*

***YES:** There exist assignments $\Pi_V: V \to R$ and $\Pi_\Phi: \Phi \to S$ such that for every $(v,\phi) \in \Psi$, $\Pi_V(v) = \pi_{v\phi}(\Pi_\Phi(\phi))$.*

***NO:** There are no assignments $\Pi_V: V \to R$ and $\Pi_\Phi: \Phi \to S$ such that for more than a fraction $|R|^{-\mu}$ of the pairs $(v,\phi) \in \Psi$, $\Pi_V(v) = \pi_{v\phi}(\Pi_\Phi(\phi))$.*

A regular label cover problem Ψ on (V,R) and (Φ,S) can be viewed as a bi-regular bipartite graph with the node set $V \cup \Phi$ and an edge $\{v,\phi\}$ for every $(v,\phi) \in \Psi$. The objective is then to assign labels to the nodes in such a way that for every edge $\{v,\phi\}$, the constraint $\Pi_V(v) = \pi_{v\phi}(\Pi_\Phi(\phi))$ holds. The above theorem says that it is **NP**-hard to distinguish the case when this is possible from the case when it is only possible to satisfy a tiny fraction of the constraints.

The inner verifier in Håstad's PCP has access to several sub-tables: one for each $v \in V$ and one for each $\phi \in \Phi$. These tables contain purported encodings of the labels assigned to the corresponding variable/vertex. The labels are encoded by the so called *long code*, first used in the context of PCPs by Bellare, Goldreich and Sudan [3]. The long code $A_{w,\sigma}$ of an assignment/label σ to some variable/vertex w assuming values in some set R is a function mapping $f \in \{-1,1\}^R$ to a value in $\{-1,1\}$ by the map $A_{w,\sigma}(f) = f(\sigma)$.

To check the proof, Håstad's inner verifier essentially selects a random $(v,\phi) \in \Psi$, random functions f and g and then checks if $A_v(f)A_\phi(g) = A_\phi(fg)$. (The test is actually slightly more complicated. In fact, the test as written above does not quite work, but we ignore this here, and for the rest of this section, for the sake of clarity.) There are two important points to note here: 1) There are two free queries, $A_v(f)$ and $A_\phi(g)$, and one "check", $A_\phi(fg)$. 2) The queries involve tables of two types, one v-table and one ϕ-table.

Trevisan put forward the idea that the free queries could be "recycled". In his first paper [18], he recycled the query $A_\phi(g)$ and made two checks: $A_v(f_1)A_\phi(g) = A_\phi(f_1 g)$ and $A_v(f_2)A_\phi(g) = A_\phi(f_2 g)$. These checks are dependent, but it turns out that the dependence is weak enough to give soundness $1/4 + \varepsilon$ at the price of 3 free and 2 non-free queries. Hence this construction has amortized query complexity $2.5 + \varepsilon$.

Sudan and Trevisan [17] extended the analysis by using a modification of the outer verifier: From an instance $\Psi \subseteq V \times \Phi$ of the basic label cover problem, a modified label cover instance is constructed as follows: The instance consists of all tuples $(v, \phi_1, \ldots, \phi_k)$ such that $(v, \phi_j) \in \Psi$ for all j; denote by $\tilde{\Psi}$ this set of tuples. The objective is then to construct labelings Π_V and Π_Φ, respectively, of V and Ψ, respectively, such that for all $(v, \phi_1, \ldots, \phi_k) \in \tilde{\Psi}$, the constraint $\Pi_V(v) = \pi_{v\phi_j}(\Pi_\Phi(\phi_j))$ holds *for every* j. Sudan and Trevisan show in their paper that it is **NP**-hard to distinguish the case when this is possible from the case when it holds for at most a tiny fraction of the tuples in $\tilde{\Psi}$ that $\Pi_V(v) = \pi_{v\phi_j}(\Pi_\Phi(\phi_j))$ *for some* j.

Using this new outer verifier, Sudan and Trevisan [17] were able to analyze an inner verifier that selects random functions f_1, f_2 and $g_1, \ldots, g_k$, and then checks $A_v(f_i)A_{\phi_j}(g_j) = A_{\phi_j}(f_i g_j)$ for all i and j. This gives amortized query complexity $1.5 + \varepsilon$. Finally, Samorodnitsky and Trevisan [16], using the same outer verifier, were able to analyze the even more general case where the verifier selects v and $\phi_1, \ldots, \phi_k$ as above, random functions $f_1, \ldots, f_k$ and $g_1, \ldots, g_k$ and then checks $A_v(f_i)A_{\phi_j}(g_j) = A_{\phi_j}(f_i g_j)$ for all i and j. This construction gives amortized query complexity $1 + \varepsilon$.

3 A Multi-layered Label Cover Problem

The outer verifier devised by Sudan and Trevisan [17] has the property that one v and several ϕ_j connected to v are used. There is one "check" for every pair (f, g) of free queries where f is queried from table A_v and g is queried from table A_{ϕ_j}. Hence, all of the above constructions check pairwise consistency between *the same node* v and *several nodes* ϕ_j *connected to* v in the underlying label cover instance. Our main idea to push the "recycling" further is to instead select $k + 1$ nodes from a tailored label cover instance in such a way that we can check pairwise consistency between *every pair of nodes*. This tailored label cover instance was recently devised by Dinur *et al.* [6] and used by them to prove strong approximation hardness results for hypergraph vertex cover.

Recall that the objective of the label cover problem on R and S is to assign to nodes in V labels from R and to nodes in Φ labels from S in such a way that many edges are "satisfied" in the sense that the corresponding projections $\{\pi_{v\phi}\}_{(v,\phi)\in\Psi}$ are satisfied. The label cover used as a basis for our construction is similar in spirit, but the instance has more structure. It can be viewed as a k-wise parallel repetition of the original label cover problem but with the objective to construct certain "hybrid" labelings. Given an instance Ψ of the basic label

cover problem from Definition 1 we construct a corresponding instance of our layered label cover problem as all k-tuples of elements from Ψ; hence the instance is simply the k-wise Cartesian product Ψ^k and some arbitrary $\psi \in \Psi^k$ can be written as $\psi = ((v_1, \phi_1), \ldots, (v_k, \phi_k))$.

In an "ordinary" k-wise parallel repetition the goal would be to construct an assignment to all k-tuples $(v_1, \ldots, v_k) \in V^k$ and an assignment to all k-tuples $(\phi_1, \ldots, \phi_k) \in \Phi^k$ in such a way that the assignments are consistent, where "consistent" means that each coordinate satisfies the corresponding projection $\pi_{v_j \phi_j}$ from the basic label cover problem. The goal in our layered label cover problem is to construct in total $k+1$ assignments: not only one assignment to tuples in V^k and one assignment to tuples in Φ^k, but also $k-1$ "hybrid" assignments to "hybrid" tuples of the form $(\phi_1, \ldots, \phi_t, v_{t+1}, \ldots, v_k) \in \Phi^t \times V^{k-t}$. We say that a tuple belongs to *layer* t if the first t coordinates of the tuple contain values from Φ and the remaining coordinates contain values from V. With this notation, layer 0 corresponds to tuples containing only variables from V, layer k corresponds to tuples containing only variables from Φ, and each $\psi \in \Psi^k$ corresponds to $k+1$ tuples, one in each layer. (Readers very familiar with the presentation of the layered verifier in [6] may want to notice that what we call a tuple corresponding to $\psi \in \Psi^k$ corresponds to a clique in the multi-partite graph defined in the proof of Theorem 3.3 in [6].) Write $\psi = ((v_1, \phi_1), \ldots, (v_k, \phi_k))$ and consider the tuples corresponding to ψ. Assignments to these tuples produce tuples of values from R (values to V-variables) and S (values to Φ-variables). There is an obvious way to require consistency between these assignments. For two tuples $(\phi_1, \ldots, \phi_i, v_{i+1}, \ldots, v_k)$ and $(\phi_1, \ldots, \phi_j, v_{j+1}, \ldots, v_k)$ where $i < j$ and corresponding assignments $(s_1, \ldots, s_i, r_{i+1}, \ldots, r_k)$ and $(s'_1, \ldots, s'_j, r'_{j+1}, \ldots, r'_k)$, we use the projections $\pi_{v\phi}$ from the basic label cover instance to define what it means for the assignments to be consistent: $s_t = s'_t$ $(t \leq i)$ and $r_t = \pi_{v_t \phi_t}(s'_t)$ $(i < t \leq j)$ and $r_t = r'_t$ $(t > j)$.

Definition 2. *Given R, S, V, Φ, Ψ, and $\{\pi_{v\phi}\}_{(v,\phi)\in\Psi}$ as in Definition 1, denote an arbitrary $\psi \in \Psi^k$ by the vector $((v_1, \phi_1), \ldots, (v_k, \phi_k))$ and define the shorthand*

$$\pi_{\psi, j \to i} \colon S^j \times R^{k-j} \to S^i \times R^{k-i}$$

as the function mapping $(s_1, \ldots, s_j, r_{j+1}, \ldots, r_k)$ to

$$(s_1, \ldots, s_i, \pi_{v_{i+1}\phi_{i+1}}(s_{i+1}), \ldots, \pi_{v_j \phi_j}(s_j), r_{j+1}, \ldots, r_k).$$

The objective of our label cover problem is to produce $k+1$ assignments such that all constraints $\pi_{\psi, j \to i}$ are satisfied. It is straightforward to see that if the case "YES" from Theorem 2 holds for the original instance Ψ, the assignments Π_V and Π_Ψ guaranteed in this case can be used to construct such assignments.

Our "inner verifier" described in § 4 checks a purported proof of an assignment to the nodes in a layered label cover problem. For a correctly encoded proof of an assignment described in the "YES" case of Theorem 2, it is easy to see that our verifier almost always accepts. For the reverse direction, we show

that if our "inner verifier" accepts with only a small advantage over a random assignment, it is possible to construct an assignment to the underlying basic label cover problem that contradicts the "NO" case in Theorem 2. Hence, this in some sense reduces the hardness of the layered label cover problem to hardness of the basic label cover problem. Dinur *et al.* [6] use the same idea in their paper but formulate the hardness of the layered label cover problem as a separate theorem. It is possible to do so also in our case, but the formulation of this separate theorem is bit more complicated, and in fact somewhat unnatural, due to some fairly intricate dependencies between the various layers in the "inner verifier". By doing the reduction to the standard label cover problem as a part of the analysis of the "inner verifier", we avoid this complication. The interested reader can find the more complicated proof in an earlier version of this paper [9].

4 A Recycling PCP

We now describe a verifier with access to several proof tables, each containing a purported encoding of a label for a node in the layered label cover instance. The verifier bases its decision to accept or reject on some carefully selected values from tables corresponding to the nodes created from some $\psi \in \Psi^k$ as described in § 3. The main properties of the verifier can loosely be described as follows: 1) If case "YES" in Theorem 2 holds, there exists a proof that makes the verifier accept with high probability; 2) If case "NO" in Theorem 2 holds, the PCP verifier accepts with very small probability.

4.1 The Proof

The proof in our PCP contains for each i such that $0 \leq i \leq k$ and each $w \in \Phi^i \times V^{k-i}$ a table A_w of size $2^{|S|^i |R|^{k-i}}$ which should be interpreted as the values of a function from $\{-1,1\}^{S^i \times R^{k-i}}$ to $\{-1,1\}$. For such a w and some function $f\colon S^i \times R^{k-i} \to \{-1,1\}$, $A_w(f)$ denotes the position in table A_w corresponding to f.

Definition 3. *Given R, S, V, Φ, Ψ, and an arbitrary $\psi \in \Psi^k$, define $A_{\psi,i}$ to be the table A_w where $w \in \Phi^i \times V^{k-i}$ is defined from ψ as follows: Write ψ as $((v_1,\phi_1),\ldots,(v_k,\phi_k))$. Then w is defined as the k-tuple where position t contains ϕ_t if $t \leq i$ and v_t if $t > i$.*

We remark here, that a function $f\colon S^i \times R^{k-i} \to \{-1,1\}$ can be used to index into A_w for *any* $w \in \Phi^i \times V^{k-i}$. This simplifies our analysis below. Also, all tables are assumed to be *folded*, i.e., $A_w(f) = -A_w(-f)$ for all w and f; this property can be enforced by certain access conventions in the verifier. Our only need for folding comes from the fact that certain Fourier coefficients of the tables in the proof are guaranteed to vanish for folded tables; see Håstad's paper [12–§§ 2.4–2.6] for details.

4.2 The Verifier

The verifier is parameterized by a rational $\varepsilon > 0$, an integer $k > 0$ and a nonempty subset P of $\{(i,j) : 0 \leq i < j \leq k\}$. The actions of the verifier are follows:

1. Select $\psi \in \Psi^k$ uniformly at random.
2. Select $f_i \colon S^i \times R^{k-i} \to \{-1,1\}$ for each i such that $0 \leq i \leq k$ independently uniformly at random.
3. Select functions $e_{ij} \colon S^j \times R^{k-j} \to \{-1,1\}$ for each $(i,j) \in P$ by selecting $e_{ij}(x)$ independently at random to be 1 with probability $1-\varepsilon$ and -1 otherwise.
4. Accept if $A_{\psi,i}(f_i)A_{\psi,j}(f_j) = A_{\psi,j}((f_i \circ \pi_{\psi,j\to i})f_j e_{ij})$ for every $(i,j) \in P$; reject otherwise. Here, $\pi_{\psi,j\to i}$ is defined as described in Definition 2 and $A_{\psi,i}$ is defined as in Definition 3.

Before embarking on the analysis of the soundness of the verifier, let us note some immediate consequences of the construction.

Lemma 1. *The verifier has completeness at least* $1 - |P|\varepsilon$*, query complexity* $|P|+k+1$*, and free bit complexity* $k+1$*. Moreover, it uses at most* $k\lceil \log |\Psi| \rceil + (k+1)|S|^k + |P||S|^k \lceil \log \varepsilon^{-1} \rceil$ *random bits.*

Proof. Consider a proof consisting of correct encodings of labels satisfying the "YES" case in Theorem 2. Let $\{r_i\}_{i=0}^k$ denote the labels to the nodes corresponding to some arbitrary ψ. Then $A_{\psi,i}(f_i) = f_i(r_i)$ and it can be seen that $A_{\psi,j}((f_i \circ \pi_{\psi,j\to i})f_j e_{ij}) = f_i(\pi_{\psi,j\to i}(r_j))f_j(r_j)e_{ij}(r_j) = f_i(r_i)f_j(r_j)e_{ij}(r_j)$. Hence, the verifier accepts if $e_{ij}(r_j) = 1$ for all $(i,j) \in P$. This establishes the completeness. The other claims are verified by straightforward inspection of the protocol.

4.3 Analysis of the Soundness

The acceptance predicate is arithmetized as usual:

$$\prod_{(i,j)\in P} \frac{1 + A_{\psi,i}(f_i)A_{\psi,j}(f_j)A_{\psi,j}((f_i \circ \pi_{\psi,j\to i})f_j e_{ij})}{2}$$

is an indicator for the event that the verifier accepts. This expression can be rewritten as

$$2^{-|P|} \sum_{S \subseteq P} \prod_{(i,j)\in S} A_{\psi,i}(f_i)A_{\psi,j}(f_j)A_{\psi,j}((f_i \circ \pi_{\psi,j\to i})f_j e_{ij}).$$

If the expectation of this is at least $2^{-|P|}+\delta$, there must exist some nonempty $S \subseteq P$ such that

$$\mathrm{E}\left[\prod_{(i,j)\in S} A_{\psi,i}(f_i)A_{\psi,j}(f_j)A_{\psi,j}((f_i \circ \pi_{\psi,j\to i})f_j e_{ij})\right] \geq \delta. \tag{1}$$

We now consider one such S and prove that it is possible to extract assignments violating property "NO" in Theorem 2 provided that the above inequality holds. The main idea in the proof of this result is to split the product inside the expectation in inequality (1) above into three parts. The Fourier transform of two of these parts are then used to devise functions Π_V and Π_Φ that violate property "NO" in Theorem 2. The precise way to split the product is carefully chosen so that the resulting functions Π_V and Π_Φ are guaranteed to depend only on certain variables, as required by Theorem 2.

Lemma 2. *Suppose that a PCP is constructed as described above from an instance of the layered label cover problem with* $|R| > (\varepsilon^{-1}\delta^{-2}/4)^{1/\mu}$ *and that*

$$\mathrm{E}\left[\prod_{(i,j)\in S} A_{\psi,i}(f_i)A_{\psi,j}(f_j)A_{\psi,j}((f_i \circ \pi_{\psi,j\to i})f_j e_{ij})\right] \geq \delta, \tag{2}$$

where the expectation is over the random choices of the verifier, for some non-empty $S \subseteq P$*. Then property "NO" in Theorem 2 is violated.*

The complete proof of Lemma 2 is omitted due to space limitations; a proof using the slightly complicated formalism mentioned earlier is given in [9].

5 Future Work: Larger Domains and Perfect Completeness

It is straightforward to extend our result to PCPs over arbitrary finite Abelian groups using the, by now, standard approach [7, 8, 12].

Håstad and Khot [13] proved results similar to those of Samorodnitsky and Trevisan [16] but for the case when the verifier has completeness 1 rather than $1 - \varepsilon$. In particular, they provide a protocol that iterates a basic test involving four free and one non-free bit, the entire test queries $4k + k^2$ bits and has soundness 2^{-k^2}. They also provide an *adaptive* protocol, i.e., a protocol where the verifier's queries may depend on the answers to earlier queries, that queries $2k + k^2$ bits and has soundness 2^{-k^2}. It seems that those results can also be adapted with reasonably straightforward but rather tedious calculations to our setting, the main complication being the adaptive protocol with $2k + k^2$ queries and soundness 2^{-k^2}.

An interesting direction for future research is to try to construct a PCP with perfect completeness that iterates a predicate with fewer free bits but still only one non-free bit. If such a predicate were found and combined with the techniques of this paper, it could potentially give results comparable to ours—we query approximately $\sqrt{2}k + k^2$ bits to get soundness 2^{-k^2}—but with perfect rather than near-perfect completeness.

In the full version [9] of this extended abstract, we also prove that it is impossible to improve the soundness of the PCP by adding additional queries of a certain form. As a consequence, new ideas are required to construct PCPs that are even more query efficient than those proposed in this paper.

Acknowledgments

We are most grateful to the anonymous referees—in particular the referee with the suggestion that many of the pairs (v_i, ϕ_i) be fixed early in the proof of Lemma 2. This change simplified, and shortened, our proof considerably.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
2. S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.
3. M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs and non-approximability—towards tight results. *SIAM J. Comput.*, 27(3):804–915, 1998.
4. M. Bellare and M. Sudan. Improved non-approximability results. In *Proc. 26th STOC*, pages 184–193, 1994.
5. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.
6. I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. In *Proc. 35th STOC*, pages 595–601, 2003.
7. L. Engebretsen. The nonapproximability of non-Boolean predicates. *SIAM J. Disc. Math.*, 18(1):114–129, 2004.
8. L. Engebretsen and J. Holmerin. Three-query PCPs with perfect completeness over non-Boolean domains. In *Proc. 18th CCC*, pages 284–299, 2003.
9. L. Engebretsen and J. Holmerin. More efficient queries in PCPs for NP and improved approximation hardness of maximum CSP. Technical Report TRITA-NA-0409, Royal Institute of Technology, April 2004.
10. U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.
11. G. Hast. Approximating MAX kCSP using random restrictions. In K. Jansen, S. Khanna, J. D. P. Rolim, and D. Ron, editors, *Proc. 7th APPROX*, 2004, volume 3122 of *Lecture Notes in Computer Science*, pages 151–162.
12. J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.
13. J. Håstad and S. Khot. Query efficient PCPs with perfect completeness. In *Proc. 42nd FOCS*, pages 610–619, 2001.
14. J. Håstad and A. Wigderson. Simple analysis of graph tests for linearity and PCP. *Random Struct. and Alg.*, 22(2):139–160, 2003.
15. R. Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998.
16. A. Samorodnitsky and L. Trevisan. A PCP characterization of NP with optimal amortized query complexity. In *Proc. 32nd STOC*, pages 191–199, 2000.
17. M. Sudan and L. Trevisan. Probabilistically checkable proofs with low amortized query complexity. In *Proc. 39th FOCS*, pages 18–27, 1998.
18. L. Trevisan. Recycling queries in PCPs and in linearity tests. In *Proc. 30th STOC*, pages 299–308, 1998.

Three Optimal Algorithms for Balls of Three Colors*

Zdeněk Dvořák[1], Vít Jelínek[1], Daniel Král'[1,**], Jan Kynčl[1], and Michael Saks[2,***]

[1] Department of Applied Mathematics and,
Institute for Theoretical Computer Science (ITI)†,
Charles University, Malostranské náměstí 25,
118 00 Prague, Czech Republic
{rakdver, jelinek, kral, kyncl}@kam.mff.cuni.cz
[2] Department of Mathematics, Rutgers, the State University of NJ
110 Frelinghuysen Road, Piscataway, NJ 08854-8019, USA
saks@math.rutgers.edu

Abstract. We consider a game played by two players, Paul and Carol. Carol fixes a coloring of n balls with three colors. At each step, Paul chooses a pair of balls and asks Carol whether the balls have the same color. Carol truthfully answers yes or no. In the Plurality problem, Paul wants to find a ball with the most common color. In the Partition problem, Paul wants to partition the balls according to their colors. He wants to ask Carol the least number of questions to reach his goal. We find optimal deterministic and probabilistic strategies for the Partition problem and an asymptotically optimal probabilistic strategy for the Plurality problem.

1 Introduction

We study a game played by two players, Paul and Carol, in which Paul wants to determine a certain property of the input based on Carol's answers. Carol fixes a coloring of n balls by k colors. Paul does not know the coloring of the balls. At each step, he chooses two balls and asks Carol whether they have the same color. Carol truthfully answers `YES` or `NO`. Paul wants to ask the least number of questions in the worst case to determine the desired property of the coloring.

The first problem of this kind which was considered is the *Majority problem*, in which Paul wants to find a ball b such that the number of balls colored with

* The REU program, during which the research was conducted, is supported by grant KONTAKT ME 521.

** A postdoctoral fellow at Institute for Mathematics of the Technical University Berlin since October 2004.

*** Research supported in part by NSF grant CCR-9988526.

† Institute for Theoretical Computer Science is supported by Ministry of Education of Czech Republic as project LN00A056.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 206–217, 2005.

the same color as b is greater than $n/2$, or to declare that there is no such ball. Saks and Werman [11], later Alonso, Reingold and Schott [4], showed that $n - \nu(n)$ questions are necessary and sufficient for Paul to resolve the Majority problem with n balls of two colors, where $\nu(n)$ is the number of 1's in the binary representation of n. Fisher and Salzberg [6] showed that $\lceil 3n/2 \rceil - 2$ questions are necessary and sufficient to solve the Majority problem with n balls and an unrestricted number of colors. Some variants of the Majority problem were also considered in [1, 8].

In this paper, we consider the Plurality problem, introduced by Aigner et al. [2], and the Partition problem. In the *Plurality problem*, Paul seeks for a ball such that the number of balls with the same color exceeds the number of balls of any other color (or he finds out that there is a tie between two or more different colors). In the *Partition problem*, Paul wants to partition the balls according to their colors. Aigner et al. [2, 3] found a strategy to solve the Plurality problem with n balls of three colors such that Paul asks at most $\lfloor \frac{5n}{3} \rfloor - 2$ questions. On the other hand, Carol can force Paul to ask at least $3\lfloor \frac{n}{2} \rfloor - 2$ questions (when the balls can have three colors) and at least $\Omega(nk)$ questions when the balls can have k colors. The Plurality problem in the probabilistic setting has been studied in [7] where the upper and lower bounds of $\frac{2}{3}n + o(n)$ on the number of questions for balls of two colors and the lower bound $\Omega(nk)$ for balls of k colors can be found. Another problem similar to the Plurality problem was studied by Srivastava [12].

We focus on the case when the balls are colored with three colors and present a probabilistic strategy for the Plurality problem and both deterministic and probabilistic strategies for the Partition problem. In the probabilistic setting, Paul may flip coins and use the outcome to choose his questions. The quality of a probabilistic strategy is measured as the maximum of the expected number of Paul's questions over all inputs. For both the deterministic and probabilistic strategies, we assume that Carol knows the strategy and chooses the worst coloring. In the deterministic setting, Carol can also choose the coloring on-line in response to the questions. This is not appropriate for probabilistic strategies, since we assume that Carol does not know outcome of the coin flips.

Our results are summarized in Table 1. In the case of deterministic strategy of the Partition problem, we provide matching lower and upper bounds on the number of Paul's questions. The result can be generalized for balls with arbitrary

Table 1. Bounds for the Plurality and Partition problems with n balls of three colors

The problem	Lower bound	Upper bound
The Plurality Problem		
Deterministic strategy [2]	$3\lfloor \frac{n}{2} \rfloor - 2$	$\lfloor \frac{5n}{3} \rfloor - 2$
Probabilistic strategy	$\frac{3}{2}n - O\left(\sqrt{n \log n}\right)$	$\frac{3}{2}n + O(1)$
The Partition Problem		
Deterministic strategy	$2n - 3$	$2n - 3$
Probabilistic strategy	$\frac{5}{3}n - \frac{8}{3}$	$\frac{5}{3}n - \frac{8}{3} + o(1)$

number of colors (see Section 6). In the probabilistic setting, our bounds for the Partition problem match up to the $o(1)$ term. For the Plurality problem, we managed to prove a lower bound in the probabilistic setting which is close to the lower bound proved by Aigner et al. [2] in the (more constrained) deterministic setting and we show that the lower bound is asymptotically tight by providing a matching upper bound.

2 Notation

In this section, we introduce a compact way of representing a deterministic strategy for Paul, and the state of his information about the colors of the balls as the strategy proceeds.

The game of Paul and Carol can be viewed as a game on a graph whose vertices are the balls. Initially the graph is empty. At each turn, Paul chooses a pair of nonadjacent vertices and adds that edge to the graph. Carol then colors the edge by red if the two vertices have the same color, or by blue if the two vertices have a different color. This edge-colored graph represents the state of Paul's knowledge and is referred to as *Paul's graph*. Notice that each connected component of red edges consists of vertices corresponding to balls with the same color. *The reduced graph* has as its vertex set each of these red connected components, with two components joined if there is at least one blue edge between them. In the Partition problem with k colors, the game ends when the reduced graph is uniquely vertex k-colorable (up to a permutation of the colors). In the Plurality problem with k colors, the game ends when there is a vertex v in the reduced graph with the property that in every vertex k-coloring, v belongs to a largest color class, where the size of a color class is the sum of orders of the contracted components.

A deterministic strategy for Paul can be represented by a rooted binary tree in which the left edge from each internal vertex is colored with red and the right edge with blue. The root is associated with Paul's first question. The left subtree represents Paul's strategy for the case when Carol answers that the colors of the balls are the same, and the right one for the case when she answers that they are different. At each node, information on the coloring obtained so far can be represented by Paul's graph. For a given coloring of the balls, there is a unique path from the root to a leaf in the tree. This path in the tree is called the *computation path*.

3 Yao's Principle

Yao [13] proposed a technique for proving lower bounds on probabilistic algorithms which is based on the minimax principle from game theory. Informally, to prove such a lower bound, instead of constructing a hard coloring of the balls for every probabilistic algorithm, it is enough to find a probability distribution on colorings which is hard for every deterministic algorithm. Yao's technique

applies to our setting, too. We formulate the principle formally using our notation as a proposition. Since the proof follows the same line as in the original setting, we do not include it and refer the reader, e.g., to [10–Subsection 2.2.2] if necessary.

Proposition 1. *If for the Plurality or Partition problem with n balls of k colors, there exists a probability distribution on colorings of the balls such that the expected number of Paul's questions is at least K for each deterministic strategy, then for each probabilistic strategy there exists a coloring $\mathcal{I}$ of the balls such that the expected number of Paul's questions for the coloring $\mathcal{I}$ is at least K.*

4 Probabilistic Strategy for the Plurality Problem

We can now provide the probabilistic strategy for the Plurality problem:

Theorem 1. *There is a probabilistic strategy for the Plurality problem with n balls of three colors such that the expected number of Paul's questions does not exceed $\frac{3}{2}n + O(1)$ for any coloring of the balls.*

Proof. Fix a coloring of the balls and choose any subset $\mathcal{B}_0$ of $3n'$ balls from the input, where $n' = \lfloor \frac{n}{3} \rfloor$. Partition randomly the set $\mathcal{B}_0$ into n' ordered triples (a_i, b_i, c_i), $1 \leq i \leq n'$. For each i, $1 \leq i \leq n'$, Paul asks Carol whether the balls a_i and b_i have the same color and whether the balls b_i and c_i have the same color. If Carol answers in both the cases that the balls have different colors, Paul asks, in addition, whether the colors of the balls a_i and c_i are the same.

Based on Carol's answers, Paul is able to classify the triples into three types:

Type A. All the three balls of the triple have the same color. This is the case when Carol answers both the initial questions positively.

Type B. Two balls of the triple have the same color, but the remaining one has a different color. This is the case when Carol answers one of the initial questions positively and the other one negatively, or both the initial questions negatively and the additional question positively.

Type C. All the three balls have different colors. This is the case when Carol answers the initial questions and the additional question negatively.

Paul now chooses randomly and independently a representative ball from each triple of type A or B. Let $\mathcal{B}$ be the set of (at most n') chosen balls. In addition, he chooses randomly a ball d from $\mathcal{B}_0$.

For each ball from the set $\mathcal{B}$, Paul asks Carol whether its color is the same as the color of d. Let $\mathcal{B}' \subseteq \mathcal{B}$ be the set of the balls whose colors are different. Paul chooses arbitrarily a ball $d' \in \mathcal{B}'$ and compares the ball d' with the remaining balls of $\mathcal{B}'$. Paul is able to determine the partition of the balls of $\mathcal{B}$ according to their colors: the balls of $\mathcal{B} \setminus \mathcal{B}'$, the balls of $\mathcal{B}'$ which have the same color as the ball d' and the balls of $\mathcal{B}'$ whose color is different from the color of d'.

Finally, Paul determines the partition of all the balls from the triples of type A or B. The balls contained in a triple of type A have the same color as its

representative. In the case of a triple of type B, Paul asks Carol whether the colors of balls d_1 and d_2 are the same, where d_1 is a ball of the triple whose color is different from the color of the representative, and d_2 is a ball of $\mathcal{B}$ whose color is different from the color of the representative. In this way, Paul obtains the partition of all the balls from triples of type A or B according to their colors.

After at most $2(n-3n')$ additional questions, Paul knows the partition of the balls of $\overline{\mathcal{B}}$ according to their colors, where $\overline{\mathcal{B}}$ is the set of all the n balls except for the balls of $\mathcal{B}_0$ which are contained in triples of type C. Since each triple of type C contains one ball of each of the three colors, the plurality color of the balls of $\overline{\mathcal{B}}$ is also the plurality color of all the balls. If there is no plurality color in $\overline{\mathcal{B}}$, then there is no plurality color in the original problem either.

Before we formally analyze the described strategy, we explain some ideas behind it. Let α, β and γ be the fractions of the balls of each of the colors among the balls of $\mathcal{B}_0$. If the ratios α, β and γ are close to $1/3$, then a lot of the balls belong to the triples of type C. Clearly, such balls can be removed from the problem and we solve the Plurality problem for the remaining balls (this reduces the size of the problem). However, if the ratios α, β and γ are unbalanced, then the previous fails to work. But in this case, with high probability, the ball d has the same color as a lot of the balls of $\mathcal{B}$ and Paul does not need to compare too many balls of $\mathcal{B}$ with the ball d'.

We are now ready to start estimating the expected number of Paul's questions. The expected numbers of triples of each type are the following:

- $\left(\alpha^3+\beta^3+\gamma^3+O\left(\frac{1}{n'}\right)\right)n'$ triples of type A,
- $\left(3(\alpha^2\beta+\alpha^2\gamma+\beta^2\alpha+\beta^2\gamma+\gamma^2\alpha+\gamma^2\beta)+O\left(\frac{1}{n'}\right)\right)n'$ triples of type B, and
- $\left(6\alpha\beta\gamma+O\left(\frac{1}{n'}\right)\right)n'$ triples of type C.

The expected numbers of Paul's questions to determine the type of the triple are 2, $\frac{7}{3}$ and 3 for types A, B and C, respectively. Therefore, the expected number of Paul's questions to determine the types of all the triples is:

$$\left(2(\alpha^3+\beta^3+\gamma^3)+7(\alpha^2\beta+\alpha^2\gamma+\beta^2\alpha+\beta^2\gamma+\gamma^2\alpha+\gamma^2\beta)+18\alpha\beta\gamma\right)n'+O(1) \tag{1}$$

Next, we compute the expected number of Paul's questions to determine the partition of the balls of $\mathcal{B}$. Fix a single ball b_0 out of the $3n'$ balls. Assume that the color of b_0 is the color with the fraction α. Since the probability that the ball b_0 is in a triple of type C is $2\beta\gamma+O(\frac{1}{n'})$, the ball b_0 is contained in the set $\mathcal{B}$ with the probability $\frac{1}{3}-\frac{2}{3}\beta\gamma+O\left(\frac{1}{n'}\right)$. If $b_0\in\mathcal{B}$ and the colors of the balls b_0 and d are the same, Paul asks Carol a single question; if $b_0\in\mathcal{B}$, the colors of b_0 and d are different, and $b_0\neq d'$, he asks two questions. The former is the case with the probability α. Hence, if $b_0\in\mathcal{B}$, Paul asks $2-\alpha$ questions on average. We may conclude that in this stage, the expected number of Paul's questions involving balls of the color with the fraction α does not exceed:

$$3\alpha n'\left(\frac{1}{3}-\frac{2}{3}\beta\gamma+O\left(\frac{1}{n'}\right)\right)(2-\alpha)$$

Hence, the expected number of the questions to determine the partition of $\mathcal{B}$ is:

$$\begin{aligned}
&3\alpha n' \left(\frac{1}{3} - \frac{2}{3}\beta\gamma + O\left(\frac{1}{n'}\right)\right)(2-\alpha) + 3\beta n' \left(\frac{1}{3} - \frac{2}{3}\alpha\gamma + O\left(\frac{1}{n'}\right)\right)(2-\beta) \\
&+ 3\gamma n' \left(\frac{1}{3} - \frac{2}{3}\alpha\beta + O\left(\frac{1}{n'}\right)\right)(2-\gamma) \\
&= (2-\alpha^2-\beta^2-\gamma^2)n' - 2\alpha\beta\gamma n'(6-\alpha-\beta-\gamma) + O(1) \\
&= (2-\alpha^2-\beta^2-\gamma^2)n' - 10\alpha\beta\gamma n' + O(1) \qquad (2)
\end{aligned}$$

Next, Paul asks Carol a single question for each triple of type B. The expected number of such questions is:

$$3(\alpha^2\beta + \alpha^2\gamma + \beta^2\alpha + \beta^2\gamma + \gamma^2\alpha + \gamma^2\beta)n' + O(1) \qquad (3)$$

Finally, Paul asks at most $2(n-3n') \leq 4$ questions to find the partition of $\overline{\mathcal{B}}$.

The expected number of all the questions asked by Paul is given by the sum of (1), (2) and (3), which is equal to the following expression:

$$\begin{aligned}
&\big(2 + 2(\alpha^3+\beta^3+\gamma^3) + 10(\alpha^2\beta + \alpha^2\gamma + \beta^2\alpha + \beta^2\gamma + \gamma^2\alpha + \gamma^2\beta) \\
&+ 8\alpha\beta\gamma - \alpha^2 - \beta^2 - \gamma^2\big)n' + O(1) \qquad (4)
\end{aligned}$$

It can be verified (we leave out the details due to space limitations) that the maximum of (4) for $\alpha, \beta, \gamma \in [0,1]$ with $\alpha+\beta+\gamma = 1$ is attained for $\alpha = \beta = 1/2$ and $\gamma = 0$ (and the two other symmetric permutations of the values of the variables α, β and γ). Therefore, the expected number of Paul's questions does not exceed:

$$\left(2 + \frac{2\cdot 2 + 10\cdot 2 + 8\cdot 0}{8} - \frac{2}{4}\right)n' + O(1) = \frac{9}{2}n' + O(1) = \frac{3}{2}n + O(1)\ .$$

5 Lower Bound for the Plurality Problem

First, we state an auxiliary lemma (the proof is omitted due to space constraints):

Lemma 1. *Let n be an even positive integer, and let T be a rooted binary tree of depth at most $n-1$ with $\binom{n}{n/2}/2$ leaves. Assume that for each inner node w of T, the edge which leads to its left child is colored by red and the edge which leads to its right child by blue. The average number of blue edges on the paths from the root to the leaves in T is at least:*

$$\frac{n}{2} - O\left(\sqrt{n\log n}\right)\ .$$

We are now ready to prove the desired lower bound:

Theorem 2. *For any probabilistic strategy for the Plurality problem with n balls of three colors, where n is an even positive integer, there is a coloring of the balls such that Paul asks at least $\frac{3}{2}n - O\left(\sqrt{n\log n}\right)$ questions on average.*

Proof. By Yao's principle (Proposition 1), it is enough to find a probability distribution on colorings of the balls such that if Paul uses any deterministic strategy, then he asks at least $\frac{3}{2}n - O\left(\sqrt{n \log n}\right)$ questions on average. The desired distribution is the uniform distribution on colorings in which half of balls have the first color and the other half have the second color. There are no balls of the third color. Clearly, there is no plurality color for any of these colorings. Let $\mathcal{I}$ be the set of all $\binom{n}{n/2}$ such colorings.

Fix a deterministic strategy. Let G be Paul's final graph for one of the colorings from $\mathcal{I}$. We claim that each of the two subgraphs of G induced by the balls of the same color is connected. Otherwise, let V_1 and V_2 be the sets of the vertices corresponding to the balls of the first and the second color, respectively, and assume that $G[V_1]$ contains a component with a vertex set $W \subset V_1$. Based on G, Paul is unable to determine whether the balls of W have the same color as the balls of $V_1 \setminus W$, and thus he is unable to decide whether there is a plurality color (which would be the case if the balls of W had the third color). We conclude that Paul's final graph contains at least $n - 2$ red edges.

Let T be the rooted binary tree corresponding to Paul's strategy and let V_{lf} be the set of the leaves of T to which there is a valid computation path for a coloring of $\mathcal{I}$ (note that T contains additional leaves corresponding to colorings not contained in $\mathcal{I}$). Since for each of the colorings of $\mathcal{I}$ the two subgraphs induced by the balls of the same color in Paul's final graph are connected, each leaf from V_{lf} corresponds to exactly two such colorings (which differ just by a permutation of the two colors). Therefore, V_{lf} consists of $\binom{n}{n/2}/2$ leaves of T.

We modify T to a binary tree T'. T' is obtained from the union of all the paths in T from the root to the leaves of V_{lf} by contracting of paths formed by degree-two vertices. T' has $\binom{n}{n/2}/2$ leaves. The edges of T' are colored by red and blue according to whether they join the parent with its left or right child.

The tree T' corresponds to a deterministic strategy for distinguishing the colorings from the set $\mathcal{I}$. At each inner node w of T', the edge corresponding to Paul's question at the node w joins two different components of Paul's graph: otherwise, the answer is uniquely determined by his graph, and the node w has a single child (there are no colorings of $\mathcal{I}$ consistent with the other answer) and the edge leading from w to its child should have been contracted.

Since all Paul's questions correspond to edges between different components, Paul's final graph (for his strategy determined by T') is a forest for each coloring of $\mathcal{I}$. In particular, Paul's final graph contains at most $n - 1$ edges. Therefore, the depth of T' does not exceed $n - 1$. By Lemma 1, the average number of blue edges on the path from the root to a leaf of T' is at least $\frac{n}{2} - O\left(\sqrt{n \log n}\right)$. Since the number of blue edges on such a path is equal to the number of blue edges in Paul's final graph if Paul follows the strategy determined by T', the average number of blue edges in Paul's graphs is at least $\frac{n}{2} - O\left(\sqrt{n \log n}\right)$.

Observe that for each coloring of $\mathcal{I}$, the edges of the computation path in T' form a subset of the edges of the computation path in T. Therefore, the average number of blue edges in Paul's final graphs with respect to the strategy corresponding to T is also at least $\frac{n}{2} - O\left(\sqrt{n \log n}\right)$. Since each final graph

contains also (at least) $n-2$ red edges, the average number of Paul's questions, which is equal to the average number of edges in the final graph, is at least $\frac{3}{2}n - O\left(\sqrt{n \log n}\right)$.

6 Deterministic Strategy for the Partition Problem

We first describe Paul's strategy (note that the considered problem for the number k of colors larger than the number n of balls is the same as for $k = n$):

Proposition 2. *There is a deterministic strategy for the Partition problem with n balls of k colors such that Paul always asks at most $(k-1)n - \binom{k}{2}$ questions (for $n \geq k$).*

Proof. Paul's strategy is divided into n steps. In the i-th step, Paul determines the color of the i-th ball.

If the first $i-1$ balls have only $k' < k$ different colors, then Paul compares the i-th ball with the representatives of all the k' colors found so far. In this case, Paul finds either that the i-th ball has the same color as one of the first $i-1$ balls, or that its color is different from the colors of all of these balls.

If the first $i-1$ balls have k different colors, then Paul compares the i-th ball with the representatives of $k-1$ colors. If Paul does not find a ball with the same color, then the color of the i-th ball is the color with no representative.

In this way, Paul determines the colors of all the balls. We estimate the number of comparisons in the worst case. Since the first $i-1$ balls have at most $i-1$ different colors, the number of comparisons in the i-th step is at most $\min\{i-1, k-1\}$. Therefore, the number of questions does not exceed:

$$\sum_{i=1}^{k}(i-1) + \sum_{i=k+1}^{n}(k-1) = \frac{k(k-1)}{2} + (n-k)(k-1) = n(k-1) - \binom{k}{2}.$$

Next, we show that Carol can force Paul to ask $(k-1)n - \binom{k}{2}$ questions:

Theorem 3. *If Paul is allowed to use only a deterministic strategy to solve the Partition problem with n balls of k colors, then Carol can force him to ask at least $(k-1)n - \binom{k}{2}$ questions (for $n \geq k$).*

Proof. We can assume that Paul never asks a question whose answer is uniquely determined. Therefore, Carol can answer each question that the colors of the pair of the balls are different. Let G be Paul's graph at the end of the game. Note that all the edges of G are blue because of Carol's strategy.

Let $V_1, \ldots, V_k$ be the vertices of G corresponding to the sets of the balls of the same color. Each of the sets V_i, $1 \leq i \leq k$, is non-empty: otherwise, there exist an empty set V_i and a set $V_{i'}$ with at least two vertices (recall that $n \geq k$). Move a vertex from $V_{i'}$ to V_i. The new partition is also consistent with the graph G and therefore Paul is unable to uniquely determine the partition.

Assume now that the subgraph of G induced by $V_i \cup V_{i'}$ for some i and i', $1 \leq i < i' \leq k$, is disconnected. Let W be the vertices of one of the components of the subgraph. Move the vertices of $V_i \cap W$ from V_i to $V_{i'}$ and the vertices of $V_{i'} \cap W$ from $V_{i'}$ to V_i. Since the new partition is consistent with the graph G, Paul cannot uniquely determine the partition of the balls according to their colors. We may conclude that each set V_i is non-empty and the subgraph of G induced by any pair of V_i and $V_{i'}$ is connected.

Let n_i be the number of vertices of V_i. For every i and i', $1 \leq i < i' \leq k$, the subgraph of G induced by $V_i \cup V_{i'}$ contains at least $n_i + n_{i'} - 1$ edges because it is connected. Since the sets V_i are disjoint, the number of edges of G, which is the number of questions asked by Paul, is at least the following:

$$\sum_{1 \leq i < i' \leq k} (n_i + n_{i'} - 1) = \sum_{i=1}^{k} (k-1) n_i - \sum_{1 \leq i < i' \leq k} 1 = (k-1)n - \binom{k}{2} .$$

7 Probabilistic Strategy for the Partition Problem

We first state an auxiliary lemma whose proof is omitted due to space limitations:

Lemma 2. *Consider a random ordering of n balls, out of which, ξn balls are white and $(1-\xi)n$ are black. The expected length of the initial segment comprised entirely of white balls in the random ordering is at least:*

$$\min \left\{ 9, \frac{\xi}{1-\xi} - O\left(\frac{\log n}{\sqrt{n}} \right) \right\} .$$

We now describe Paul's strategy:

Theorem 4. *There is a probabilistic strategy for the Partition problem with n balls of three colors such that the expected number of Paul's questions does not exceed $\frac{5}{3}n - \frac{8}{3} + O\left(\frac{\log n}{\sqrt{n}}\right)$ for any coloring of the balls.*

Proof. Fix a coloring of the n balls. Let α, β and γ be fractions of the balls of each of the three colors in the coloring. Paul first chooses a random ordering of the balls. His strategy is divided into n steps, and in the i-th step Paul determines the color of the i-th ball (in the random ordering).

Paul compares the i-th ball with a representative of a randomly chosen color among the colors of the first $i-1$ balls. If Carol answers that the balls have different colors and the first $i-1$ balls have two or three distinct colors, then Paul also compares the i-th ball with a representative of another color (in case of three colors, again chosen in random).

If the input does not contain balls of all three colors, then the expected number of Paul's questions in each step is at most $3/2$. Consequently, the expected number of all the questions asked by Paul does not exceed $\frac{3}{2}n$, and the statement of the lemma readily follows. Therefore, we assume in the rest that there is a ball of each of the three colors, i.e., $\alpha, \beta, \gamma > 0$.

Fix an ordering of the balls. Let j be the largest integer such that the first j balls have the same color, and j' the largest integer such that the first j' balls have at most two different colors. We compute the expected number of Paul's questions for the fixed ordering. In the first $j+1$ steps (except for the first step), Paul always asks a single question. At each of the next $j'-j-1$ steps, Paul asks $3/2$ questions on average. In the $(j'+1)$-th step, Paul asks two questions. At each of the $n-j'-1$ remaining steps, Paul asks $5/3$ questions on average. Hence, the expected number of Paul's questions for the fixed ordering is:

$$j + \frac{3}{2}(j' - j - 1) + 2 + \frac{5}{3}(n - j' - 1) = \frac{5}{3}n - \frac{1}{2}j - \frac{1}{6}j' - \frac{7}{6} .$$

The expected number of the questions (averaged through all the orderings) is:

$$\frac{5}{3}n - \frac{1}{2}\overline{j} - \frac{1}{6}\overline{j'} - \frac{7}{6} \tag{5}$$

where $\overline{j}$ and $\overline{j'}$ are the expected lengths of the initial segments in the random ordering comprised by balls of one color and two colors, respectively.

Let $\overline{j_A}$, $\overline{j_B}$ and $\overline{j_C}$ be the expected lengths of initial segments in the ordering formed entirely by balls of the color with the fractions α, β and γ, respectively. Similarly, $\overline{j_{AB}}$, $\overline{j_{AC}}$ and $\overline{j_{BC}}$ are the expected lengths of initial segments formed by balls of two indexed colors. Clearly, the following holds:

$$\begin{aligned}\overline{j} &= \overline{j_A} + \overline{j_B} + \overline{j_C} \\ \overline{j'} &= \overline{j_{AB}} + \overline{j_{AC}} + \overline{j_{BC}} - \overline{j_A} - \overline{j_B} - \overline{j_C}\end{aligned}$$

If one of the numbers $\overline{j_A}$, $\overline{j_B}$, $\overline{j_C}$, $\overline{j_{AB}}$, $\overline{j_{AC}}$ and $\overline{j_{BC}}$ is at least 9, then the sum $\overline{j} + \overline{j'}$ is at least 9, and the expected number of Paul's questions is at least $\frac{5}{3}n - \frac{9}{6} - \frac{7}{6} = \frac{5}{3}n - \frac{8}{3}$. If none of the numbers exceed 9, we have by Lemma 2:

$$\begin{aligned}\overline{j} &= \tfrac{\alpha}{1-\alpha} + \tfrac{\beta}{1-\beta} + \tfrac{\gamma}{1-\gamma} + O\left(\tfrac{\log n}{\sqrt{n}}\right) \quad \text{and} \\ \overline{j} + \overline{j'} &= \tfrac{\alpha+\beta}{1-\alpha-\beta} + \tfrac{\alpha+\gamma}{1-\alpha-\gamma} + \tfrac{\beta+\gamma}{1-\beta-\gamma} + O\left(\tfrac{\log n}{\sqrt{n}}\right) .\end{aligned}$$

Since the function $\frac{\xi}{1-\xi}$ is convex and $\alpha + \beta + \gamma = 1$, the minimum of both the expressions on the right hand side in the above equations is attained for $\alpha = \beta = \gamma = 1/3$. Therefore, we have the following:

$$\overline{j} \geq 3 \cdot \frac{1/3}{1 - 1/3} - O\left(\frac{\log n}{\sqrt{n}}\right) = \frac{3}{2} - O\left(\frac{\log n}{\sqrt{n}}\right) \tag{6}$$

$$\overline{j} + \overline{j'} \geq 3 \cdot \frac{2/3}{1 - 2/3} - O\left(\frac{\log n}{\sqrt{n}}\right) = 6 - O\left(\frac{\log n}{\sqrt{n}}\right) \tag{7}$$

Plug (6) and (7) to the expression (5) for the expected number of questions:

$$\frac{5}{3}n - \frac{1}{3}\overline{j} - \frac{1}{6}\left(\overline{j} + \overline{j'}\right) - \frac{7}{6} \leq \frac{5}{3}n - \frac{1}{2} - 1 - \frac{7}{6} + O\left(\frac{\log n}{\sqrt{n}}\right) = \frac{5}{3}n - \frac{8}{3} + O\left(\frac{\log n}{\sqrt{n}}\right) .$$

A proof of the next lemma is due to space limitations:

Lemma 3. *Let T be a rooted binary tree with N leaves. Assume that on each path from the root of T to a leaf, at least ℓ inner nodes have the following property ($\star$): the number of the leaves in the left subtree is precisely half of the number of the leaves in the right subtree. The average length of the path from the root to a leaf of T is at least the following:*

$$\log_2 N + \left(\frac{5}{3} - \log_2 3\right)\ell$$

We now show that the number of Paul's questions in Theorem 4 is optimal:

Theorem 5. *For any probabilistic strategy for the Partition problem with n balls of three colors, there is a coloring of the balls which forces Paul to ask at least $\frac{5}{3}n - \frac{8}{3}$ questions on average.*

Proof. By Yao's principle (Proposition 1), it is enough to find a probability distribution on colorings such that if Paul uses any deterministic strategy, then he asks at least $\frac{5}{3}n - \frac{8}{3}$ questions on average. We claim that the uniform distribution on all 3^n possible colorings has this property.

Fix a deterministic strategy and let T be the corresponding binary tree. Since Paul is able to solve the Partition problem using this strategy, the computation paths may end in the same leaf for at most six different colorings (they can differ only by a permutation of the colors). Hence, T has at least $N = 3^n/6$ leaves.

Consider an inner node w of T at which the question corresponds to an edge e between two different components of Paul's graph. Let $\mathcal{I}$ be the set of colorings whose computation path reaches w. Since e joins two different components, for exactly one third of the input colorings Carol answers that the balls have the same colors, and for exactly two thirds she answers that their colors are different: for each coloring $X \in \mathcal{I}$, the set $\mathcal{I}$ contains all the five other colorings obtained from X by permutation of the colors in one of the components. Hence, if the question at w corresponds to an edge between two different components of Paul's graph, then w has the property ($\star$) from the statement of Lemma 3.

Observe that Paul's final graph is connected: otherwise, permute the colors of the balls in one of the components and keep the colors of the remaining balls. This yields a different partition consistent with Paul's final graph, and Paul is unable to uniquely determine the partition. Since Paul's final graph is connected, on each path from the root to a leaf, there are at least $n-1$ nodes in which Paul asks a question corresponding to an edge between two components. Therefore, on each such path, at least $n-1$ nodes have the property ($\star$) from Lemma 3.

By Lemma 3, the average length of the path from the root to a leaf of T, which is equal to the expected number of Paul's questions, is at least:

$$\begin{aligned}\log_2 \frac{3^n}{6} + \left(\frac{5}{3} - \log_2 3\right)(n-1) &= \log_2 3 \cdot n - \log_2 3 - 1 + \left(\frac{5}{3} - \log_2 3\right)(n-1) \\ &= \frac{5}{3}n - 1 - \frac{5}{3} = \frac{5}{3}n - \frac{8}{3}\,.\end{aligned}$$

The arguments of this section generalize to colorings with more colors, but the corresponding lower and upper bounds do not match: a probabilistic strategy similar to that in Proposition 4 requires $\frac{k^2+k-2}{2k}n + O(1)$ questions on average for the Partition problem with n balls of k colors. On the other hand, the tree of any deterministic strategy must contain at least $n-1$ nodes w on any path from the root to a leaf such that the subtree of the left child of w contains exactly the fraction of $1/k$ of the leaves of the whole subtree of w. This yields a lower bound of $\left(\frac{k-1}{k}\log_2(k-1)+1\right)n - \Theta(k\log k)$ on the expected number of questions.

Acknowledgements

The paper is based on the results obtained during the DIMACS/DIMATIA Research Experience for Undergraduates program in 2004. The REU program is a joint project of DIMACS at Rutgers University and DIMATIA at Charles University. The authors are also grateful to János Komlós, the director of the REU program at Rutgers, for his helpful comments on the considered problems.

References

1. Aigner M.: Variants of the majority problem, Discrete Applied Mathematics **137** (2004) 3–25.
2. Aigner M., De Marco G., Montangero M.: The plurality problem with three colors, in: Diekert V., Habib M. (eds.): STACS 2004, LNCS 2296, Springer-Verlag, Berlin, Heidelberg, 2004, 513–521.
3. Aigner M., De Marco G., Montangero M.: The plurality problem with three colors and more, submitted.
4. Alonso L., Reingold E. M., Schott R.: Determining the majority, Information Processing Letters **47** (1993) 253–255.
5. Cover T. M., Thomas J. A.: Elements of information theory, John Wiley & Sons, 1991.
6. Fisher M., Salzberg S.: Finding a majority among n votes, Journal of Algorithms **3** (1982) 375–379.
7. Král' D., Sgall J., Tichý T.: Randomized strategies for the plurality problem, in preparation.
8. De Marco G., Pelc A.: Randomized algorithms for determining the majority on graphs, in: Rovan B., Vojtás P. (eds.): MFCS 2003, LNCS 2747, Springer-Verlag, Berlin, Heidelberg, 2003, 368–377.
9. Matoušek J., Nešetřil J.: Invitation to discrete mathematics, Oxford University Press, 1998.
10. Motwani R., Raghavan P.: Randomized Algorithms, Cambridge University Press, 1995.
11. Saks M., Werman M.: On computing majority by comparisons, Combinatorica **11** (1991) 383–387.
12. Srivastava N.: Tight bounds on plurality using equality tests, unpublished manuscript, 2003.
13. Yao A. C.-C.: Probabilistic computations: Towards a unified measure of complexity, in: Proc. of the 17th Annual Symposium on Foundations of Computer Science (FOCS), 1977, 222–227.

Cost Sharing and Strategyproof Mechanisms for Set Cover Games

Xiang-Yang Li[1,⋆], Zheng Sun[2,⋆⋆], and Weizhao Wang[1]

[1] Illinois Institute of Technology, Chicago, IL, USA
{lixian,wangwei4}@iit.edu
[2] Hong Kong Baptist University, Hong Kong, China
sunz@comp.hkbu.edu.hk

Abstract. We develop for set cover games several general cost-sharing methods that are approximately budget-balanced, core, and/or group-strategyproof. We first study the cost sharing for a single set cover game, which does not have a budget-balanced core. We show that there is no cost allocation method that can always recover more than $\frac{1}{\ln n}$ of the total cost if we require the cost sharing being a core. Here n is the number of all players to be served. We give an efficient cost allocation method that always recovers $\frac{1}{\ln d_{max}}$ of the total cost, where d_{max} is the maximum size of all sets. We then study the cost allocation scheme for all induced subgames. It is known that no cost sharing scheme can always recover more than $\frac{1}{n}$ of the total cost for every subset of players. We give an efficient cost sharing scheme that always recovers at least $\frac{1}{2n}$ of the total cost for every subset of players and furthermore, our scheme is cross-monotone. When the elements to be covered are selfish agents with privately known valuations, we present a strategyproof charging mechanism, under the assumption that all sets are simple sets, such that each element maximizes its profit when it reports its valuation truthfully; further, the total cost of the set cover is no more than $\ln d_{max}$ times that of an optimal solution. When the sets are selfish agents with privately known costs, we present a strategyproof payment mechanism in which each set maximizes its profit when it reports its cost truthfully. We also show how to *fairly* share the payments to all sets among the elements.

1 Introduction

Generalized Set Cover Problem. Let $U = \{e_1, e_2, \cdots, e_n\}$ be a finite set, and let $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$ be a collection of multisets (or *sets* for short) of U. For each $e_i \in U$ and each $S_j \in \mathcal{S}$, we denote the multiplicity of e_i in S_j by $k_{j,i}$. Each S_j is associated with a cost c_j. For any $\mathcal{X} \subseteq \mathcal{S}$, let $C(\mathcal{X})$ denote the

⋆ The research of the author was supported in part by NSF under Grant CCR-0311174. The research was conducted when the author was visiting Hong Kong Baptist University.

⋆⋆ The research of the author was supported in part by Grant FRG/03-04/II-21 and Grant RGC HKBU2107/04E.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 218–230, 2005.

total costs of the sets in $\mathcal{X}$, *i.e.*, $C(\mathcal{X}) = \sum_{S_j \in \mathcal{X}} c_j$. For a given $k > 0$ and a set of *element coverage requirements* $\{r_1, r_2, \cdots, r_n\}$, a *k-partial-cover* $\mathcal{C}$ is defined to be a subset $\{S_{j_1}, S_{j_2}, \cdots, S_{j_l}\}$ of $\mathcal{S}$ such that $\sum_{i=1}^{n} \min\{r_i, \sum_{t=1}^{l} k_{j_t,i}\} \geq k$. The *generalized set cover problem* is to compute an optimum k-partial-cover $\mathcal{C}_{opt}$ with the minimum cost $C(\mathcal{C}_{opt})$.

This problem becomes the traditional multicover problem [1, 2] when we set $k = \sum_{i=1}^{n} r_i$ and $k_{j,i} = 1$ for all S_j and e_i, as each element e_i should be *fully* covered and each set S_j is a simple set. When we set $r_i = 1$, it becomes the traditional partial cover problem [3]. This problem is therefore a natural extension of the classic set cover problem by allowing partial cover, multiset, and element coverage requirement greater than 1. Accordingly, the greedy algorithm for this problem is a combination of the algorithms designed for partial cover and multicover [1, 2, 3].

Set Cover Game. Consider the following scenario: a company can choose from a set of service providers $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$ to provide services to a set of service receivers $U = \{e_1, e_2, \cdots, e_n\}$.

- With a fixed cost c_j, each service provider S_j can provide services to a fixed subset of service receivers.
- There may be a limit $k_{j,i}$ on the number of units of service that a service provider S_j can provide to a service receiver e_i. For example, each service provider may be a cargo company that is transporting goods to various cities (the service receivers), and the amount of goods that can be transported to a particular city daily is limited by the number of trains/trucks that are going to that city everyday.
- Each service receiver e_i may have a limit r_i on the number of units of service that it desires to receive (and is willing to pay for).
- There may be a limit k on the total number of units of service that the service providers shall provide to the service receivers.

The problem can be modeled by the generalized set cover problem defined previously. There may be different types of games according to various conditions:

1. Each service receiver e_i has to receive at least r_i units of service, and the costs incurred by the service providers will be shared by the service receivers.
2. Each service receiver e_i declares a bid $b_{i,r}$ for the r-th unit of service it shall receive, and is willing to pay for it only if the assigned cost is at most $b_{i,r}$.
3. Each service provider S_j declares a cost c_j, and is willing to provide the service only if the payment received is at least c_j.

There are different algorithmic issues for these games. For example, for Game 1, we shall define a cost allocation method so that every subset of service receivers feel that the total cost they need to pay is "fair" according to certain criteria. For Games 1 and 2, the cost allocation method, by charging service receivers, needs to recover (either entirely or a constant fraction of) the total cost of the chosen service providers. For Games 2 and 3, we need a mechanism (for determining costs charged to service receivers and payments paid to service providers) that can guarantee that the players are truthful with their declaration of bids/costs.

Our Results. We first study how we share the cost of the selected service providers among the service receivers such that some fairness criteria are met. We present a cost sharing method that is $\frac{1}{\ln d_{max}}$-budget-balanced and core, where d_{max} is the maximum set size. The bound $\frac{1}{\ln d_{max}}$ is tight. We also present a cost sharing method that is $\frac{1}{2n}$-budget-balanced core and cross-monotone, which is almost the optimum [4].

We then design greedy set cover methods that are cognizant of the fact that the service providers or the service receivers are selfish and rational. By "selfish," we mean that they only care about their own benefits without consideration for the global performances or fairness issues. By "rational," we mean that when the methods of computing the output for the set cover game are instituted, they will always choose their actions to maximize their benefits. When the elements to be covered are selfish agents with privately known valuations, we present a strategyproof charging mechanism, under the assumption that all sets are simple sets, such that each element maximizes its profit when it reports its valuation truthfully; further, the total cost of the set cover is no more than $\ln d_{max}$ times that of an optimal solution for these selected service receivers and their coverage requirements. When the sets are selfish agents with privately known costs, we present a strategyproof payment mechanism in which each set maximizes its profit when it reports its cost truthfully. We also show how to *fairly* share the payments to all sets among the elements.

Paper Organization. In Section 2, we give the exact definitions of fair cost sharing and mechanism design. In Section 3, we study how to fairly share the cost of the service providers among the covered service receivers when the receivers must receive the service. We show in Section 4 how to charge the cost of service providers to the selfish service receivers when each receiver has a valuation on the r-th cover received. We then show in Section 5 a strategyproof payment scheme to the selfish service providers when each has a privately known cost. We conclude our paper in Section 6.

2 Preliminaries and Prior Art

2.1 Preliminaries

Algorithm Mechanism Design (AMD). Assume that there are n agents. Each agent i, for $i \in \{1, \cdots, n\}$, has some *private* information t_i, called its *type*. All agents' types define a type vector $t = (t_1, t_2, \cdots, t_n)$. A mechanism defines, for each agent i, a set of strategies A_i. For each strategy vector $a = (a_1, \cdots, a_n)$, *i.e.*, agent i plays a strategy $a_i \in A_i$, the mechanism computes an *output* $o = \mathcal{O}(a)$ and a *payment* vector $\mathcal{P}(a) = (p_1, \cdots, p_n)$, where $p_i = \mathcal{P}_i(a)$ is the amount of money given to the participating agent i. Let $v_i(t_i, o)$ denote agent i's preferences to an output o and $u_i(t_i, o(a), p_i(a))$ denote its *utility* at the outcome (o, p) of the game. We assume that agents are *rational* and have quasi-linear utility functions. The utility function is *quasi-linear* if $u_i(t_i, o) = v_i(t_i, o) + p_i$. An agent is called *rational* if it always adopts its best strategy (called *dominant strategy*)

that maximizes its utility regardless of what other agents do. A direct-revelation mechanism is *incentive compatible* (IC) if reporting valuation truthfully is a dominant strategy. Another common requirement in the literature for mechanism design is the so called *individual rationality* (IR): the agent's utility of participating in the output of the mechanism is not less than the utility of the agent if it did not participate at all. A mechanism is called *truthful* or *strategyproof* if it satisfies both IC and IR properties. To make the mechanism tractable, both methods $\mathcal{O}()$ and $\mathcal{P}()$ should be computable in polynomial time. A mechanism $M = (\mathcal{O}, \mathcal{P})$ is β-efficient if $\forall t$, $\sum_{i=1}^{n} v_i(t_i, \mathcal{O}(t)) \geq \beta \cdot \max_o \sum_{i=1}^{n} v_i(t_i, o)$. Obviously for the set cover game, we cannot design an $o(\ln n)$-efficient polynomial-time computable strategyproof mechanism unless $NP \subset DTIME(n^{\log \log n})$ [2].

Cost Sharing. Consider a set U of n players. For a subset $S \subseteq U$ of players, let $C(S)$ be the cost of providing service to S. Here $C(S)$ could be the minimum cost, denoted by $\mathrm{OPT}(S)$, or the cost computed by some algorithm $\mathcal{A}$, denoted by $\mathcal{A}(S)$. We always assume that the cost function $C(S)$ is *cohesive*, *i.e.*, for any two disjoint subsets S_1 and S_2, $C(S_1 \cup S_2) \leq C(S_1) + C(S_2)$. A cost sharing scheme is simply a function $\xi(i, S)$ with $\xi(i, S) = 0$ for $i \notin S$, for every set $S \subseteq U$ of players. An obvious criterion is that the sharing method should be *fair*. While the definition of budget-balance is straightforward, defining fairness is more subtle: many fairness concepts were proposed in the literature, such as *max-min* [5], *min-max* [6], *core* and *bargaining set* [7]. Typically, the following three properties are required by a cost sharing scheme.

1. (α**-budget-balance**) For a given parameter $\alpha \leq 1$, $\alpha \cdot C(U) \leq \sum_{i \in U} \xi(i, U) \leq C(U)$. If $\alpha = 1$, we call the cost sharing scheme *budget-balanced*.
2. (**fairness under core**) For any subset $S \subseteq U$, $\sum_{i \in S} \xi(i, U) \leq \mathrm{OPT}(S)$.
3. (**Cross-monotonicity**) For any two subsets $S \subseteq T$ and $i \in S$, $\xi(i, S) \geq \xi(i, T)$.

When only the first two conditions are satisfied, we call the cost sharing scheme to be in the α-*core*. When all three conditions are met, we call the cost sharing scheme to be cross-monotone α-*core*. When each player i has a valuation v_i on getting the service, a mechanism should first decide the output of the game (who will get the service), and then decide what is the share of each selected player (what is the payment method). It is well-known that a cross-monotone cost sharing scheme implies a *group-strategyproof* mechanism [8]. Notice that the cross-monotone property is not the necessary condition for group-strategyproof. Naturally, several additional properties are required for a cost sharing scheme when every player has a valuation on getting the service.

1. (**Incentive Compatibility**) Assume that the valuation by player i on getting the service is v_i. Let $b = (b_1, b_2, \cdots, b_n)$ be the bidding vector of n players. Let $\mathcal{O}(b) = (o_1, o_2, \cdots, o_n)$ denote whether a player is selected to get the service or not and $\mathcal{P}(b)$ be the charge to player i, *i.e.*, the mechanism is $M = (\mathcal{O}(b), \mathcal{P}(b))$. It satisfies IC if every player maximizes its profit $v_i \cdot o_i - p_i$ when $b_i = v_i$.
2. (**No Positive Transfer**) For every player i, $p_i \geq 0$.

3. (**Individual Rationality**) For every player i, $v_i \cdot o_i - p_i \geq 0$.
4. (**Consumer Sovereignty**) Fix the bids of all other players, there exists a value τ_i such that player i is guaranteed to get the service when its bid is larger than τ_i.

2.2 Prior Arts on Cost Sharing and Algorithm Mechanism Design

Although the traditional set cover problem (without multisets and partial-cover requirement) can be viewed as a special case of multicast, several results were proposed specifically for set cover in selfish environment. Devanur *et al.* [9] studied, for the set cover game and facility location game, how the cost of shared resource is to be distributed among its users in such a way that revealing the true valuation is a dominant strategy for each user. Their cost sharing method is not in the *core* of the game. One of the open questions left in [9] is to design a strategyproof cost sharing method for multicover game in which the bidders might want to get covered multiple times. For facility location game, Pál and Tardos [10] gave a cost sharing method that can recover $\frac{1}{3}$ of the total cost, and recently, Immorlica *et al.* [4] showed that this is the best achievable upper bound for any cross-monotonic cost sharing method. Sharing the *cost* of the multicast structure among receivers was studied in [8, 11, 12, 13, 14, 15, 16] so some fairness is accomplished.

3 Cost Sharing Among Unselfish Service Receivers

In this section, we study how to share the cost of the service providers among a given set of service receivers. For this scenario, it is difficult to find realistic examples where a partial cover is desired. Therefore, in the remainder of this section, we only consider the multiset multicover problem, *i.e.*, $k = \sum_{i=1}^{n} r_i$. However, the results presented here can easily be generalized to the partial cover case, should such a scenario arise.

3.1 α-Core

Given a subset of elements X, let $\mathrm{OPT}(X)$ denote the cost of an optimum cover $\mathcal{C}_{opt}(X)$ of X. This cost function clearly is *cohesive*: for every partition $T_1, T_2, \cdots, T_t$ of U, $\mathrm{OPT}(U) \leq \sum_{i=1}^{t} \mathrm{OPT}(T_i)$. A *cost allocation* for U is a n-dimensional vector $x = (x_1, x_2, \cdots, x_n)$ that specifies for each element $e_i \in U$ the share $x_i \geq 0$ of the total cost of serving U that e_i shall pay. Ideally, when the set of elements to be covered is fixed to be U, we want the cost allocation x to be budget-balanced and fair, *i.e.*, being in core. However, a simple example in [17] shows that there is no budget-balanced core for the set-cover game. We then relax the notion of budget-balance to the notion of α-budget-balance for some $\alpha \leq 1$. See [17] for the proof of the achievable α-core.

Theorem 1. *For the set cover game,* no *cost allocation method is α-core for $\alpha > \frac{1}{\ln n}$.*

We then give a cost allocation method that can recover $\frac{1}{\ln d_{max}}$ of the total cost OPT(U) for a multiset multicover game, where $d_{max} = \max_{1 \le j \le m} |S_j|$. Without loss of generality, we assume that $d_{max} \le \sum_{i=1}^{n} r_i$. The basic approach of our cost allocation method is as follows. We first run the greedy Algorithm 1 to find a set cover $\mathcal{C}_{grd}$ with an approximation ratio of $\ln d_{max}$. Starting with $\mathcal{C}_{grd} = \emptyset$, the greedy algorithm adds to $\mathcal{C}_{grd}$ a set $S_{j_{t'}}$ at each round t'. After the s-th round, we define the *remaining required coverage* r'_i of an element e_i to be $r_i - \sum_{t'=1}^{s} k_{j_{t'},i}$. For any $S_j \notin \mathcal{C}_{grd}$, the *effective coverage* $k'_{j,i}$ of e_i by S_j is defined to be $\min\{k_{j,i}, r'_i\}$, the *value* v_j of S_j is defined to be $\sum_{i=1}^{n} k'_{j,i}$, and the *effective average cost* of S_j is defined to be $\frac{c_j}{v_j}$.

Algorithm 1 Greedy algorithm for multiset multicover problem.

1: $\mathcal{C}_{grd} \leftarrow \emptyset$; $r'_i \leftarrow r_i$ for each e_i.
2: **while** $U \neq \emptyset$ **do**
3: pick the set $S_{t'}$ in $\mathcal{S} \setminus \mathcal{C}_{grd}$ with the minimum effective average cost.
4: $\mathcal{C}_{grd} \leftarrow \mathcal{C}_{grd} \cup \{S_{t'}\}$.
5: **for all** $e_i \in U$ **do**
6: $r'_i \leftarrow \max\{0, r'_i - k_{t',i}\}$.
7: **if** $r'_i = 0$ **then** $U \leftarrow U \setminus \{e_i\}$.

The greedy algorithm will select a set S_j with the least effective average cost. For any e_i and r such that $r_i - r'_i + 1 \le r \le r_i - r'_i + k'_{j,i}$, we let $\text{price}(i, r) = \frac{c_j}{v_j}$. Let $x'_i = \sum_{r=1}^{r_i} \text{price}(i, r)$ and $x_i = \frac{x'_i}{\ln d_{max}}$. We claim the following theorem (see [17]):

Theorem 2. *The above-defined cost allocation x is a $\frac{1}{\ln d_{max}}$-core.*

Recall that the core we defined, given a set of players U, required that $\sum_{e_i \in T} \xi(i, U)$ is at most the *optimum* cost of providing service to elements in T. For a set cover game, clearly it is NP-hard to find the optimum cost of covering T. Naturally, one may relax the α-core as follows: a cost sharing method $\xi(i, \cdot)$ is called a *relaxed* α-core if (1) $\alpha \cdot \mathcal{C}_{grd}(U) \le \sum_{i \in U} \xi(i, U) \le \mathcal{C}_{grd}(U)$; and (2) $\sum_{i \in T} \xi(i, U) \le \mathcal{C}_{grd}(T)$ for every subset $T \subseteq U$. Even we relax the definition of the core to this, we can still prove in [17] that with the cost function computed by the greedy algorithm, there is no cost sharing method that is a relaxed α-core for $\alpha = \Omega(\frac{1}{\ln n})$.

3.2 Cross-Monotone α-Core

Clearly, if a cost sharing scheme is cross-monotone α-core then every cost allocation method $\xi(\cdot, S)$ induced on a subset S of players is always α-core, but the reverse is not true. From Theorem 1, clearly *no* cost sharing scheme for the set cover game is cross-monotone α-core for $\alpha = \frac{1}{\ln n}$. Recently, it was claimed in [4] that for set cover game, there is *no* cross-monotone α-core cost sharing scheme for $\alpha = \frac{1}{n} + \epsilon$.

For generalized set cover games, we will present a cross-monotone cost sharing scheme $\xi(i,S)$ (see Algorithm 2) that can recover $\frac{1}{2n}$ of the total cost. We show an example in [17] that the bound $\frac{1}{2n}$ is tight for Algorithm 2. Further, the bound is tight, for set cover games without multisets (but still allowing multicover requirements): our cross-monotone cost sharing scheme $\xi(i,S)$ can recover $\frac{1}{n}$ of the total cost.

Algorithm 2 Cost sharing for multiset multicover game with elements T

1: Set $\mathcal{C}_{\mathcal{A}} \leftarrow \emptyset$, $Y(i,j)=0$ and $\zeta(i,j)=0$ for $1 \le i \le n$ and $1 \le j \le m$. Here $Y(i,j)$ denotes how many cover requirements of element e_i are provided by set S_j, and $\zeta(i,j)$ denotes the fraction cost of set S_j shared by the element e_i.
2: **for all** element $e_i \in T$ **do**
3: Set $r'_i \leftarrow r_i$;
4: **while** $r'_i > 0$ **do**
5: Find the set S_t with the minimum ratio $\min_{S_j \in \mathcal{S}-\mathcal{C}_{\mathcal{A}}} \frac{c_j}{\min(k_{j,i}, r'_i)}$;
6: $Y(i,t) \leftarrow \min(k_{j,i}, r'_i)$; $r'_i \leftarrow r'_i - Y(i,t)$; and $\mathcal{C}_{\mathcal{A}} \leftarrow \mathcal{C}_{\mathcal{A}} \cup \{S_t\}$.
7: **for all** set S_j **do**
8: **if** $\sum_{1 \le i \le n} Y(i,j) > 0$ **then** $\rho_j \leftarrow \frac{c_j}{\sum_{1 \le i \le n} Y(i,j)}$;
9: **for all** element $e_i \in T$ **do**
10: Set $\zeta(i,j) = Y(i,j) \cdot \rho_j$.
11: **for all** element $e_i \in T$ **do**
12: Set $\xi'(i,T) = \sum_{1 \le j \le m} \zeta(i,j)$ and $\xi(i,T) = \frac{\sum_{1 \le j \le m} \zeta(i,j)}{2|T|}$.

Theorem 3. *The cost sharing scheme $\xi(\cdot,\cdot)$ is a cross-monotone $\frac{1}{2n}$-core and is cross-monotone $\frac{1}{n}$-core for set cover game when every set S_j is a simple set.*

4 Cost Sharing Among Selfish Service Receivers

In Section 3 we assumed that all elements (service receivers) are unselfish and all their coverage requirements are to be satisfied. In this section, we consider the problem of selecting service providers under the constraint of a collection of bids $B = B_1 \cup B_2 \cup \cdots \cup B_n$. Each B_i contains a series of bids $b_{i,1}, b_{i,2}, \cdots, b_{i,r_i}$, where $b_{i,r}$ denotes the declared price that element e_i is willing to pay for the r-th coverage (*i.e.*, the valuation of the r-th coverage). In this scenario, we may also consider partial cover, as the total number of units of service available may be limited by a constant k.

We assume that $b_{i,1} \ge b_{i,2} \ge \cdots \ge b_{i,r_i}$. This is often true in realistic situations: the marginal valuations are usually decreasing. A bid $b_{i,r}$ will be served (and the subsequent bid $b_{i,r+1}$ will be considered) only if $b_{i,r} \ge \text{price}(i,r)$, where $\text{price}(i,r)$ is the cost to be paid by e_i for its r-th coverage. Further, to guarantee that the mechanism is both strategyproof and budget-balanced, we assume that each set is a simple set.

We use a greedy algorithm (see Algorithm 3) similar to the one for the traditional set cover game [9]. Informally speaking, we start with $y = 0$, where y is the cost to be shared by each bid served. We raise y until there exists a set S_j whose cost can be sufficiently covered by the element copies in S_j, if each element copy needs to pay y. To adapt to the multicover scenario, we make the following changes:

- For any set $S_j \notin \mathcal{C}_{grd}$ and any e_i, we define the *collection of alive bids* $B_i^{(j)}$ of e_i with respect to S_j to be $\{b_{i,r_i-r'_i+1}\}$ if $k'_{j,i} > 0$ (*i.e.*, $k'_{j,i} = 1$ since S_j is a simple set) and $b_{i,r_i-r'_i+1} \geq y$, or $\emptyset$ if otherwise. That is, if y is the cost to be paid for each bid served, $B_i^{(j)}$ contains the bid of e_i covered by S_j that can afford the cost (if any).
- Define the value v_j of S_j as $\sum_{i=1}^{n} |B_i^{(j)}|$, and its effective average cost as $\frac{c_j}{v_j}$.

Algorithm 3 Cost sharing for multicover game with selfish receivers.

$\mathcal{C}_{grd}(B) \leftarrow \emptyset$; $A \leftarrow \emptyset$; $y \leftarrow 0$; $k' \leftarrow k$; $B' = \emptyset$;
while $A \neq U$ and $k' > 0$ **do**
  Raise y until one of the two events happens:
  if $B_i^{(j)} = \emptyset$ for all S_j **then** $U \leftarrow U \setminus \{e_i\}$;
  if $c_j \leq v_j \cdot y$ for some set S_j **then**
    $\mathcal{C}_{grd}(B) \leftarrow \mathcal{C}_{grd}(B) \bigcup \{S_j\}$; $k' \leftarrow k' - v_j$;
    for all element e_i with $B_i^{(j)} \neq \emptyset$ **do**
      price$(i, r_i - r'_i + 1) \leftarrow \frac{c_j}{v_j}$; $B' \leftarrow B' \cup \{b_{i,r_i-r'_i+1}\}$;
      $r'_i \leftarrow r'_i - 1$;
      if $r'_i = 0$ **then** $A \leftarrow A \bigcup \{e_i\}$;
    update all $B_i^{(j')}$ for all $S_{j'} \notin \mathcal{C}_{grd}$ and $e_i \in S_j \bigcap S_{j'}$;

When the algorithm terminates, B' contains all bids (of all elements) that are served. We prove the following theorem about this mechanism (see [17] for proof):

Theorem 4. *Algorithm 3 defines a strategyproof mechanism. Further, the total cost of the sets selected is no more than* $\ln d_{max}$ *times that of an optimal solution.*

In [9] multicover game was also considered. However, the algorithm used is different from ours and also they did not assume that the bids by the same element are non-increasing, and their mechanism is not strategyproof.

5 Selfish Service Providers

The underline assumption made so far in previous sections is that the service providers are truthful in revealing their costs of providing the service. In this section, we will address the scenario when service providers are selfish in revealing their costs.

5.1 Strategyproof Mechanism

We want to find a subset of agents D such that $\bigcup_{j\in D} S_j$ has r_i copies of element e_i for every element $e_i \in U$. Let $c = (c_1, c_2, \cdots, c_m)$. The social efficiency of the output D is $-\sum_{j\in D} c_j$, which is the objective function to be maximized. Clearly a VCG mechanism [18, 19, 20] can be applied if we can find the subset of $\mathcal{S}$ that satisfies the multicover requirement of elements in U with the minimum cost. Unfortunately this is NP-hard. Let $\mathcal{C}_{grd}(\mathcal{S}, c, U, r)$ be the sets selected from $\mathcal{S}$ (with cost specified by a cost vector $c = (c_1, \cdots, c_m)$) by the greedy algorithm to cover elements in U with cover requirement specified by a vector $r = (r_1, \cdots, r_n)$ (see Algorithm 1). We assume that the type of an agent is (S_j, c_j), *i.e.*, every service provider j could lie not only about its cost c_j but also about the elements it could cover. This problem now looks very similar to the combinatorial auction with single minded bidder studied in [21]. We show in [17] that the mechanism $M = (\mathcal{C}_{grd}, \mathcal{P}^{VCG})$ is not truthful, *i.e.*, use Algorithm 1 to find a set cover, and apply VCG mechanism to compute the payment to the selected agents: the payment to an agent j is 0 if $S_j \notin \mathcal{C}_{grd}$; otherwise, the payment to a set $S_j \in \mathcal{C}_{grd}$ is $\mathcal{P}_j^{VCG} = C(\mathcal{C}_{grd}(\mathcal{S} \setminus \{S_j\}, c|^j\infty, U, r)) - C(\mathcal{C}_{grd}(\mathcal{S}, c, U, r)) + c_j$. Here $C(\mathcal{X})$ is the total cost of the sets in $\mathcal{X} \subseteq \mathcal{S}$.

For the moment, we assume that agent j won't be able to lie about its element S_j. We will drop this assumption later. Clearly, the greedy set cover method presented in Algorithm 1 satisfies a monotone property: if a set S_j is selected with a cost c_j, then it is still selected with a cost less than c_j. Monotonicity guarantees that there exists a strategyproof mechanism for generalized set cover games using Algorithm 1 to compute its output. We then show how to compute the payment to each service provider efficiently. We assume that for any set S_j, if we remove S_j from $\mathcal{S}$, $\mathcal{S}$ still satisfies the coverage requirements of all elements in U. Otherwise, we call the set cover problem to be *monopoly*: the set S_j can charge an arbitrarily large cost in the monopoly game. The following presents our payment scheme for multiset multicover set cover problem.

Algorithm 4 Strategyproof payment $\mathcal{P}_j^{grd}$ to service provider $S_j \in \mathcal{C}_{grd}$

1: $\mathcal{C}_{grd} \leftarrow \emptyset$ and $s \leftarrow 1$;
2: $k' \leftarrow k$, $r'_i = r_i$ for each e_i;
3: **while** $k' > 0$ **do**
4: pick the set $S_t \neq S_j$ in $\mathcal{S} \setminus \mathcal{C}_{grd}$ with the minimum effective average cost;
5: Let v_t and v_j be the values of the sets S_t and S_j at this moment;
6: $\kappa(j, s) \leftarrow \frac{v_j}{v_t} c_t$ and $s \leftarrow s + 1$; $\mathcal{C}_{grd} \leftarrow \mathcal{C}_{grd} \cup \{S_t\}$; $k' \leftarrow k' - v_t$;
7: for each e_i, $r'_i \leftarrow \max\{0, r'_i - k_{t,i}\}$;
8: $\mathcal{P}_j^{grd} = \max_{t=1}^{s-1} \kappa(j, t)$ is the payment to selfish service provider S_j.

We show in [17] that the mechanism $M = (\mathcal{C}_{grd}, \mathcal{P}^{grd})$ is strategyproof (when the agent j does not lie about the set S_j of elements it can cover) and the payment $\mathcal{P}_j^{grd}$ is the minimum to the selfish service provider j among any strategyproof mechanism using Algorithm 1 as its output. We now consider the scenario when

agent j can also lie about S_j. Assume that agent j cannot lie upward[1], *i.e.*, it can only report a $S'_j \subseteq S_j$. We argue that agent j will not lie about its elements S_j. Notice that the value $\kappa(j,s)$ computed for the s-th round is $\kappa(j,s) = \frac{v_j}{v_t} c_t = \frac{\sum_{1 \le i \le n} \min(r'_i, k_{j,i})}{\sum_{1 \le i \le n} \min(r'_i, k_{t,i})} c_t$. Obviously v_j cannot increase when agent j reports any set $S'_j \subseteq S_j$. Thus, falsely reporting a smaller set S'_j will not improve the payment of agent j.

Theorem 5. *Algorithm 1 and 4 together define a* $\ln d_{max}$*-efficient strategyproof mechanism* $M = (\mathcal{C}_{grd}, \mathcal{P}^{grd})$ *for multiset multicover set cover game.*

5.2 Sharing the Payment Fairly

In the previous subsection, we only define what is the payment to a selfish service provider S_j. A remaining question is how the payment should be charged fairly (under some subtle definitions) to encourage cooperation among service receivers. One natural way of defining fair payment sharing is to extend the fair cost sharing method. Consider a strategyproof mechanism $M = (\mathcal{O}, \mathcal{P})$. Let $\mathcal{P}(T)$ be the total payment to the selfish service providers when T is the set of service receivers to be covered. A payment sharing scheme is simply a function $\pi(i,T)$ such that $\pi(i,T) = 0$ for any element $e_i \notin T$. A payment sharing scheme is called α*-budget-balanced* if $\alpha \cdot \mathcal{P}(T) \le \sum_{e_i \in T} \pi(i,T) \le \mathcal{P}(T)$. A payment sharing scheme is said to be a *core* if $\sum_{e_i \in S} \pi(i,T) \le \mathcal{P}(S)$ for any subset $S \subset T$. A payment sharing scheme is said to be an α*-core* if it is α-budget-balanced and it is a core. For payment method $\mathcal{P}^{grd}$, we prove in [17] that

Theorem 6. *There is no* α*-core payment sharing scheme for* $\mathcal{P}^{grd}$ *if* $\alpha > \frac{1}{\ln n}$.

It is easy to show that if we share the payment to a service provider equally among all service receivers covered by this set, the scheme is not in the core of the game. We leave it as an open problem whether we can design an α-core payment sharing scheme for the payment $\mathcal{P}^{grd}$ with $\alpha = O(\frac{1}{\ln n})$.

In the next, we study the cross-monotone payment sharing scheme. A payment sharing scheme is said to be *cross-monotone* if $\pi(i,T) \le \pi(i,S)$ for any two subsets $S \subset T$ and $i \in S$. A payment sharing scheme is said to be a *cross-monotone* α*-core* if it is α-budget-balanced and cross-monotone, and it is a core. We propose the following conjecture.

Conjecture 1. *For the strategyproof mechanism* $M = (\mathcal{C}_{grd}, \mathcal{P}^{grd})$ *of a set cover game, there is no payment sharing scheme* $\pi(\cdot,\cdot)$ *that is cross-monotone* α*-core for* $\alpha = \frac{1}{n} + \epsilon$.

In the remaining of this section we will present a cross-monotone budget-balanced payment sharing scheme for a strategyproof payment scheme of the

[1] This can be achieved by imposing a large enough penalty if an agent could not provide the claimed service when it is selected.

set cover game. Our payment sharing scheme is coupled with the following *least cost set* mechanism $M = (\mathcal{C}_{lcs}, \mathcal{P}^{lcs})$. It uses the output called *least cost set* $\mathcal{C}_{lcs}$ (described in Algorithm 5): for each service receiver e_i, we find the service provider S_j with the least cost efficiency $\frac{c_j}{\min(r_i,k_{j,i})}$ to cover the element e_i. New cost efficient sets are found till the cover requirement of e_i is satisfied. The payment (described in Algorithm 6) to a set S_j is defined as $\mathcal{P}_j^{lcs} = \max_{e_i \in U} p_j^i$, where p_j^i is the largest cost that S_j can declare while S_j is still selected to cover e_i. If the set S_j is not selected to cover e_i, then $p_j^i = 0$.

Algorithm 5 Least cost set greedy for multiset multicover game.

Let $\mathcal{C}_{lcs} \leftarrow \emptyset$.
for all element $e_i \in T$ **do**
  Let $r_i' \leftarrow r_i$;
  while $r_i' > 0$ **do**
    Find the set S_t with the minimum ratio $\min_{S_j \in \mathcal{S} - \mathcal{C}_{lcs}} \frac{c_j}{\min(k_{j,i}, r_i')}$;
    $r_i' \leftarrow r_i' - \min(k_{j,i}, r_i')$; $\mathcal{C}_{lcs} \leftarrow \mathcal{C}_{lcs} \cup \{S_t\}$.

Algorithm 6 Compute the payment $\mathcal{P}_j^{lcs}$ to a set S_j in $\mathcal{C}_{lcs}$

Let $\mathcal{C}_{lcs} \leftarrow \emptyset$, $p_j^i = 0$ for $1 \le i \le n$ and $s = 1$;
for all element $e_i \in T$ **do**
  Let $r_i' \leftarrow r_i$;
  while $r_i' > 0$ **do**
    Find the set $S_t \neq S_j$ with the minimum ratio $\min_{S_x \in \mathcal{S} - \mathcal{C}_{lcs} - \{S_j\}} \frac{c_x}{\min(k_{x,i}, r_i')}$;
    $\kappa(j,i,s) = \frac{\min(k_{j,i}, r_i')}{\min(k_{t,i}, r_i')} c_t$; $r_i' \leftarrow r_i' - \min(k_{j,i}, r_i')$; $\mathcal{C}_{lcs} \leftarrow \mathcal{C}_{lcs} \cup \{S_t\}$ and $s \leftarrow s + 1$;
  $p_j^i \leftarrow \max_{1 \le x < s} \kappa(j,i,s)$;
$\mathcal{P}_j^{lcs} \leftarrow \max_{1 \le i \le n} p_j^i$;

Theorem 7. *The mechanism $M = (\mathcal{C}_{lcs}, \mathcal{P}^{lcs})$ is $\frac{1}{2n}$-efficient and strategyproof.*

We then study how we charge the service receivers so that a budget-balance is achieved and the charging scheme also is fair under some concepts. Notice that, given a subset of elements T, we can view the total payments $\mathcal{P}(T)$ to all service providers covering T as a "cost" to T. The payment computed by mechanism $M = (\mathcal{C}_{lcs}, \mathcal{P}^{lcs})$ clearly is cohesive. Then naturally, we could use the cost-sharing schemes studied before to share this special cost among elements. However, it is easy to show by example that the previous cost-sharing schemes (studied in Section 3) are not in the core and also not cross-monotone.

Roughly speaking, our payment sharing scheme works as follows. Notice that a final payment to a set S_j is the maximum of payments p_j^i by all elements. Since different elements may have different value of payment to set S_j, the final payment $\mathcal{P}_j^{lcs}$ should be shared *proportionally* to their values, not *equally* among them as cost-sharing.

Algorithm 7 Sharing MV cost $\mathcal{P}$ among receivers

1: Initialize $\xi(i,U)=0$ and $\zeta_j(i,U)=0$. Here $\zeta_j(i,U)$ denotes the payment to set S_j shared by the element e_i when the set of elements is U.
2: **for all** $S_j \in \mathcal{S}$ **do**
3: For all elements e_i, we compute the payment p_j^i. Sort the payments p_j^i, $1 \le i \le n$, in an increasing order. Assume that $p_j^{\sigma(1)}$, $p_j^{\sigma(2)}$, $\cdots$, $p_j^{\sigma(n-1)}$, $p_j^{\sigma(n)}$ are the sorted list of payments in an incremental order.
4: For elements $e_{\sigma(1)}, \cdots, e_{\sigma(n)}$, let $\zeta_j(\sigma(i),U) \leftarrow \sum_{t=1}^{i} \frac{p_j^{\sigma(t)} - p_j^{\sigma(t-1)}}{n-t+1}$. Here we assume that $p_j^{\sigma(0)}=0$. Update the payment sharing as follows: $\xi(i,U) = \xi(i,U) + \zeta_j(i,U)$ for each $e_i \in U$.
5: $\xi(i,U)$ is the final payment sharing of service receiver e_i.

Our payment sharing method described in Algorithm 7 applies to a more general cost function. A cost function $\mathcal{P}$ is said to be *maximum-view cost* (MV cost) if it is defined as $\mathcal{P}_j = \max_{e_i \in U} p_j^i$ where p_j^i is the *view* of the cost of set S_j by element e_i. Obviously, the traditional cost c is a MV cost function by setting $p_j^i = c_j$ for each element e_i. The payment function $\mathcal{P}^{lcs}$ is also a MV cost function.

A service receiver is called *free-rider* in a payment sharing scheme if its shared total payment is no more than $\frac{1}{n}$ of its total payment it has to pay if it acts alone. Notice that, when a service receiver acts alone, the same mechanism is applied to compute the payment to the service providers.

Theorem 8. *The payment sharing scheme described in Algorithm 7 is budget-balanced, cross-monotone, in the core and does not permit free-rider.*

6 Conclusion

We studied cost sharing and strategyproof mechanisms for various set cover games. We gave an efficient cost allocation method that always recovers $\frac{1}{\ln d_{max}}$ of the total cost, where d_{max} is the maximum size of all sets. We gave an efficient cost sharing scheme that is $\frac{1}{2n}$-budget-balanced, core and cross-monotone. When the elements to be covered are selfish agents with privately known valuations, we presented a strategyproof charging mechanism. When the sets are selfish agents with privately known costs, we presented two strategyproof payment mechanisms in which each set maximizes its profit when it reports its cost truthfully. We also showed how to *fairly* share the payments to all sets among the elements.

There are several open problems left for future works. Are the bounds on the α-budget-balanced cost sharing schemes tight, although we proved that they are asymptotically tight? Consider the strategyproof mechanism $M = (\mathcal{C}_{grd}, \mathcal{P}^{grd})$. Is there a payment sharing method that is $\frac{1}{\ln n}$-core? Is there a payment sharing method that is cross-monotone $\frac{1}{n}$-core? Is this $\frac{1}{n}$ a tight lower bound?

References

1. Chvátal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operation Research **4** (1979) 233–235
2. Feige, U.: A threshold of for approximating set cover. JACM **45** (1998) 634–652
3. Slavík, P.: Improved performance of the greedy algorithm for partial cover. Information Processing Letters **64** (1997) 251–254
4. Immorlica, N., Mahdian, M., Mirrokni, V.S.: Limitations of cross-monotonic cost-sharing schemes. In: ACM SODA 2005. To appear.
5. Marbach, P.: Priority service and max-min fairness. IEEE/ACM Transactions on Networking **11** (2003) 733–746
6. Radunović, B., Boudec, J.Y.L.: A unified framework for max-min and min-max fairness with applications. In: Proceedings of 40th Annual Allerton Conference on Communication, Control, and Computing. (2002)
7. Osborne, M.J., Rubinstein, A.: A course in game theory. The MIT Press (1994)
8. Moulin, H., Shenker, S.: Strategyproof sharing of submodular costs:budget balance versus efficiency. Economic Theory **18** (2001) 511–533
9. Devanur, N., Mihail, M., Vazirani, V.: Strategyproof cost-sharing mechanisms for set cover and facility location games. In: ACM EC (2003) 108–114
10. Pál, M., Tardos, E.: Group strategyproof mechanisms via primal-dual algorithms. In: IEEE FOCS (2003) 584–593
11. Feigenbaum, J., Papadimitriou, C.H., Shenker, S.: Sharing the cost of multicast transmissions. Journal of Computer and System Sciences **63** (2001) 21–41
12. Libman, L., Orda, A.: Atomic resource sharing in noncooperative networks. Telecommunication Systems **17** (2001) 385–409
13. Shenker, S., Clark, D., Estrin, D., Herzog, S.: Pricing in computer networks: Reshaping the research agenda. ACM Computer Comm. Review **26** (1996) 19–43
14. Feigenbaum, J., Krishnamurthy, A., Sami, R., Shenker, S.: Hardness results for multicast cost sharing. Theoretical Computer Science **304** (2003) 215–236
15. Archer, A., Feigenbaum, J., Krishnamurthy, A., Sami, R., Shenker, S.: Approximation and collusion in multicast cost sharing. Games and Economic Behavior **47** (2004) 36–71
16. Herzog, S., Shenker, S., Estrin, D.: Sharing the cost of multicast trees: An axiomatic analysis. IEEE/ACM Transactions on Networking **5** (1997) 847–860
17. Li, X.Y., Sun, Z., Wang, W.: Cost sharing and strategyproof mechanisms for set cover games (2004) at http://www.cs.iit.edu/∼xli/publications-select.htm.
18. Vickrey, W.: Counterspeculation, auctions and competitive sealed tenders. Journal of Finance **16** (1961) 8–37
19. Clarke, E.H.: Multipart pricing of public goods. Public Choice **11** (1971) 17–33
20. Groves, T.: Incentives in teams. Econometrica **41** (1973) 617–631
21. Lehmann, D.J., Oćallaghan, L.I., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. JACM **49** (2002) 577–602

On Weighted Balls-into-Bins Games*

Petra Berenbrink[1], Tom Friedetzky[2], Zengjian Hu[3], and Russell Martin[4]

[1] School of Computing Science, Simon Fraser University,
Burnaby, B.C., V5A 1S6, Canada
http://www.cs.sfu.ca/~petra/
[2] Department of Computer Science, University of Durham,
Durham, DH1 3LE, U.K.
http://www.dur.ac.uk/tom.friedetzky/
[3] School of Computing Science, Simon Fraser University,
Burnaby, B.C., V5A 1S6, Canada
http://www.cs.sfu.ca/~zhu/
[4] Department of Computer Science, University of Warwick,
Coventry, CV4 7AL, U.K.
http://www.dcs.warwick.ac.uk/~martin/

Abstract. We consider the well-known problem of randomly allocating m balls into n bins. We investigate various properties of *single-choice games* as well as *multiple-choice games* in the context of *weighted balls*. We are particularly interested in questions that are concerned with the *distribution* of ball weights, and the *order* in which balls are allocated. Do any of these parameters influence the maximum expected load of any bin, and if yes, then how?

The problem of weighted balls is of practical relevance. Balls-into-bins games are frequently used to conveniently model load balancing problems. Here, weights can be used to model resource requirements of the jobs, i.e., memory or running time.

1 Introduction

The *balls-into-bins game*, also referred to as *occupancy problem* or *allocation process*, is a well known and much investigated model. The goal of a (static) balls-into-bins game is to sequentially allocate, at random, a set of m independent balls (tasks, jobs, ...) into a set of n bins (printers, servers, ...), such that the maximum number of balls in any bin is minimised. In the dynamic case, we do not have a fixed number of balls but rather new balls arrive over time (and existing ones may be removed).

In this paper, we are interested in *static sequential games*, where a fixed number of balls, m, are allocated one after the other; see [13] for an overview of balls-into-bins games in different settings. The classical *single-choice game* allocates each ball to a bin that is chosen independently and uniformly at random

* A full version of this paper including all proofs can be found at http://www.dur.ac.uk/tom.friedetzky/pub/

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 231–243, 2005.

(i.u.r.). For $m = n$ balls and n bins the maximum *load* (maximum number of balls) in any bin is $\Theta(\log(n)/\log\log(n))$. More generally, for m balls and n bins the maximum load is $(m/n) + \Theta(\sqrt{m \log n/n})$. Surprisingly, the maximum load can be decreased dramatically by allowing every ball to i.u.r. choose a small number of $d > 1$ bins. The ball is then allocated to a least loaded of the d chosen bins. Then, the maximum load drops to $\Theta(\log\log(n)/\log(d))$ (see [1]) in the $m = n$ case, and $(m/n) + \Theta(\log\log(n)/\log(d))$ in the general case, respectively (see [2]). Notice that the results cited above all hold *with high probability*[1] (w.h.p.). Following [1], we refer to the multiple-choice algorithm defined above as Greedy[d].

Most work done so far assumes that the balls are *uniform* and *indistinguishable*. In this paper we concentrate on the *weighted case* where the i-th ball comes with a weight w_i. We define the *load* of a bin to be the sum of the weights of the balls allocated to it. In [5] the authors compare the maximum load of weighted balls-into-bins games with the maximum load of corresponding uniform games. They compare the maximum load of a game with m weighted balls with maximum weight 1 and total weight $W = w_1 + \cdots + w_m$ to a game with cW uniform balls with constant $c \approx 4$. Basically, they show that the maximum load of the weighted game is not larger than the load of the game with uniform balls (which has a slightly larger total weight). Their approach can be used for a variety of balls-into-bins games and can be regarded as a general framework. See [4] for more details.

However, the results of [5] seem to be somewhat unsatisfactory. The authors compare the allocation of a (possibly huge) number of "small" weighted balls with an allocation of fewer but "heavier" uniform balls. Intuitively, it should be clear that it is better to allocate many "small" balls compared to fewer "big" balls. After all, the many small balls come with more random choices. The main goal of this paper is to get tighter results for the allocation of weighted balls, both for the single-choice and the multiple-choice game. To show our results we will use the majorisation technique introduced in [1].

1.1 Known Results

Single-Choice Game. In [13] the authors give a tight bound on the maximum load of any bin when m *uniform* balls are allocated uniformly at random into n bins.

In [11] Koutsoupias et al. consider the random allocation of weighted balls. Similar to [5], they compare the maximum load of an allocation of weighted balls to that of an allocation of a smaller number of uniform balls with a larger total weight. They repeatedly fuse the two smallest balls together to form one larger ball until the weights of all balls are within a factor of two of each other. They show that the bin loads after the allocation of the weighted balls are *majorised* by the loads of the bins after the allocation of the balls generated by the fusion

[1] We say an event A occurs with high probability, if $\Pr[A] \geq 1 - 1/n^{\alpha}$ for some constant $\alpha \geq 1$.

process. Their approach also applies to more general games in which balls can be allocated into bins with nonuniform probabilities.

Multiple-Choice Game. During recent years much research has been done for games with multiple choices in different settings. See [13] for a nice overview. Here, we shall only mention the "classical" and most recent results.

Azar et al. [1] introduced Greedy[d] to allocate n balls into n bins. Their algorithm Greedy[d] chooses d bins i.u.r. for each ball and allocates the ball into a bin with minimum load. They show that after placing n balls the maximum load is $\Theta(\log\log(n)/\log(d)+1)$, w.h.p. Compared to single-choice games, this is an *exponential* decrease of the maximum load. Vöcking [19] introduced the Always-Go-Left protocol yielding a maximum load of $(\log\log n)/d$, w.h.p. In [18], Sanders et al. show that in the general case it is possible to achieve a maximum load of $\lceil m/n\rceil+1$, w.h.p., using a *centralised* flow algorithm. In [2] the authors analyse Greedy[d] for $m \gg n$. They show that the maximum load is $m/n+\log\log(n)$, w.h.p. This shows that the multiple-choice process behaves inherently different from the single-choice process, where it can be shown that the difference between the maximum load and the average load depends on m. They also show a *memorylessness* property of the Greedy process, i.e., whatever the situation is after allocation of some ball, after sufficiently many additional balls the maximum load of any bin can again be bounded as expected. Finally, Mitzenmacher et al. [14] show that a similar performance gain occurs if the process is allowed to store the location of the least loaded bin in memory.

1.2 Model and Definitions

We assume that we have m balls and n bins. In the following we denote the set $\{1,\dots,m\}$ by $[m]$. Ball i has weight w_i for all $i\in[m]$. Let $w=(w_1,\dots,w_m)$ be the *weight vector* of the balls. We assume $w_i>0$ for all $i\in[m]$. $W=\sum_{i=1}^{m} w_i$ is the *total weight* of the balls. If $w_1=\cdots=w_m$ we refer to the balls as *uniform*. In this case, we normalise the ball weights such that $w_i=1$ for $\forall i\in[m]$.

The *load* of a given bin is the sum of the weights of all balls allocated to it. In the case of uniform balls the load is simply the number of balls allocated to the bin. The status of an allocation is described by a *load vector* $L(w)=(\ell_1(w),\dots,\ell_n(w))$. Here, ℓ_i is the load of the i-th bin after the allocation of weight vector w. Whenever the context is clear we write $L=(\ell_1,\dots,\ell_n)$. In some cases we consider the change that occurs in a system after allocating some number of additional balls. Then we define L_t to be the load vector after the allocation of the first t balls with weights $w_1,\dots,w_t$ for $1\le t\le m$. In many cases we will *normalise* a load vector L by assuming a non-increasing order of bin loads, i.e. $\ell_1\ge\ell_2\ge\cdots\ge\ell_n$. We then define $S_i(w)=\sum_{j=1}^{i}\ell_j(w)$ as the total load of the i highest-loaded bins. Again, when the context is clear we shall drop the "w" and write $S_i=\sum_{j=1}^{i}\ell_j$. Finally, in what follows, we let $\Omega=[n]$.

To compare two load vectors and also the balancedness of vectors of balls weights we use the concept of *majorisation*. First, we briefly review the notion of majorisation from [12].

Definition 1. *For two normalised vectors* $w = (w_1, \ldots, w_m) \in \mathbb{R}^m$ *and* $w' = (w'_1, \ldots, w'_m) \in \mathbb{R}^m$ *with* $\sum_{i=1}^{m} w_i = \sum_{i=1}^{m} w'_i$*, we say that* w' majorises w*, written* $w' \succ w$*, if* $\sum_{i=1}^{k} w'_i \geq \sum_{i=1}^{k} w_i$ *for all* $1 \leq k \leq m$*.*

Majorisation is a strict partial ordering between (normalised) vectors of the same dimensionality. Intuitively, vector v' *majorises* another vector v if v is "more spread out", or "more balanced", than v'. In the following we will say that weight vector w is more balanced than weight vector w' if w' majorises w, and we will use the term majorisation if we refer to load vectors.

1.3 New Results

In the next section we first present some additional definitions that we will use later on in this paper. Section 3 is concerned with the single-choice game. In Theorem 2 we fix the number of balls and show that the maximum load is smaller for more balanced ball weight vectors. In more detail, we allocate two sets of balls into bins, where the first set has a more even weight distribution than the second one, i.e., the second corresponding weight vector majorises the first one. We show that the expected maximum load after allocating the first set is smaller than the one after allocation the second set. This also holds for the sum of the loads of the i largest bins. One could say that the *majorisation is preserved*: if one weight vector majorises another one, then we have the same order with respect to the resulting expected bin load vectors. Hence, uniform balls minimise the expected maximum load. Theorem 2 uses majorisation together with T-transformations (see the definition in the next section), thereby allowing us to compare sets of balls that only differ in *one* pair of balls.

Corollary 2 extends the results showing that the allocation of a large number of small balls with total weight W ends up with a smaller expected maximum load than the allocation of a smaller number of balls with the same total weight. We also show that the results are still true for many other random functions that are used to allocate the balls into the bins. Our results are much stronger than the ones of [11] since we compare arbitrary systems with the same number of balls and the same total weight. We also consider the entire load distribution and not only the maximum load.

Section 4 deals with multiple-choice games. The main result here is Theorem 3. It shows that, for sufficiently many balls, allocation of uniform balls is *not* necessarily better than allocation of weighted balls. It is better to allocate first the "big balls" and then some smaller balls on top of them, instead of allocation the same number of average sized balls. This result uses the memorylessness property of [2].

For fewer balls we show in Theorem 4 that the majorisation order is not generally preserved. Assume that we have two systems $\mathcal{A}$ and $\mathcal{B}$, and that the

load vector of system $\mathcal{A}$ is majorised by the load vector of system $\mathcal{B}$. Now, throwing only *one* additional ball into both systems may reverse the majorisation order and suddenly $\mathcal{B}$ is majorised by $\mathcal{A}$. The previous results mentioned for the single-choice game use the majorisation technique inductively. Unfortunately, it seems difficult to use T-transformations and the majorisation technique to obtain results for weighted balls in the multiple-choice game. We also present several examples showing that, for the case of a small number of balls with multiple-choices, the maximum load is not necessarily smaller if we allocate more evenly weighted balls.

2 Majorisation and T-Transformations

In Section 1.2 we defined the concept of majorisation. In [1] Azar et al. use this concept for random processes. Here we give a slightly different definition adjusted for our purposes.

Definition 2 *(***Majorisation***).* *Let w and w' be two weight vectors with m balls, and let Ω^m be the set of all possible random choices for Greedy applied on m balls. Define $w(\omega)$ (respectively, $w'(\omega)$) to be the allocation resulting from the choices $\omega \in \Omega^m$, and let $f : \Omega^m \longrightarrow \Omega^m$ be a* one-to-one *correspondence. Then we say that w' is majorised by w if there exists a function f such that for any $\omega \in \Omega^m$ we have $w(\omega) \succ w'(f(\omega))$.*

A slightly weaker form of the majorisation is the *expected majorisation* defined below. We will use it in order to compare the allocation of two different load vectors with each other.

Definition 3 *(***Expected majorisation***).* *Let w and w' be two weight vectors with m balls, and let Ω^m be the set of all possible random choices. Let $L(w,\omega) = \ell_1(w,\omega),\ldots,\ell_n(w,\omega)$ (resp., $L'(w',\omega) = \ell_1(w',\omega),\ldots,\ell_n(w',\omega)$) be the normalised load vector that results from the allocation of w (respectively, w') using $\omega \in \Omega^m$. Let $S_i(w,\omega) = \sum_{j=1}^{i} \ell_j(w,\omega)$ and $S_i(w',\omega) = \sum_{j=1}^{i} \ell_j(w',\omega)$. Then we say that $L(w')$ is* expectedly majorised *by $L(w)$ if for all $i \in [n]$, we have $E[S_i(w)] \geq E[S_i(w')]$.*

Note that the expectation is over all possible n^m elements (selected uniformly at random) in Ω^m.

Now we introduce a class of linear transformations on vectors called *T-transformations* which are crucial to our later analysis. We write $w \overset{T}{\Longrightarrow} w'$, meaning that w' can be derived from w by applying one T-transformation.

Recall that a square matrix $\Pi = (\pi_{ij})$ is said to be *doubly stochastic* if all $\pi_{ij} \geq 0$, and each row and column is one. Π is called a *permutation matrix* if each row and each column contains exactly one unit and all other entries are zero (in particular, a permutation matrix is doubly stochastic).

Definition 4 *(**T-transformation**). A* T-transformation *matrix T has the form $T = \lambda I + (1-\lambda)Q$, where $0 \le \lambda \le 1$, I is the identity matrix, and Q is a permutation matrix that swaps exactly two coordinates. Thus, for some vector x of correct dimensionality, $xT = (x_1, \ldots, x_{j-1}, \lambda x_j + (1-\lambda)x_k, x_{j+1}, \ldots, x_{k-1}, \lambda x_k + (1-\lambda)x_j, x_{k+1}, \ldots, x_m)$.*

T-transformations and majorisation are closely linked by the following lemma (see [12]).

Lemma 1. *For $w, w' \in \mathbb{R}^m$, $w \succ w'$ if and only if w' can be derived from w by successive applications of at most $m-1$ T-transformations.*

One of the fundamental theorems in the theory of majorisation is the following.

Theorem 1. *(Hardy, Littlewood and Pólya, 1929). For $w, w' \in \mathbb{R}^m, w \succ w'$ if and only if $w' = wP$, for some doubly stochastic matrix P.*

3 Weighted Single-Choice Games

In this section we study the classical balls-into-bins game where every ball has only one random choice. Let w and w' be two $m-$dimensional weight vectors. Recall that $S_i(w)$ is defined to be the random variable counting the cumulative loads of the i largest bins after allocating w. In this section we show that, if there exist a majorisation order between two weight vectors w and w', the same order holds for $E[S_i(w)]$ and $E[S_i(w')]$. This implies that, if w majorises w', the expected maximum load after allocating w is larger than or equal to the expected maximum load after allocating w'.

Note that in the single-choice game, the final load distribution does not depend upon the order in which the balls are allocated. From Lemma 1 we know that, if $w \succ w'$, then w' can be derived from w by applying at most $m-1$ T-transformations. Thus, it is sufficient to show the case in which w' can be derived from w by applying one T-transformation, which is what we do in Lemma 2.

Lemma 2. *If $w \overset{T}{\Longrightarrow} w'$ (i.e. $w \succ w'$), then $E[S_i(w')] \le E[S_i(w)]$ for $\forall i \in [n]$.*

Proof. (Sketch) Let $w = (w_1, \ldots, w_m)$. According to the definition of a T-transformation, for some $0 \le \lambda \le 1$ we have

$$w' = wT = (w_1, \ldots, w_{j-1}, \lambda w_j + (1-\lambda)w_k, w_{j+1}, \ldots, w_{k-1}, \lambda w_k + (1-\lambda)w_j, w_{k+1}, \ldots, w_m).$$

We define $y_j = \max\{\lambda w_j + (1-\lambda)w_k, \lambda w_k + (1-\lambda)w_j\}, y_k = \min\{\lambda w_j + (1-\lambda)w_k, \lambda w_k + (1-\lambda)w_j\}$.

Since the final allocation does not depend on the order in which the balls are allocated, we can assume in the following that both w_j, w_k and y_j, y_k are allocated in the last two steps. Now fix the random choices for the first $m-2$

balls and let $\ell = (\ell_1, \ldots, \ell_n)$ be the resulting normalised load vector from the allocation of those balls. Let $\Omega^2 = [n]^2$ be the set of random choices of the last two balls. Note that every random choice in Ω^2 occurs with probability $1/n^2$.

Fix a pair of choices (p, q) for the last two balls and let $L(\ell, (w_j, p), (w_k, q))$ be load vector after placing the ball with weight w_j into the bin with rank p in ℓ and the ball with weight w_k to the bin with rank q in ℓ. (Note, after the allocation of w_j the order of the bins might change but q still refers to the old order. Let $S_i(\ell, (w_j, p), (w_k, q))$ be the cumulative load of the i largest bins of $L(\ell, (w_j, p), (w_k, q))$. Similarly define $L(\ell, (y_j, p), (y_k, q))$ and $S_i(\ell, (y_j, p), (y_k, q))$. To prove this lemma we compare the two choices (p, q) and (q, p) with each other and show that for all ℓ

$$\begin{aligned} &S_i(\ell, (w_j, p), (w_k, q)) + S_i(\ell, (w_j, q), (w_k, p)) \\ \geq\ &S_i(\ell, (y_j, p), (y_k, q)) + S_i(\ell, (y_j, q), (y_k, p)) \end{aligned}$$

Since we compute expected values over all pairs (p, q), this shows that the *expected* cumulative loads of the i largest bins of both systems also obey the same order.

The repeated application of Lemma 2 can now be used to generalize the majorisation result for vectors that only differ by a single T-transformation to vectors that differ by several T-transformations. This results in the following theorem that is presented without formal proof.

Theorem 2. *If $w \succ w'$, then for $\forall i \in [n]$, $E[S_i(w')] \leq E[S_i(w)]$.*

Finally, it is clear that the uniform weight vector is majorised by all other vectors with same dimension and same total weight. Using Theorem 2, we get the following corollary.

Corollary 1. *Let $w = (w_1, \ldots, w_m)$, $W = \sum_{i=1}^{m} w_i$, and $w' = (\frac{W}{m}, \ldots, \frac{W}{m})$. For $\forall i \in [n]$, we have $E[S_i(w')] \leq E[S_i(w)]$.*

Proof. Note that $w' = wP$, where $P = (p_{ij})$ and $p_{ij} = \frac{1}{m}$ $\forall i, j \in [m]$. Clearly P is a doubly stochastic matrix. Hence by Lemma 1, $w \succ w'$. Consequently, from Theorem 2 we have $E[S_i(w')] \leq E[S_i(w)]$.

Theorem 2 also shows that an allocation of a large number of small balls with total weight W ends up with a smaller expected load than the allocation of a smaller number of balls with the same total weight.

Corollary 2. *Let $w = (w_1, \ldots, w_m)$ and $W = \sum_{i=1}^{m} w_i$. Suppose that $w' = (w'_1, \ldots, w'_{m'})$ with $m \leq m'$, and also that $W = \sum_{i=1}^{m} w'_i$. If $w \succ w'$ we have $E[S_i(w')] \leq E[S_i(w)]$ $\forall i \in [n]$.*

[In this case the relation $w \succ w'$ must be treated somewhat loosely because the vectors do not necessarily have the same length, but the meaning should be clear, namely that $\sum_{i=1}^{j} w_i \geq \sum_{i=1}^{j} w'_i$ for all $j \in [m]$].

Proof. Simply add zeros to w until it has the same dimension than w'.

It is easy to see that we can generalise the result to other probability distributions that are used to chose the bins.

Corollary 3. *If $w \succ w'$, and the probability that a ball is allocated to bin b_i, $1 \leq i \leq n$, is the same for all balls, then we have for $\forall i \in [n]$, $E[S_i(w')] \leq E[S_i(w)]$.*

4 Weighted Multiple-Choice Games

In the first sub-section we show that for multiple-choice games it is not always better to allocate uniform balls. For $m \gg n$ we construct a set of weighted balls that ends up with a smaller maximum load than a set of uniform balls with the same total weight. The second sub-section considers the case where m is not much larger than n. As we will argue in the beginning of that section, it appears that it may not be possible to use the majorisation technique to get tight results for the weighted multiple-choice game. This is due to the fact that the order in which weighted balls are allocated is crucial, but the majorisation order is not necessarily preserved for weighted balls in the multiple-choice game (in contrast to [1] for uniform balls). We discuss several open questions and give some weight vectors that result in a smaller expected maximum load than uniform vectors with the same total weight.

4.1 Large Number of Balls

We compare two systems, $\mathcal{A}$ and $\mathcal{B}$, respectively. In $\mathcal{A}$ we allocate $m/2$ balls of weight 3 each and thereafter $m/2$ balls of weight 1 each, using the multiple-choice strategy. System $\mathcal{B}$ is the uniform counterpart of $\mathcal{A}$ where all balls have weight 2. We show that the expected maximum load in $\mathcal{A}$ is strictly smaller than that in $\mathcal{B}$. We will use the *short term memory property* stated below in Lemma 3. See [2] for a proof. Basically, this property says that after allocating a sufficiently large number of balls, the load depends on the last poly(n) many balls only. If m is now chosen large enough (but polynomially large in n suffices), then the maximum load is w.h.p. upper bounded by $2m/n + \log\log n$. In the case of balls with weight 2, the maximum load is w.h.p. upper bounded by $2m/n + 2\log\log n$. Since [2] gives only upper bounds on the load, we can not use the result directly. We introduce two auxiliary systems named $\mathcal{C}$ and $\mathcal{D}$, respectively. System $\mathcal{C}$ is derived from system $\mathcal{A}$, and $\mathcal{D}$ is derived from $\mathcal{B}$. The only difference is that in systems $\mathcal{C}$ and $\mathcal{D}$ we allocate the first $m/2$ balls optimally (i.e., we always place the balls into the least loaded bins). In Lemma 5 we first show that the maximum loads of $\mathcal{A}$ and $\mathcal{C}$ will be nearly indistinguishable after allocating all the balls. Similarly, the maximum loads of $\mathcal{B}$ and $\mathcal{D}$ will be nearly indistinguishable. Moreover, we show that the expected maximum load in $\mathcal{D}$ is larger than in $\mathcal{C}$. Then we can show that the expected maximum load in $\mathcal{A}$ is smaller than in $\mathcal{B}$ (Theorem 3). For an overview, we refer to Table 1.

To state the short memory property we need one more definition. For any two random variables X and Y defined jointly on the same sample space, the

Table 1. Systems $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$

	First $m/2$ balls		Last $m/2$ balls	
Systems	ball weights	algorithm	ball weights	algorithm
$\mathcal{A}$	3	Greedy[d]	1	Greedy[d]
$\mathcal{B}$	2	Greedy[d]	2	Greedy[d]
$\mathcal{C}$	3	Optimal	1	Greedy[d]
$\mathcal{D}$	2	Optimal	2	Greedy[d]

variation distance between $\mathcal{L}(X)$ and $\mathcal{L}(Y)$ is defined as: $\|\mathcal{L}(X) - \mathcal{L}(Y)\| = \sup_A |\Pr(X \in A) - \Pr(Y \in A)|$. The following Lemma is from [2].

Lemma 3. *Suppose $L_0 = (\ell_1, \ldots, \ell_n)$ is an arbitrary* normalised *load vector describing an allocation of m balls into n bins. Define $\Delta = \ell_1 - \ell_n$ to be the maximum load difference in L_0. Let L'_0 be the load vector describing the optimal allocation of the same number of balls to n bins. Let L_k and L'_k, respectively, denote the vectors obtained after inserting k further balls to both systems using the multiple-choice algorithm, Then: $|| \mathcal{L}(L_k) - \mathcal{L}(L'_k) || \leq k^{-\alpha}$. for $k \geq n^5 \cdot \Delta$, where α is an arbitrary constant.*

Intuitively, Lemma 3 indicates that given any configuration with a maximum difference Δ, in $\Delta \cdot \text{poly}(n)$ steps the system "forgets" the difference, i.e., the allocation is nearly indistinguishable from the allocation obtained by starting from a completely balanced system. This is in contrast to the single-choice game requiring $\Delta^2 \cdot \text{poly}(n)$ steps in order to "forget" a load difference Δ (see [2]).

Lemma 4. *Suppose we allocate m balls to n bins using Greedy[d] with $d \geq 2$, $m \gg n$. Then the number of bins with load at least $\frac{m}{n} + i + \gamma$ is bounded above by $n \cdot \exp(-d^i)$, w.h.p, where γ denotes a suitable constant. In particular, the maximum load is $\frac{m}{n} + \frac{\log \log n}{\log d} \pm \Theta(1)$ w.h.p.*

Let $L_i(\mathcal{A})$ (or $L_i(\mathcal{B})$, $L_i(\mathcal{C})$, $L_i(\mathcal{D})$) be the maximum load in System $\mathcal{A}$ (respectively, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$) after the allocation of the first i balls. If we refer to the maximum load after the allocation of all m balls we will simply write $L(\mathcal{A})$ (or $L(\mathcal{B})$, $L(\mathcal{C})$, $L(\mathcal{D})$). The following lemma compares the load of the four systems.

Lemma 5. *Let $m = \Omega(n^7)$.*

1. $|\, E[L(\mathcal{A})] - E[L(\mathcal{C})]\, | \leq m^{-\beta}$, *where β is an arbitrary constant.*
2. $|\, E[L(\mathcal{B})] - E[L(\mathcal{D})]\, | \leq m^{-\beta'}$, *where β' is an arbitrary constant.*
3. $E[L(\mathcal{D})] - E[L(\mathcal{C})] \geq \frac{\log \log n}{\log d} - \Theta(1)$.

Finally, we present the main result of this section, showing that uniform balls do not necessarily minimize the maximum load in the multiple-choice game.

Theorem 3. $E[L(\mathcal{B})] \geq E[L(\mathcal{A})] + \frac{\log \log n}{\log d} - \Theta(1)$.

4.2 Majorisation Order for Arbitrary Values of m

In this section we consider the Greedy[2] process applied on weighted balls, but most of the results can be generalised to the Greedy[d] process for $d > 2$. Just to remind you, in the Greedy[2] process each ball sequentially picks i.u.r. two bins and the current ball is allocated in the least loaded of the two bins (ties can be broken arbitrarily). This means, of course, that a bin with relative low load is more likely to get an additional ball than one of the highly loaded bins.

Another way to model the process is the following: Assume that the load vector of the bins are normalised, i.e. $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_n$. If we now place an additional ball into the bins, the ball will be allocated to bin i with probability $(i^d - (i-1)^d)/n^d$, since all d choices have to be among the first i bins, and at least one choice has to be i. For $d = 2$ this simplifies to $(2i-1)/n$. Hence, in this fashion, the process can be viewed as a "one choice process", provided after the allocation of each ball the system is re-normalised. This means that he load distribution of the bins depends highly on the order in which the balls are allocated.

Unfortunately, the dependence of the final load distribution on the order in which the balls are allocated makes it very hard to get tight bounds using the majorisation technique together with T-transformations. Theorem 2 highly depends on the fact that we can assume that w_j and w_k (y_j and y_k) are allocated at the very end of the process, an assumption that can not be used in the multiple-choice game. In order to use T-transformations in this case, we would again need a result that shows that the majorisation order is preserved when we add more (similar) balls into the system. We need a result showing that if $\mathcal{A} \succ \mathcal{B}$ and we add an additional ball to both $\mathcal{A}$ and $\mathcal{B}$, after the allocation we still have $\mathcal{A}' \succ \mathcal{B}'$ (where $\mathcal{A}'$ and $\mathcal{B}'$ denote the new systems with the one additional ball). While this is true for uniform balls (see [1]), the following easy example shows that it is not true for weighted balls. Let $\mathcal{A} = (7, 6, 5, 0, \ldots, 0)$ and $\mathcal{B} = (7, 5.8, 5.2, 0 \ldots, 0)$. If we now allocate one more ball with weight $w = 2$ into the pair of systems, with probability $5/n^2$ the ball is allocated to the third bin in both systems and we have $\mathcal{A}' = (7, 7, 6, 0, \ldots, 0)$ and $\mathcal{B}' = (7.2, 7, 5.8, 0, \ldots, 0)$, hence $\mathcal{B}' \succ \mathcal{A}'$. Alternatively, with probability $3/n^2$ the ball is allocated to the second bin in each system resulting in load vectors $\mathcal{A}'' = (8, 7, 5, 0, \ldots, 0)$ and $\mathcal{B}'' = (7.8, 7, 5.2, 0, \ldots, 0)$ and in this case we have $\mathcal{A}'' \succ \mathcal{B}''$. Note that the load distributions of $\mathcal{A}$ and $\mathcal{B}$ are not "atypical", but they can easily come up using Greedy[2]. For example weight vectors $w = (7, 6, 5, 2)$ and $w' = (7, 5.8, 5.2, 2)$ with $w \succ w'$ will do the job for values of n large enough (≈ 16). This shows that the expected maximum load after the allocation of w using Greedy[2] is majorised by the one of w'.

The next lemma gives another example showing that the majorisation relation need *not* be preserved for weighted balls in the multiple-choice game. The idea is that we can consider two systems $\mathcal{C}$ and $\mathcal{D}$ where $\mathcal{C} \succ \mathcal{D}$, but by adding one additional ball (with large weight w), we then have $\mathcal{D}' \succ \mathcal{C}'$. It is easy to generalise the lemma to cases where w is not larger than the maximum bin load to show that the majorisation relation need not be preserved.

Lemma 6. *Let v and u be two (normalised) load vectors with $v \overset{T}{\Longrightarrow} u$ (so $v \succ u$). Let w be the weight of an additional ball with $w > v_1$. Let v', u' be the new (normalised) load vectors after allocating the additional ball into the systems with loads v and u. Then we have $E[u'_1] > E[v'_1]$.*

We can easily extend the above result to the following.

Corollary 4. *Let v and u be two load vectors with $v \succ u$. Let w be the weight of an additional ball with $w \geq v_1$. Let v', u' be the load vectors after the allocation of the additional ball into v and u. Then we have $E[u'_1] > E[v'_1]$.*

Lemma 6 and the example preceding that lemma both showed that a more unbalanced weight vector can end up with a smaller expected maximum load. However, in those cases we assumed that the number of bins is larger than the number of balls, or that one of the balls is very big. Simulation results (see full version) show that for most weight vectors w, w' with $w \succ w'$ the expected maximum load after the allocation of w' is smaller than the one after the allocation of w. Unfortunately, we have been unable to show this result formally.

Order of the Balls. Another interesting question concerns the order of allocating balls under the multiple-choice scenario. In the case that $m \geq n$ we conjecture that if all the balls are allocated in decreasing order, the expected maximum is the smallest among all possible permutations. This is more or less intuitive since if we always allocate bigger balls first, the chances would be low to place the remaining balls in those bins which are already occupied by the bigger balls. However, we still do not know how to prove this conjecture. We can answer the peer question: what about if we allocate balls in increasing order? The next observation shows that the increasing order does *not* always produce the worst outcome.

Observation 4. *Fix a set of weighted balls. The maximum load is not necessarily maximised by allocating the balls in increasing order.*

Proof. We compare two systems $\mathcal{A}$ and $\mathcal{B}$ both with n bins. Let $w_{\mathcal{A}} = \{1, 2, 1, 5\}$, and $w_{\mathcal{B}} = \{1, 1, 2, 5\}$ be two weight vectors (sequences of ball weights). Notice that $w_{\mathcal{B}}$ is a monotonically increasing sequence while $w_{\mathcal{A}}$ is not. Note that after allocating the first three balls, $\mathcal{B}$ certainly majorises $\mathcal{A}$. Since the last ball (with weight 5) is bigger than the loads of all bins in both $\mathcal{A}$ and $\mathcal{B}$ after allocating the first three balls, by Lemma 4 the expected maximum load after allocating $w_{\mathcal{A}}$ is bigger than that after allocating $w_{\mathcal{B}}$.

Many Small Balls. Another natural question to ask is the one we answered in Corollary 2 for the single-choice game. Is it better to allocate a large number of small balls compared to a smaller number of large balls with the same total weight? The next example shows again that the majorisation relation is not always maintained in this case.

Observation 5. *Let* $\mathcal{A} = (k+1, k, \ldots, k)$ *and* $\mathcal{B} = (k+0.5, k+0.5, k, \ldots, k)$ *denote two load vectors. After the allocation of one more ball with weight* 1 *the expected maximum load in* $\mathcal{B}$ *is bigger than in* $\mathcal{A}$*.*

Proof. It is easy to check (simply enumerate all cases) that the expected load in $\mathcal{A}$ is $k + 1 + \frac{1}{n^2}$ and that in $\mathcal{B}$ is $k + 1 + \frac{2}{n^2}$.

We emphasize again that the initial majorisation relation is no longer preserved during the allocation. However, we still conjecture that in "most" cases the allocation of a large number of small balls is majorised by the one of a smaller number of large balls with the same total weight, but so far we have been unable to show formal results. The full version of this paper contains empirical results obtained by computer simulations.

References

1. Yossi Azar, Andrei Z. Broder, Anna R. Karlin, Eli Upfal: Balanced Allocations. *SIAM J. Computing* **29** (1999), pp. 180–200.
2. Petra Berenbrink, Artur Czumaj, Angelika Steger, Berthold Vöcking: Balanced Allocations: The Heavily Loaded Case. *Proc. of the 30th Annual ACM Symposium on Theory of Computing (STOC 2000)*, pp. 745–754.
3. Russ Bubley, Martin E. Dyer: Path Coupling: A Technique for Proving Rapid Mixing in Markov Chains. *Proc. of the 38th Annual Symposium on Foundations of Computer Science (FOCS 1997)*, pp. 223–231.
4. Petra Berenbrink: Randomized Allocation of Independent Tasks. University of Paderborn (2000).
5. Petra Berenbrink, Friedhelm Meyer auf der Heide, Klaus Schröder: Allocating Weighted Jobs in Parallel. *Theory of Computing Systems* **32** (1999), pp. 281–300.
6. Chang Cheng-Shang: A New Ordering for Stochastic Majorization: Theory and Applications. *Advances in Applied Probability* **24** (1992), pp. 604–634.
7. Artur Czumaj: Recovery Time of Dynamic Allocation Processes. *Theory of Computing Systems* **33** (2000), pp. 465–487.
8. Artur Czumaj, Volker Stemann: Randomized Allocation Processes. *Random Structures and Algorithms* **18** (2001), pp. 297–331.
9. Artur Czumaj, Chris Rily, Christian Scheideler: Perfectly Balanced Allocation. *Proc. of the 7th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM 2003)*, pp. 240–251.
10. Richard M. Karp, Michael Luby, Friedhelm Meyer auf der Heide: Efficient PRAM Simulation on a Distributed Memory Machine. *Proc. of the 22nd Annual ACM Symposium on Theory of Computing (STOC 1992)*, pp. 318–326.
11. Elias Koutsoupias, Marios Mavronicolas, Paul G. Spirakis: Approximate Equilibria and Ball Fusion. *Theory of Computing Systems* **36** (2003), pp. 683–693.
12. Albert W. Marshall, Ingram Olkin: *Inequalities: Theory of Majorization and Its Applications*. Academic Press, 1979.
13. Michael Mitzenmacher, Andrea W. Richa, Ramesh Sitaraman: The Power of Two Random Choices: A Survey of Techniques and Results. *Handbook of Randomized Computing*, 2000.

14. Michael Mitzenmacher, Balaji Prabhakar, Devarat Shah: Load Balancing with Memory. *Proc. of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002)*, pp. 799–808.
15. Rajeev Motwani, Prabhakar Raghavan: *Randomized Algorithms*. Cambridge University Press, 1995.
16. Alistair Sinclair: *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Birkhäuser, Boston, MA, 1993.
17. Volker Stemann: Parallel Balanced Allocations. *Proc. of the 8th ACM Symposium on Parallel Algorithms and Architectures (SPAA 1996)*, pp. 261–269.
18. Peter Sanders, Sebastian Egner, Jan H. M. Korst: Fast Concurrent Access to Parallel Disks. *Algorithmica* **35** (2003), pp. 21–55.
19. Berthold Vöcking: How Asymmetry Helps Load Balancing. *Proc. of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1999)*, pp. 131-140.

Computing Minimal Multi-homogeneous Bézout Numbers Is Hard

Gregorio Malajovich[1,⋆] and Klaus Meer[2,⋆⋆]

[1] Departamento de Matemática Aplicada,
Universidade Federal do Rio de Janeiro,
Caixa Postal 68530, CEP 21945-970, Rio de Janeiro, RJ, Brasil
gregorio@ufrj.br
[2] Department of Mathematics and Computer Science,
University of Southern Denmark, Campusvej 55,
DK-5230, Odense M, Denmark
meer@imada.sdu.dk

Abstract. Computing good estimates for the number of solutions of a polynomial system is of great importance in many areas such as computational geometry, algebraic geometry, mechanical engineering, to mention a few. One prominent and frequently used example of such a bound is the multi-homogeneous Bézout number. It provides a bound for the number of solutions of a system of multi-homogeneous polynomial equations, in a suitable product of projective spaces. Given an arbitrary, not necessarily multi-homogeneous system, one can ask for the optimal multi-homogenization that would minimize the Bézout number. In this paper, it is proved that the problem of computing, or even estimating the optimal multi-homogeneous Bézout number is **NP**-hard. In terms of approximation theory for combinatorial optimization, the problem of computing the best multi-homogeneous structure does not belong to **APX**, unless $\mathbf{P} = \mathbf{NP}$. As a consequence, polynomial time algorithms for estimating the minimal multi-homogeneous Bézout number up to a fixed factor cannot exist even in a randomized setting, unless $\mathbf{BPP} \supseteq \mathbf{NP}$.

1 Introduction

Many important algorithmical problems can be phrased as problems about polynomial systems. To mention a few, it is easy to reduce **NP**-complete or **NP**-hard problems such as SAT, the Traveling Salesman problem, Integer Programming

⋆ G.M. is partially supported by CNPq (Brasil). This work was done while visiting Syddansk Universitet at Odense, thanks to the generous support of the Villum Kann Rasmussen Fond.

⋆⋆ K.M. was partially supported by EU Network of Excellence PASCAL Pattern Analysis, Statistical Modelling and Computational Learning and by SNF (Danish Natural Science Research Council).

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 244–255, 2005.

(and thus all other **NP** problems as well) to the question whether certain polynomial systems have a common zero.[1] Other examples include computational geometry (f.e. robot motion planning), computer algebra and problems in engineering like the design of certain mechanisms [10, 13].

In analyzing such problems one of the most important questions is to determine the structure of the zero set of a polynomial system. This structure usually has tremendous impact on the complexity of all kind of related algorithms for such systems. In particular, if the solution variety is 0-dimensional the number of zeros is an important quantity for many algorithms. This holds for the two major families of algorithms for polynomial systems, namely symbolic and numerical methods. For example, the very successfully used path-following methods for numerically solving polynomial systems heavily rely on good estimates for the number of solutions; such estimates determine how many paths to follow and thus influence the complexity of such algorithms. It is therefore crucial to get as tight as possible bounds for the number of zeros.

The best-known example giving such an estimate certainly is the Fundamental Theorem of Algebra. It was generalized to multivariate polynomial systems at the end of the 18th century by Etienne Bézout. The Bézout number bounds the number of (isolated) complex zeros of a polynomial $f : \mathbb{C}^n \mapsto \mathbb{C}^n$ from above by the product of the degrees of the involved polynomials. However, in many cases this estimate is far from optimal. A well known example is given by the eigenvalue problem: Given a $n \times n$ matrix M, find the eigenpairs $(\lambda, u) \in \mathbb{C} \times \mathbb{C}^n$ such that $Mu - \lambda u = 0$. If we equate u_n to 1, the classical Bézout number becomes 2^{n-1}, though of course only n solutions exist.

The *multi-homogeneous Bézout numbers* provide sharper bounds on the number of isolated solutions of a system of equations, in a suitable product of projective spaces. The multi-homogeneous Bézout bounds depend on the choice of a *multi-homogeneous structure*, that is of a partition of the variables (λ, u) into several groups.

In the eigenvalue example, the eigenvector u is defined up to a multiplicative constant, so it makes sense to define it as an element of $\mathbb{P}^{n-1}$. With respect to the eigenvalue λ, we need to introduce a homogenizing variable. We therefore rewrite the equation as: $\lambda_0 Mu - \lambda_1 u = 0$, and $\lambda = \lambda_1/\lambda_0$. Now the pair $(\lambda_0 : \lambda_1)$ is an element of $\mathbb{P}^1$. The multi-homogeneous Bézout number for this system is precisely n.

Better bounds on the root number are known, such as Kushnirenko's [6] or Bernstein's [3]. However, interest in computing the multi-homogeneous Bézout number stems from the fact that hardness results are known for those sharper bounds. Another reason of interest is that in many cases, a natural multi-homogeneous structure is known or may be found with some additional human work. An application of multi-homogeneous Bézout bounds outside the realm of

[1] Note, however, that many of these reductions lead to overdetermined systems; below, we are mainly interested in generic systems.

algebraic equation solving is discussed in [4], where the number of roots is used to bound geometrical quantities such as volume and curvature.

In this paper, we consider the following problem. *Let $n \in \mathbb{N}$ and a finite $A \subset \mathbb{N}^n$ be given as input. Find the minimal multi-homogeneous Bézout number, among all choices of a multi-homogeneous structure for a polynomial system with support A.*

Geometrically, this minimal Bézout number is an upper bound for the number of isolated roots of the system in $\mathbb{C}^n$.

The main result in this paper (restated formally in section 2.1 below) is:

Theorem 1. *There cannot possibly exist a polynomial time algorithm to approximate the minimal multi-homogeneous Bézout number up to any fixed factor, unless* $\mathbf{P} = \mathbf{NP}$.

This means that computing or even approximating the minimal Bézout number up to a fixed factor is **NP**-hard. In terms of the hierarchy of approximation classes (see [2]) the minimal multi-homogeneous Bézout number does not belong to the class **APX** unless $\mathbf{P} = \mathbf{NP}$.

Motivated by what is known on volume approximation one could ask whether allowing for randomized algorithms would be of any improvement. By standard arguments the theorem implies

Corollary 1. *There cannot possibly exist a randomized polynomial time algorithm to approximate the minimal multi-homogeneous Bézout number up to any fixed factor, with probability of failure $\epsilon < 1/4$, unless* $\mathbf{BPP} \supseteq \mathbf{NP}$.

2 Background and Statement of Main Results

2.1 Bézout Numbers

A system $f(z)$ of n polynomial equations with support $(A_1, \ldots, A_n)$, each $A_i \subset \mathbb{N}^n$, is a system of the form:

$$\begin{cases} f_1(z) = \sum_{\boldsymbol{\alpha} \in A_1} f_{1\boldsymbol{\alpha}} z_1^{\alpha_1} z_2^{\alpha_2} \cdots z_n^{\alpha_n} \\ \qquad \vdots \\ f_n(z) = \sum_{\boldsymbol{\alpha} \in A_n} f_{n\boldsymbol{\alpha}} z_1^{\alpha_1} z_2^{\alpha_2} \cdots z_n^{\alpha_n} \ , \end{cases} \tag{1}$$

where the coefficients $f_{i\boldsymbol{\alpha}}$ are non-zero complex numbers. (Note that we are interested in systems with the same number of variables and polynomials).

A *multi-homogeneous structure* is given by a partition of $\{1, \ldots, n\}$ into (say) k sets $I_1, \ldots, I_k$. Then for each set I_j, we consider the group of variables $Z_j = \{z_i : i \in I_j\}$.

The degree of f_i in the group of variables Z_j is

$$d_{ij} \stackrel{\text{def}}{=} \max_{\boldsymbol{\alpha} \in A_i} \sum_{l \in I_j} \alpha_l \ .$$

When for some j, for all i, the maximum d_{ij} is attained for all $\boldsymbol{\alpha} \in A_i$, we say that (1) is homogeneous in the variables Z_j. The dimension of the projective space associated to Z_j is:

$$a_j \stackrel{\text{def}}{=} \begin{cases} \#I_j - 1 \text{ if (1) is homogeneous in } Z_j, \text{ and} \\ \#I_j \qquad \text{otherwise} \end{cases} .$$

We assume that $n = \sum_{j=1}^{k} a_j$. Otherwise, we would have underdetermined ($n < \sum_{j=1}^{k} a_j$) systems (which we do not treat in this paper) or overdetermined ($n > \sum_{j=1}^{k} a_j$) polynomial systems (for which multi-homogeneous Bézout numbers have no meaning).

The multi-homogeneous Bézout number $\text{Béz}(A_1, \ldots, A_n; I_1, \ldots, I_k)$ is the coefficient of $\prod_{j=1}^{k} \zeta_j^{a_j}$ in the formal expression $\prod_{i=1}^{n} \sum_{j=1}^{k} d_{ij}\zeta_j$ (see [11, 7, 12]). Here, the ζ_j's are new indeterminates associated to each j-block of variables. The multi-homogeneous Bézout number bounds the maximal number of isolated roots of (1) in $\mathbb{P}^{a_1} \times \cdots \times \mathbb{P}^{a_k}$. Therefore it also bounds the number of *finite* roots of (1), i.e. the roots in $\mathbb{C}^n$.

In the particular case where $A = A_1 = \cdots = A_n$ (i.e. all polynomials have the same support) there is a simpler expression for the multi-homogeneous Bézout number $\text{Béz}(A; I_1, \ldots, I_k) \stackrel{\text{def}}{=} \text{Béz}(A_1, \ldots, A_n; I_1, \ldots, I_k)$, namely:

$$\text{Béz}(A; I_1, \ldots, I_k) = \binom{n}{a_1\ a_2\ \cdots\ a_k} \prod_{j=1}^{k} d_j^{a_j} , \tag{2}$$

where $d_j = d_{ij}$ (equal for each i) and the multinomial coefficient

$$\binom{n}{a_1\ a_2\ \cdots\ a_k} \stackrel{\text{def}}{=} \frac{n!}{a_1!\ a_2!\ \cdots\ a_k!}$$

is the coefficient of $\prod_{j=1}^{k} \zeta_j^{a_k}$ in $(\zeta_1 + \cdots + \zeta_k)^n$ (recall that $n = \sum_{j=1}^{k} a_j$).

Heuristics for computing a suitable multi-homogeneous structure $(I_1, \ldots, I_k)$ given $A_1, \ldots, A_n$ are discussed in [8, 9]. Surprisingly enough, there seems to be no theoretical hardness results available on the complexity of computing the minimal Bézout number. It was conjectured in [8–p.78] that computing the minimal multi-homogeneous Bézout number is **NP**-hard.

Even, no polynomial time algorithm for computing the multi-homogeneous Bézout number *given a multi-homogeneous structure* seems to be known (see [9, p.240]).

This is why in this paper, we restrict ourselves to the case $A = A_1 = \cdots = A_n$. This is a particular subset of the general case, and any hardness result for this particular subset implies the same hardness result in the general case.

More formally, we adopt the Turing model of computation and we consider the function:

$$\text{Béz} : n, k, A, I_1, \ldots, I_k \mapsto \text{Béz}(A; I_1, \ldots, I_k) ,$$

where all integers are in binary representation, and A is a list of n-tuples $(\alpha_1, \ldots, \alpha_n)$, and each I_j is a list of its elements. In particular, the input size is

bounded below by $n\#A$ and by $\max_{\boldsymbol{\alpha},i}\lceil\log_2 \alpha_i\rceil$. Therefore, $\text{Béz}(A; I_1, \cdots, I_k)$ can be computed in polynomial time by a straight-forward application of formula (2). As a matter of fact, it can be computed in time polynomial in the size of A.

Problem 1. Given n and A, compute $\min_{\boldsymbol{I}} \text{Béz}(A; \boldsymbol{I})$, where $\boldsymbol{I} = (I_1, \ldots, I_k)$ ranges over all the partitions of $\{1, \ldots, n\}$.

Problem 2. Let $C > 1$ be fixed. Given n and A, compute some B such that $BC^{-1} < \min \text{Béz}(A; \boldsymbol{I}) < BC$. Again, $\boldsymbol{I} = (I_1, \ldots, I_k)$ ranges over all the partitions of $\{1, \ldots, n\}$.

Important Remark: The different multi-homogeneous Bézout numbers do only depend on the support of the system (i.e. on the monomials with non-zero coefficients) and on the chosen partition. They do not depend on the actual values of the coefficients $f_{i\boldsymbol{\alpha}}$. Thus, throughout the paper we consider only n and A as input, but not the values $f_{i\boldsymbol{\alpha}} \neq 0$.

In the problems above, we are not asking for the actual partition. Thus, our results are even stronger. In the statements below that refer to approximation classes like APX computation of a feasible solution is included in the problem.

Theorem 1 (Restated). *Problem 2 is* **NP***-hard.*

This is actually stronger than the conjecture by Li and Bai [8], that corresponds to the first part of the following immediate corollary:

Corollary 2. *Problem 1 is* **NP***-hard. Moreover, it does not belong to* **APX***, unless* $\mathbf{P} = \mathbf{NP}$*.*

Corollary 1 (Restated). *There is no* $\epsilon < 1/2$ *and no probabilistic machine solving Problem 2 with probability* $1 - \epsilon$*, unless* $\mathbf{BPP} \supseteq \mathbf{NP}$*.*

Concerning other bounds on the number of roots let us briefly mention that Kushnirenko [6] bounds the number of isolated solutions of (1) (for supports $A_i = A$) in $(\mathbb{C}^*)^n$ by $n! \cdot \text{Vol}(\text{Conv}A)$, where $\text{Conv}A$ is the smallest convex polytope containing all the points of A. This bound is sharper than the Bézout bound, but the known hardness results are far more dramatic: In [5], Khachiyan proved that computing the volume of a polytope given by a set of vertices is $\#\mathbf{P}$-hard. For the huge amount of literature on algorithms for approximating the volume of a convex body in different settings we just refer to [14] as a starting point for further reading.

3 Proof of the Main Theorems

3.1 From Graph Theory to Systems of Equations

It is well known that the Graph 3-Coloring Problem is **NP**-hard. We will need to consider an equivalent formulation of the Graph 3-coloring problem.

Recall that the cartesian product of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the graph $G_1 \times G_2 = (V_1 \times V_2, E)$ with $((v_1, v_2), (v_1', v_2')) \in E$ if and only if $v_1 = v_1'$ and $(v_2, v_2') \in E_2$ or $v_2 = v_2'$ and $(v_1, v_1') \in E_1$.

Also, let K_3 denote the complete graph with 3 vertices.

Lemma 1. *The graph G admits a 3-coloring if and only if the graph $G \times K_3$ admits a 3-coloring $\boldsymbol{I} = (I_1, I_2, I_3)$ with $\#I_1 = \#I_2 = \#I_3 = |G|$ (where I_1, I_2, I_3 are subsets of vertices colored with each of the three colors).* □

To each graph $H = (V, E)$ we will associate two spaces of polynomial systems. Each of those spaces is characterized by a support set $A = A(H)$ (resp. $A(H)^l$) to be constructed and corresponds to the space of polynomials of the form (1) with complex coefficients. Of particular interest will be graphs of the form $H = G \times K_3$.

We start by identifying the set V of vertices of H to the set $\{1, \ldots, m\}$. Let K_s denote the complete graph of size s, i.e. the graph with s vertices all of them pairwise connected by edges.

To each copy of K_s, $s = 0, \ldots, 3$ that can be embedded as a subgraph of H (say the subgraph generated by $\{v_1, \cdots, v_s\}$) we associate the monomial $z_{v_1} z_{v_2} \cdots z_{v_s}$ (the empty graph K_0 corresponds to the constant monomial). Then

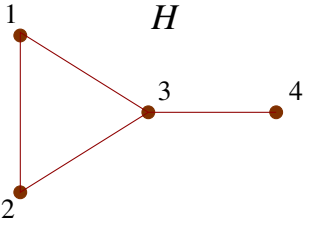


Fig. 1. In this example, $A(H) = \{(0,0,0,0), (1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1), (1,1,0,0), (1,0,1,0), (0,1,1,0), (0,0,1,1), (1,1,1,0)\}$. A possible polynomial with that support would be $1 + v_1 + v_2 + v_3 + v_4 + v_1v_2 + v_1v_3 + v_2v_3 + v_3v_4 + v_1v_2v_3$

we consider the linear space generated by all those monomials (Figure 1). Therefore, the support $A(H)$ is the set of all $e_{v_1} + \cdots + e_{v_s} \subset \mathbb{N}^m$ such that $0 \leq s \leq 3$ and $\{v_1, \ldots, v_s\}$ induces a copy of K_s as a subgraph of H. Here, e_i denotes the i-th vector of the canonical basis of $\mathbb{R}^n$.

Given a set A, we denote by A^l the l-fold cartesian product of A.

The two spaces of polynomial systems associated to a graph H will be the polynomial systems with support $A(H)$ and $A(H)^l$.

Remark that none of the two classes of systems above is homogeneous in any possible group of variables (because we introduced a constant monomial). Therefore, in the calculation of the Bézout number for a partition $\boldsymbol{I}$, we can set $a_j = \#I_j$. The following is straightforward:

Lemma 2. *Let l be fixed. Then, there is a polynomial time algorithm to compute $A(H)$ and $A(H)^l$, given H.* □

3.2 A Gap Between Bézout Numbers

In case the graph H admits a 3-coloring $\boldsymbol{I} = (I_1, I_2, I_3)$, any corresponding polynomial system is always trilinear (linear in each set of variables). If moreover H is of the form $H = G \times K_3$ with $|G| = n$, the cardinality of the I_j is always n, and formula (2) becomes:

$$\text{Béz}(A(G \times K_3); \boldsymbol{I}) = \binom{3n}{n\ n\ n}.$$

The crucial step in the proof of Theorem 1 is to show that this number is $\geq \frac{4}{3}\binom{3n}{n\ n\ n}$ unless $k = 3$ and $\boldsymbol{I}$ is a 3-coloring of $G \times K_3$.

In order to do that, we introduce the following cleaner abstraction for the Bézout number: if $k \in \mathbb{N}$ and $\mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{N}^k$ are such that $\sum_{j=1}^k a_j = 3n$, we set

$$B(\mathbf{a}) \stackrel{\text{def}}{=} \binom{3n}{a_1\ a_2\ \cdots\ a_k} \prod_{j=1}^{k} \left\lceil \frac{a_j}{n} \right\rceil^{a_j}.$$

Lemma 3. *If $H = G \times K_3$ and $\boldsymbol{I} = (I_1, \ldots, I_k)$ is a partition of the set $\{1, \ldots, 3n\}$ of vertices of H, then $\text{Béz}(A(H); \boldsymbol{I}) \geq B(\mathbf{a})$ with $a_j = \#I_j$.* □

The main step towards establishing the "gap" is the following Proposition:

Proposition 1. *Let $n, k \in \mathbb{N}$ and let $a_1 \geq a_2 \geq \cdots \geq a_k \geq 1$ be such that $\sum_{j=1}^k a_j = 3n$. Then, either $k = 3$ and $a_1 = a_2 = a_3 = n$, or $B(\mathbf{a}) \geq \frac{4}{3}B(n, n, n)$. Moreover, this bound is sharp.*

The proof of Proposition 1 is postponed to section 4.

Putting it all together,

Lemma 4. *Let G be a graph and $n = |G|$. Then, either*

$$\min_{\boldsymbol{I}} \text{Béz}(A(G \times K_3); \boldsymbol{I}) = \binom{3n}{n\ n\ n} \text{ or } \min_{\boldsymbol{I}} \text{Béz}(A(G \times K_3); \boldsymbol{I}) \geq \frac{4}{3}\binom{3n}{n\ n\ n},$$

depending on whether G admits a 3-coloring or not.

Proof. According to Lemma 1, G admits a 3-coloring if and only if $G \times K_3$ admits a 3-coloring. If $\boldsymbol{I} = (I_1, I_2, I_3)$ is a 3-coloring of $G \times K_3$, then

$$\text{Béz}(A(G \times K_3); \boldsymbol{I}) = \binom{3n}{n\ n\ n}.$$

If $\boldsymbol{I} = (I_1, \ldots, I_k)$ is not a 3-coloring of $G \times K_3$, then we distinguish two cases. We set $a_j = \#I_j$.

Case 1: a $= (n, n, n)$ and hence $k = 3$. Then the degree in at least one group of variables is ≥ 2, and

$$\text{Béz}(A(G \times K_3); \boldsymbol{I}) \geq 2^n \binom{3n}{n\ n\ n}.$$

Case 2: a $\neq (n, n, n)$. Then

$$\text{Béz}(A(G \times K_3); \boldsymbol{I}) \geq B(a_1, \ldots, a_k) \geq \frac{4}{3} \binom{3n}{n\ n\ n}$$

by Lemma 3 and Proposition 1. In both cases,

$$\min_{\boldsymbol{I}} \text{Béz}(A(G \times K_3), \boldsymbol{I}) \geq \frac{4}{3} \binom{3n}{n\ n\ n}. \qquad \square$$

3.3 Improving the Gap

In order to obtain a proof valid for any constant gap C the idea is to increase the gap by considering several copies of a polynomial system, but each copy in a new set of variables. This idea works out because of the special multiplicative structure of the multi-homogeneous Bézout number. We will need:

Proposition 2. *Let $m, l \in \mathbb{N}$. Let $A \subset \mathbb{N}^m$ be finite and assume that $0 \in A$. Then,*

$$\min_{\boldsymbol{J}} \text{Béz}(A^l; \boldsymbol{J}) = \binom{lm}{m\ m\ \cdots\ m} \left(\min_{\boldsymbol{I}} \text{Béz}(A; \boldsymbol{I}) \right)^l .$$

Proof. 1. Let $\boldsymbol{I} = (I_1, \cdots, I_k)$ be the partition of $\{1, \ldots, m\}$ where the minimal Bézout number for A is attained. This induces a partition $\boldsymbol{J} = (J_{js})_{1 \leq j \leq k, 1 \leq s \leq l}$ of $\{1, \ldots, m\} \times \{1, \ldots, l\}$, given by $J_{js} = I_j \times \{s\}$. Identifying each pair (i, s) with $i + ms$, the J_{js} are also a partition of $\{1, \ldots, lm\}$. By construction of A^l, the degree d_{js} in the variables corresponding to J_{js} is equal to the degree d_j of the variables I_j in A. The systems corresponding to A and A^l cannot be homogeneous for any partition, since $0 \in A$ and $0 \in A^l$. Then we have $a_j = \#I_j = a_{js}$ for any s. Therefore,

$$\begin{aligned}
\min_{\boldsymbol{K}} \text{Béz}(A^l, \boldsymbol{K}) \leq \text{Béz}(A^l, \boldsymbol{J}) &= \binom{\underbrace{a_1\ \cdots\ a_1}_{l \text{ times}}\ \cdots\ \underbrace{a_k\ \cdots\ a_k}_{l \text{ times}}}{lm} \prod_{s=1}^{l} \prod_{j=1}^{k} d_j^{a_j} \\
&= \binom{lm}{m\ m\ \cdots\ m} \left(\binom{m}{a_1\ a_2\ \cdots\ a_k} \prod_{j=1}^{k} d_j^{a_j} \right)^l \\
&= \binom{lm}{m\ m\ \cdots\ m} \left(\min_{\boldsymbol{I}} \text{Béz}(A; \boldsymbol{I}) \right)^l .
\end{aligned}$$

2. Now, suppose that the minimal Bézout number for A^l is attained for a partition $\boldsymbol{J} = (J_1, \cdots, J_r)$. We claim that each J_t fits into exactly one of the l sets $\{1, \ldots, m\} \times \{s\}$.

Suppose this is not the case. Assume without loss of generality that J_1 splits into $K \subset \{1,\dots,m\} \times \{1\}$ and $L \subset \{1,\dots,m\} \times \{2,\dots,l\}$, both K and L nonempty. If d_K denotes the degree in the K-variables and d_L the degree in the L variables, then $d_1 = d_K + d_L$. Also, $a_1 = a_K + a_L$ where a_K is the size of K and a_L is the size of L. The multi-homogeneous Bézout number corresponding to the partition $\boldsymbol{J}' = (K, L, J_2, \cdots, J_r)$ is:

$$\text{Béz}(A^l; \boldsymbol{J}') = \binom{3lm}{a_K\ a_L\ a_2\ \cdots\ a_r} d_K^{a_K} d_L^{a_L} \prod_{j=2}^{r} d_j^{a_j} .$$

Therefore,

$$\frac{\text{Béz}(A^l; \boldsymbol{J}')}{\text{Béz}(A^l, \boldsymbol{J})} = \frac{\binom{a_1}{a_K} d_K^{a_K} d_L^{a_L}}{(d_K + d_L)^{a_1}} < 1$$

and the Bézout number was not minimal, thus establishing the claim.

3. Denote by $\boldsymbol{J} = \cup_{s=1}^{l} \boldsymbol{J}^{(s)}$ the partition minimizing the Bézout number corresponding to A^l. In the notation above, we assume that $\boldsymbol{J}^{(s)}$ is a partition of $\{1,\dots,m\} \times \{s\}$.

In that case,

$$\begin{aligned} \text{Béz}(A^l; \boldsymbol{J}) &= \binom{lm}{m\ m\ \cdots\ m} \prod_{s=1}^{l} \left(\binom{m}{a_1^{(s)}\ \cdots\ a_k^{(s)}} \prod_{j=1}^{k} (d_j^{(s)})^{a_j^{(s)}} \right) \\ &= \binom{lm}{m\ m\ \cdots\ m} \prod_{s=1}^{l} \text{Béz}(A, \boldsymbol{J}^{(s)}) \geq \binom{lm}{m\ m\ \cdots\ m} \left(\min_{\boldsymbol{I}} \text{Béz}(A; \boldsymbol{I}) \right)^l . \end{aligned}$$

□

Combining Lemma 4 and Proposition 2, we established that:

Lemma 5. *Let G be a graph and $n = |G|$. Let $l \in \mathbb{N}$. If G admits a 3-coloring, then*

$$\min_{\boldsymbol{J}} \text{Béz}(A(G \times K_3)^l, \boldsymbol{J}) = \binom{3nl}{3n\ 3n\ \cdots\ 3n} \binom{3n}{n\ n\ n}^l .$$

Otherwise,

$$\min_{\boldsymbol{J}} \text{Béz}(A(G \times K_3)^l, \boldsymbol{J}) \geq \left(\frac{4}{3}\right)^l \binom{3nl}{3n\ 3n\ \cdots\ 3n} \binom{3n}{n\ n\ n}^l . \qquad \square$$

Proof (of Theorem 1). Assume that `ApproxBéz` is a deterministic, polynomial time algorithm for solving problem 2, i.e., for estimating the Bézout number up to a factor of C. Then we could decide Graph 3-coloring in polynomial time as follows:

Let $l = \left\lceil \frac{\log C}{2 \log 4/3} \right\rceil$ and apply the potential approximation algorithm to $\left(A(G \times K_3)^l\right)$. Compute

$$\rho \leftarrow \frac{\textsc{ApproxBéz}\left(A(G \times K_3)^l\right)}{\binom{3nl}{3n\ 3n\ \cdots\ 3n}\binom{3n}{n\ n\ n}^l} .$$

By our choice of the constant l, $\sqrt{C} \leq (4/3)^l$. Therefore, Lemma 5 asserts that the output of our algorithm is correct.

The bit-size of the numbers that occur when computing the denominator of line 2 are bounded above by $O(3nl \log(3nl))$. The size of the graph $G \times K_3$ is $O(n)$, and Lemma 2 says that A^l can be computed in polynomial time. It follows that the above algorithm runs in polynomial time. Since Graph 3-coloring is **NP**-complete, we deduce that $\mathbf{P} = \mathbf{NP}$. □

Proof (of Corollary 1). Assume now that `ApproxBéz` is a probabilistic polynomial time algorithm for solving problem 2, which returns a correct result with probability $1 - \epsilon$, $\epsilon < 1/4$. Then the algorithm will return the correct answer for the Graph 3-coloring Problem, with probability at least $1 - \epsilon$. This implies that Graph 3-coloring is actually in **BPP**. □

4 Proof of Proposition

Lemma 6. *Let $x, n \in \mathbb{N}$. Then, $\left(\left\lceil \frac{x}{n} \right\rceil \frac{n}{x}\right)^x \geq 1 + \left((n-x) \mod n\right)$.*
In particular, the left-hand side is ≥ 2 whenever $n \not| x$, and is always ≥ 1.

Next, we will make use of the Stirling Formula [1, (6.1.38)]:

$$x! = \sqrt{2\pi}\ x^{x+\frac{1}{2}}\ e^{-x+\frac{\theta(x)}{12x}} , \tag{3}$$

where $0 < \theta(x) < 1$.

Proof (of Proposition 1). The ratio between $B(\mathbf{a})$ and $B(n,n,n)$ is:

$$\frac{B(\mathbf{a})}{B(n,n,n)} = \prod_{j=1}^{k} \left\lceil \frac{a_j}{n} \right\rceil^{a_j} \ \frac{n!\ n!\ n!}{a_1!\ a_2!\ \cdots\ a_k!} .$$

From Stirling formula (3) it follows immediately that:

$$\frac{B(\mathbf{a})}{B(n,n,n)} = \sqrt{2\pi}^{3-k} \prod_{j=1}^{k} \left\lceil \frac{a_j}{n} \right\rceil^{a_j} \ \frac{n^{3n+\frac{3}{2}}}{\prod_{j=1}^{k} a_j^{a_j+\frac{1}{2}}}\ e^{\frac{\theta(n)}{4n} - \sum \frac{\theta(a_j)}{12a_j}} . \tag{4}$$

Now we distinguish the cases $k = 1$, $k = 2$, and $k \geq 3$. The first two cases are easy:

Case 1: If $k = 1$, then $a_1 = 3n$ and (4) becomes:

$$\frac{B(\mathbf{a})}{B(n,n,n)} = 2\pi \ \frac{n}{\sqrt{3}} \ e^{\frac{\theta(n)}{4n} - \frac{\theta(3n)}{36n}}$$

which is bounded below by $\frac{2\pi}{\sqrt{3}} e^{-1/36} \simeq 3.528218766$.

Case 2: If $k = 2$, Lemma 6 implies that $\frac{B(\mathbf{a})}{B(n,n,n)} \geq \sqrt{2\pi} \ \frac{n^{\frac{3}{2}}}{\sqrt{a_1 \ a_2}} \ e^{-1/6}$.

Since $\sqrt{a_1 \ a_2} \leq \frac{a_1+a_2}{2} = \frac{3n}{2}$, we obtain: $\frac{B(\mathbf{a})}{B(n,n,n)} \geq \frac{2}{3}\sqrt{2\pi} e^{-1/6} \simeq 1.41454$.

Case 3: Let $k \geq 3$. If $a_3 = n$, then $k = 3$ and $a_1 = a_2 = a_3 = n$, so there is nothing to prove. Therefore, we assume from now on that $a_3 < n$.

We separate the right-hand side of (4) into two products, the first for $j = 1, 2, 3$ and the second for $j \geq 4$. Equation (4) becomes now:

$$\begin{aligned} \frac{B(\mathbf{a})}{B(n,n,n)} = &\left(\prod_{j=1}^{3} \left(\left\lceil \frac{a_j}{n} \right\rceil \frac{n}{a_j} \right)^{a_j} \ \frac{n^{\frac{3}{2}}}{\sqrt{a_1 a_2 a_3}} \ e^{\frac{\theta(n)}{4n} - \sum_{j=1}^{3} \frac{\theta(a_j)}{12 a_j}} \right) \\ &\left(\sqrt{2\pi}^{3-k} \ \prod_{j=4}^{k} \frac{n^{a_j}}{a_j^{a_j+\frac{1}{2}}} \ e^{-\sum_{j=4}^{k} \frac{\theta(a_j)}{12 a_j}} \right) \end{aligned} \tag{5}$$

using the fact that $a_j < n$ for $j \geq 4$. In case $k = 3$, the second factor in equation (5) above is equal to one. Since $a_3 < n$, $n \not| \ a_3$ and Lemma 6 implies that for $a_3 < n$

$$\prod_{j=1}^{3} \left(\left\lceil \frac{a_j}{n} \right\rceil \frac{n}{a_j} \right)^{a_j} \geq 2 \ .$$

Moreover, $\sqrt[3]{a_1 a_2 a_3} \leq (a_1+a_2+a_3)/3 \leq n$, so the first factor of the right-hand side of (5) can be bounded below by

$$\prod_{j=1}^{k} \left(\left\lceil \frac{a_j}{n} \right\rceil \frac{n}{a_j} \right)^{a_j} \ \frac{n^{\frac{3}{2}}}{\sqrt{a_1 a_2 a_3}} \ e^{\frac{\theta(n)}{4n} - \sum_{j=1}^{3} \frac{\theta(a_j)}{12 a_j}} \geq 2e^{-1/4} \simeq 1.557601566 \ .$$

If $k = 3$ we are done. Otherwise, we notice that since the a_j are non-increasing, $a_j \leq \frac{3n}{4}$ for all $j \geq 4$. In order to bound the second factor of (5), we will need the following technical Lemma:

Lemma 7. *Let $n, x \in \mathbb{N}$ and let $x \leq \frac{3n}{4}$. Then, unless $(n, x) \in \{(2,1), (3,2), (4,3), (6,4), (7,5), (8,6)\}$, we have:*

$$\frac{n^x}{\sqrt{2\pi} x^{x+\frac{1}{2}}} e^{-\frac{1}{12x}} > 1 \ .$$

Therefore, unless some of the pairs (n, a_j), $j \geq 4$ belong to the exceptional subset $\{(2,1), (3,2), (4,3), (6,4), (7,5), (8,6)\}$, we have:

$$\frac{B(\mathbf{a})}{B(n,n,n)} \geq 2e^{-\frac{1}{4}} \simeq 1.557601566 \ .$$

Finally, we consider the values of n and $\mathbf{a}$ where some (n, a_j), $j \geq 4$, is in the exceptional subset. Of course, the exceptional set is finite and thus we are in principal done. However, inspecting these values gives as lower bound the ratio $4/3$ which is attained for $n = 2$ and $\mathbf{a} = (3, 1, 1, 1)$. □

References

1. Abramowitz, M. and Stegun, Irene A. (editors): Handbook of mathematical functions with formulas, graphs, and mathematical tables. Dover Publications Inc., New York (1992).
2. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and approximation. Springer-Verlag, Berlin (1999).
3. Bernstein, D. N.: The number of roots of a system of equations (in russian). Funkcional. Anal. i Priložen., vol. 9, nr. 3 (1975)
4. Dedieu, J.P., Malajovich, G., Shub, M.: On the curvature of the central path of linear programming theory, Preprint (2003).
5. Khachiyan, L. G.: The problem of calculating the volume of a polyhedron is enumeratively hard (in russian). Uspekhi Mat. Nauk, vol. 44, nr. 3(267), 179–180 (1989).
6. Kushnirenko, A.G.: Newton Polytopes and the Bézout Theorem. Funct. Anal. Appl., vol. 10, 233–235 (1976).
7. Li, T. Y.: Numerical solution of multivariate polynomial systems by homotopy continuation methods. Acta numerica, Cambridge University Press, vol. 6, 399–436 (1997).
8. Li, T., Bai, F.: Minimizing multi-homogeneous Bézout numbers by a local search method. Math. Comp., vol. 70, nr. 234, 767–787 (2001).
9. Li, T., Lin, Z., Bai, F.: Heuristic methods for computing the minimal multi-homogeneous Bézout number. Appl. Math. Comput., vol. 146, nr. 1, 237–256 (2003).
10. Morgan, A.: Solving polynomial systems using continuation for engineering and scientific problems. Prentice Hall Inc., Englewood Cliffs, NJ (1987).
11. Morgan, A., Sommese, A.: A homotopy for solving general polynomial systems that respects m-homogeneous structures. Appl. Math. Comput., vol. 24, nr.2, 101–113 (1987).
12. Shafarevich, I. R.: Basic algebraic geometry. Springer Study Edition, Springer-Verlag, Berlin (1977).
13. Wampler, C., Morgan, A., Sommese, A.: Numerical continuation methods for solving polynomial systems arising in kinematics. Journal Mechanical Design, vol. 112, 59–68 (1990).
14. Werschulz, A. G., Woźniakowski, H.: What is the complexity of volume calculation?, Journal of Complexity, vol. 18, nr. 2, 660–678 (2002).

Dynamic Complexity Theory Revisited

Volker Weber and Thomas Schwentick

Philipps-Universität Marburg, FB Mathematik und Informatik
{webervo, tick}@informatik.uni-marburg.de

Abstract. Dynamic complexity asks for the effort needed to maintain the information about properties of a structure under operations changing the structure. This paper introduces a refined notion of dynamic problems which takes the initial structure into account. It develops the basic structural complexity notions accordingly. It also shows that the dynamic version of the LOGCFL-complete problem D_2LREACH(acyclic) can be maintained with first-order updates.

1 Introduction

For a set S, the *static decision problem* asks whether a given input I is an element of S. Classical *static complexity theory* studies the inherent computational effort to answer this question. But often one is not interested only once whether $I \in S$ but I undergoes small changes and information about its membership in S should be available after each change. E.g., S could contain all triples (G, s, t), where G is a graph and s and t are nodes such that t is reachable from s in G. Changes could be insertion and deletion of edges. Another typical example is a view in a database with tuple insertions and deletions to the base relations.

Of course, in many cases one can expect that if I' results from I by applying a small change, whether $I' \in S$ might be closely related to whether $I \in S$. In particular, it should often be simpler to *maintain* information about membership in S under small changes than to recompute it from scratch for each new instance. This might involve auxiliary data structures which are updated accordingly.

These considerations are the starting point of *dynamic complexity theory* which was initiated in [16] and [17]. This theory has been focusing on two lines of research, structural results about complexity classes of dynamic problems, including suitable reduction concepts and complete problems and upper bounds for concrete problems. Both lines consider problems with a very low update complexity, especially with updates expressible in first-order logic (or, equivalently, by uniform families of constant-depth, polynomial size circuits with unbounded fan-in, aka AC^0). This class is called DynFO in [17].

In [11], low-level reductions for dynamic problems are defined and a complete problem (CSSCV) for DynFO is presented. Although this was an important step, there remained some steps to be done. First, as the authors pointed out themselves, it is still a challenge to find *natural* complete problems for dynamic

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 256–268, 2005.

classes. Second, in the construction of the complete problem a technical difficulty arose that was caused by the role of the initial structure of a dynamic problem in the setting of [17]. Further, the reductions seem hard to use for more natural problems. Also, the class DynFO itself does not distinguish between problems of quite different static complexity, e.g., between LOGSPACE and PTIME.

Concerning upper bounds, the dynamic versions of many problems inside NL have been shown to be in DynFO [17], e.g., reachability in directed, acyclic graphs or in undirected graphs. The technically most demanding result was that reachability in directed (possibly cyclic) graphs is in DynTC^0, i.e., the corresponding dynamic class with constant-depth threshold circuits [10].

It is natural to ask for the highest complexity of a (static) problem such that its dynamic version allows for first-order updates. The answer to this question has two facets. It was shown in [16, 17] that there are even P-complete problems in DynFO. Nevertheless, these problems are highly artificial, using redundant encodings of the form $w^{|w|}$. Concerning non-redundant problems the complexity-wise highest problems in DynFO known so far are complete for NL.

Contributions. This paper contributes to both lines described above. First, by taking the complexity of the initial instance I into account in a very simple fashion, we define more refined complexity classes and corresponding reduction concepts. More precisely, our classes are of the form $\text{Dyn}(\mathcal{C}, \mathcal{C}')$, where $\mathcal{C}$ is the complexity of computing the auxiliary data structure for the initial input I and $\mathcal{C}'$ is the complexity of the updates. We show that these classes and reductions behave nicely and that the results of [17, 10, 11] translate in a straightforward way. The new classes allow a more precise classification of problems. We show that most of the problems mentioned above are in the respective class $\text{Dyn}(\mathcal{C}, \text{FO})$, where $\mathcal{C}$ is the complexity of the underlying static problem. Nevertheless, optimality w.r.t. the initial input complexity is not automatic, e.g., it is not clear whether the dynamic reachability problem is in $\text{Dyn}(\text{NL}, \text{TC}^0)$.

The technically most difficult result of this paper contributes to the other line of research. It presents a (non-redundant) LOGCFL-complete problem with first-order updates, more precisely, in $\text{Dyn}(\text{LOGCFL}, \text{FO})$.

Related Work. In a series of papers (e.g., [2, 3, 5, 4, 1]) *first-order incremental evaluation systems* have been studied, which are basically the analogue of DynFO for database queries. In [14, 15] SQL was used as update language. There is a huge body of work on algorithms for dynamic problems, e.g., [12, 18] and on *Online Algorithms* [7].

Organization. In Section 2 we define dynamic problems and classes. Some precise upper bounds are given in Section 3. Reductions are addressed in Section 4, complete problems in section 5. Connections to static complexity are focused in Section 6. In Section 7 we exhibit a LOGCFL-complete problem which is in $\text{Dyn}(\text{LOGCFL}, \text{FO})$. We conclude in Section 8.

2 Definitions

In this section, we give the basic definitions on dynamic problems and dynamic complexity classes. These definitions depart considerably from [17]. Intuitively, in our view, a dynamic problem D is induced by a static problem S, i.e., a set of structures and a set of operations. A pair (A, w) consisting of a structure A and a sequence w of operations is in D if the structure resulting from A after applying w is in S. As an example, recall the reachability example from Section 1. Instead of graphs we might consider arbitrary structures, instead of operations `insert` and `delete` we might have other operations as well.

We turn to the formal definitions. We write $\mathrm{STRUC}_n(\tau)$ for the class of structures with n elements over vocabulary τ, e.g., $\mathrm{STRUC}_5(E)$ denotes all graphs with 5 vertices. The universe of a structure is always $[n] := \{0, \ldots, n-1\}$. We only consider vocabularies with relation and constant symbols.

In general, we use *operation symbols* σ from a finite set Σ with an associated arity $\mathrm{arity}(\sigma)$. E.g., in the graph example, $\mathrm{arity}(\texttt{insert}) = \mathrm{arity}(\texttt{delete}) = 2$. An *operation* on a structure $\mathcal{A} \in \mathrm{STRUC}_n(\tau)$ is simply a tuple $\sigma(a_1, \ldots, a_m)$ with $a_i \in [n]$, for $i \leq m$ and $m = \mathrm{arity}(\sigma)$. We denote the set of operations with symbols from Σ over structures from $\mathrm{STRUC}_n(\tau)$ by Σ_n.

The semantics are given by an *update function* g which maps a pair $(A, \sigma(a_1, \ldots, a_m))$ from $\mathrm{STRUC}_n(\tau) \times \Sigma_n$ to a structure $\sigma^g(a_1, \ldots, a_m)(A)$ from $\mathrm{STRUC}_n(\tau)$. We usually write $\sigma(a_1, \ldots, a_m)(A)$ for this structure. For a string $w = w_1 \cdots w_m$ of operations and a structure A let $w(A)$ be $w_m(\cdots(w_1(A)\cdots)$.

Definition 1. Let τ be a vocabulary, S a set of τ-structures and Σ a set of operation symbols. The *dynamic problem* $D(S, \Sigma)$ associated with S and Σ is the set of pairs (A, w), where, for some $n > 0$, $A \in \mathrm{STRUC}_n(\tau)$, $w \in \Sigma_n^*$ and $w(A) \in S$. We call S the *underlying static problem of* D.

The computations we want to model are of the following kind. First, from the input structure A an *auxiliary data structure* B is computed. Afterwards, for each operation w_i this auxiliary structure is updated in order to reflect the changes of A and to allow to find out quickly whether $w_1 \cdots w_i(A)$ is in S. In our dynamic complexity classes we consider the costs for the initial computation and the updates separately. We are mainly interested in problems where the costs for updates are very low. In this case the costs of the initial computation can not be better than the complexity of S itself. More precisely, $\textsc{Dyn}(\mathcal{C}, \mathcal{C}')$ is the class of problems, for which the initial computation can be done within complexity class $\mathcal{C}$ and the updates within $\mathcal{C}'$.

Definition 2. Let τ be a fixed vocabulary and let $\mathcal{C}$ and $\mathcal{C}'$ be complexity classes. Let $\textsc{Dyn}(\mathcal{C}, \mathcal{C}')$ be the class of all dynamic problems $D = D(S, \Sigma)$ satisfying the following conditions:

- There is a vocabulary ρ, a set S' of ρ-structures, and a $\mathcal{C}$-computable function $f : \mathrm{STRUC}[\tau] \to \mathrm{STRUC}[\rho]$ such that
  - $f(A) \in \mathrm{STRUC}_n[\rho]$ for all $A \in \mathrm{STRUC}_n[\tau]$;
  - $f(A) \in S'$ if and only if $A \in S$

– There is a $\mathcal{C}'$-computable function f' mapping tuples $(B, \sigma, a_1, \cdots, a_{\text{arity}(\sigma)})$, where, for some n, B is from $\text{STRUC}_n[\rho]$ and $\sigma(a_1, \cdots, a_{\text{arity}(\sigma)}) \in \Sigma_n$, to structures in $\text{STRUC}_n[\rho]$ such that for each n, each $A \in \text{STRUC}_n[\tau]$ and each operation sequence $w \in \Sigma_n^*$:

$$w(A) \in S \Longleftrightarrow f'(f(A), w) \in S'$$

where f' is extended to sequences of operations in the obvious way; and
– $S' \in \mathcal{C}'$.

We are mainly interested in the case where $\mathcal{C}'$ is a very weak complexity class, AC^0. As AC^0 contains exactly the problems which can be characterized by first-order formulas (with built-in arithmetic) and as in the context of mathematical structures logical formulas are the natural means to express function f', we usually express $\mathcal{C}'$ by its corresponding logic, e.g. in $\text{DYN}(\mathcal{C}, \text{FO})$.

Remarks 3. – More abstractly, Definition 2 requires a kind of reduction from a dynamic problem D to a dynamic problem D'. The computation of the initial structure is in $\mathcal{C}$, the effect of operations for D' are computable in $\mathcal{C}'$ and the underlying static problem of D' is in $\mathcal{C}'$ as well. Actually, it is a 1-bounded $(\mathcal{C}, \mathcal{C}')$-homomorphism in the notation below.
– Instead of $\mathcal{C}$ and $\mathcal{C}'$ it would be more precise to use complexity classes of *functions*. We only deal with cases with clear connection between the function class and its language class as, e.g., FP, FL, $\text{FNL} = \text{FL}^{\text{NL}}$.
– We follow [17] in that we do not allow operations which delete or insert *elements* into structures. On the other hand, that the auxiliary structure $f(A)$ has the same size as A is not a severe restriction, as all auxiliary structures of polynomial size $p(n)$ can be encoded over the universe of A.

The main difference to the setting of [17, 11] is that they always start with a fixed initial structure and only distinguish precomputations of the same complexity as the updates ($\text{DYN}\mathcal{C}$) and polynomial-time precomputations ($\text{DYN}\mathcal{C}^+$). The relationship to our classes is considered in the following proposition. We call a dynamic problem *polynomially connected*, if for each pair A, A' of structures of size n there is a polynomial size sequence w of operations such that $w(A) = A'$.

Proposition 4. *If a problem in* $\text{DYN}\mathcal{C}^+$ *is polynomially connected then it is also in* $\text{DYN}(\text{P}, \mathcal{C})$.

Note that the first notion of a problem in this statement refers to [17], i.e., it consists of sequences of operations only.

3 Some Precise Upper Bounds

In this section, we show for various dynamic problems membership in $\text{DYN}(\mathcal{C}, \text{FO})$ where the underlying static problem is complete for $\mathcal{C}$. dynREACH is the (directed) reachability problem with operations that insert or delete edges and set s or t to a new node. All other dynamic graph problems have insertion and deletion of edges as operations. dyn2SAT has deletion and insertion of clauses.

Theorem 5. *(a) The dynamic* PARITY *problem* dynPARITY*, with operations which set or unset one bit is in* $\textsc{Dyn}(\mathrm{ACC}^0[2], \mathrm{FO})$.
(b) The dynamic deterministic reachability problem $\mathrm{dynREACH}_d$*, which asks for a path on which every node has outdegree 1, is in* $\textsc{Dyn}(\mathrm{LOGSPACE}, \mathrm{FO})$.
(c) dynREACH *for undirected graphs is in* $\textsc{Dyn}(\mathrm{SL}, \mathrm{FO})$.
(d) dynREACH *for acyclic graphs is in* $\textsc{Dyn}(\mathrm{NL}, \mathrm{FO})$.
(e) Dynamic 2-colorability dyn2COL *for undirected graphs is in* $\textsc{Dyn}(\mathrm{SL}, \mathrm{FO})$.

Proof (Sketch). (a) requires only one auxiliary bit, the parity bit itself. For (b) we use a ternary auxiliary relation containing all triples (x, y, z) such that there is a deterministic path from x to y via z. (c), (d) and (e) follow directly by analyzing the proofs in [17, 3]. □

From [10] we can conclude dynREACH $\in \textsc{Dyn}(\mathrm{P}, \mathrm{TC}^0)$, the precise classification remains open. From this result one also gets dyn2SAT $\in \textsc{Dyn}(\mathrm{P}, \mathrm{TC}^0)$.

4 Reductions

Given a pair (A, w) a reduction from D to D' has to map A to an initial structure A'. Further, an operation w_i has to be mapped to one or more operations of D'. The image of w_i could depend on the previous operations $w_1, \ldots, w_{i-1}$ and on A. But we follow [11] and require that each w_i is mapped independently.

Definition 6. Let $D = D(S, \Sigma)$ and $D' = D(S', \Sigma')$ be dynamic problems over vocabularies τ and τ', respectively. A *reduction* from D to D' is a pair (f, h) of mappings with the following properties:

- For each $n \in \mathbb{N}$, f maps τ-structures of size n to τ'-structures of size n', where $n' = p(n)$ for some polynomial p.
- For each $n \in \mathbb{N}$, h is a string homomorphism from Σ_n^* to $\Sigma_{n'}'^*$.
- For each τ-structure A, and each sequence w of operations on A,

$$(f(A), h(w)) \in D' \iff (A, w) \in D.$$

If f and h can be computed in complexity $\mathcal{C}$ and $\mathcal{C}'$, respectively, we say that the reduction is a $(\mathcal{C}, \mathcal{C}')$-reduction. We write $D \leq_{\mathcal{C},\mathcal{C}'} D'$.

If (f, h) is a reduction from $D(S, \Sigma)$ to $D' = D(S', \Sigma')$ then f is a reduction from S to S'. Also, although we require that $h(\sigma)(f(A)) \in D'$ if and only if $\sigma(A) \in D$, the structures $h(\sigma)(f(A))$ and $f(\sigma(A))$ need not to be the same. If (f, h) additionally fulfills $h(\sigma)(f(A)) = f(\sigma(A))$ it is called a *homomorphism*.[1]

A reduction is *k-bounded*, if $|h(s)| \leq k$, for each $s \in \cup_{i=1}^{n} \Sigma_i$. It is *bounded* if it is k-bounded, for some k. Note, that $|h(s)|$ refers to the number of operations, not to the length of their encoding. We write $\leq^b$ for bounded reductions. We are mainly interested in the case where $\mathcal{C}$ is LOGSPACE or FO and $\mathcal{C}' = \mathrm{FO}$.

[1] In [11] the term *homomorphism* was used for what we call *reduction*.

Example 7. dynPARITY $\leq^b_{\mathrm{FO,FO}}$ dynREACH: By f a string $x = x_1 \cdots x_n$ is mapped to a graph with nodes (i, j), where $i \in [n+1]$ and $j \in \{0, 1\}$. If $x_i = 0$ there are edges from $(i-1, 0)$ to $(i, 0)$ and from $(i-1, 1)$ to $(i, 1)$. Otherwise, there are edges from $(i-1, 0)$ to $(i, 1)$ and from $(i-1, 1)$ to $(i, 0)$. Clearly, x is in PARITY iff there is a path from $(0, 0)$ to $(n, 0)$. Each operation on the string is mapped to at most four operations in the graph in a straightforward manner.

The following propositions show that our dynamic reductions and dynamic complexity classes fit together.

Proposition 8. *The relations* $\leq^b_{\mathrm{LOGSPACE,FO}}$ *and* $\leq^b_{\mathrm{FO,FO}}$ *are transitive.*

Proposition 9. *Let* $\mathcal{C}$ *and* $\mathcal{C}'$ *be closed under functional composition,* $\mathrm{FO} \subseteq \mathcal{C}$ *and* $\mathrm{FO} \subseteq \mathcal{C}'$*. Then* $\textsc{Dyn}(\mathcal{C}, \mathcal{C}')$ *is closed under bounded* $(\mathrm{FO}, \mathrm{FO})$*-reductions.*

Proposition 9 also holds for bounded $(\mathrm{LOGSPACE}, \mathrm{FO})$-reductions if $\mathcal{C}$ is required to be closed under logspace-reductions.

5 Complete Problems

In [11] a complete problem for DynFO under bounded first-order homomorphisms was established. We show next that the problem can be translated into our setting and, furthermore, it can be adapted to obtain complete problems for other classes of the type $\textsc{Dyn}(\mathcal{C}, \mathrm{FO})$.

The dynamic problem single step circuit value (SSCV) of possibly cyclic circuits is defined in [11]. They consider operations to change the type of a gate, "and", "or", or "nand", to add and delete wires between gates, and to set the current value of a gate. There is also a operation that propagates values one step through the circuit. Only sequences of these operations are considered in [11]. To translate SSCV in our setting, we have to add an underlying static problem. The most natural choice would be the set of all such circuits whose gate 0 has value "1". But the resulting problem has very low complexity.

Proposition 10. SSCV *is complete for* $\textsc{Dyn}(\mathrm{FO}, \mathrm{FO})$ *under bounded* $(\mathrm{FO}, \mathrm{FO})$*-homomorphisms.*

This follows directly from [11] since no auxiliary data structure is needed to prove SSCV $\in$ DynFO and the circuits they use are first-order definable.

To obtain a dynamic problem complete for a larger class, e.g. $\textsc{Dyn}(\mathrm{P}, \mathrm{FO})$, we have to choose a static problem that is complete for P. We use a modification of the generic complete problem for P. The set of all tuples $(c, x, 1^m)$, such that c encodes a Turing machine that computes a circuit whose gate 0 has value "1" on input x within m steps, is complete for P. This problem can be combined with the operations from above to $\mathrm{SSCV}_{\mathrm{P}}$, i.e., we substitute the circuit of SSCV by instructions to build a circuit. The operations have to change these instructions accordingly, e.g., the operation $\mathtt{and}(i)$ should result in instructions to build the same circuit except that gate i is an "and" gate.

Theorem 11. $\mathrm{SSCV_P}$ *is complete for* $\textsc{Dyn}(\mathrm{P}, \mathrm{FO})$ *under bounded* $(\mathrm{FO}, \mathrm{FO})$*-homomorphisms.*

This result can be transfered to other complexity classes like $\textsc{Dyn}(\mathrm{L}, \mathrm{FO})$ and $\textsc{Dyn}(\mathrm{NL}, \mathrm{FO})$ as well as to classes using TC^0 circuits to compute updates.

6 Connections to Static Complexity

Now, we establish some connections between static and dynamic problems.

To transfer properties from static problems to dynamic problems, the dynamic problems need to depend in a uniform way on their underlying static problems. The most problems studied in earlier work (e.g., [17, 3, 6, 10]) have essentially the same operations: insertion and deletion of tuples and setting constants. Therefore, we will now study dynamic problems with these operations. For a static problem $S \subseteq \mathrm{STRUC}[\tau]$ we define the set $\Sigma_{\mathrm{can}}(S)$ of canonical operations by

$$\begin{aligned}\Sigma_{\mathrm{can}}(S) = \{&\mathtt{insert}_R, \mathtt{delete}_R \mid \text{for all relation symbols } R \in \tau\} \\ &\cup \{\mathtt{set}_c \mid \text{for all constant symbols } c \in \tau\}\end{aligned}$$

We call $D_{\mathrm{can}}(S) := D(S, \Sigma_{\mathrm{can}}(S))$ the *canonical dynamic problem* for S.

An interesting question is the dynamic complexity of NP-problems. One might assume that if an NP-complete problem has polynomial dynamic complexity (i.e., is in $\textsc{Dyn}(\mathcal{C}, \mathrm{P})$, for some class $\mathcal{C}$) then $\mathrm{P} = \mathrm{NP}$. Of course, this holds for $\mathcal{C} = \mathrm{P}$, but we can not prove it for more powerful classes $\mathcal{C}$. Nevertheless, we can show the following result which draws a simple but interesting connection between dynamic and non-uniform complexity.

Theorem 12. *Let* S *be* NP*-complete. Then* $\mathrm{NP} \subseteq \mathrm{P/POLY}$ *iff there is a class* $\mathcal{F}$ *of polynomially size-bounded functions such that* $D_{can}(S) \in \textsc{Dyn}(\mathcal{F}, \mathrm{P})$.

Proof. For the *if* direction, as P/POLY is closed under reductions, it suffices to show that S is in P/POLY if its canonical dynamic version is in $\textsc{Dyn}(\mathcal{F}, \mathrm{P})$. For each n, let E_n denote the τ-structure of size n with empty relations and all constants set to 0. The advice for size n inputs is just the auxiliary structure for E_n. For a structure S of size n, an auxiliary structure can be computed by adding tuples to E_n and updating the auxiliary structure for E_n accordingly. Whether S is accepted can be derived from the final result of this process.

For the *only if* direction, assuming $S \in \mathrm{NP} \subseteq \mathrm{P/POLY}$, let C_n denote the polynomial advice for S for inputs of size n. The auxiliary structure for a structure A simply consists of A itself together with an encoding of C_n. The update operations only change the A part of the auxiliary structure. Clearly, updates can be done in polynomial time and checking membership of (A, C_n) is in polynomial time by assumption. □

The definition of reductions between dynamic problems already requires that there is a reduction between the underlying static problems. The following result

shows that sometimes one can say more about this reduction, if we are starting from a homomorphism between dynamic problems. In a *bounded* FO*-reduction* (bfo) each tuple of the source structure affects only a bounded number of tuples in the target structure [17].

Theorem 13. *Let S and T be problems. For every bounded* $(\mathrm{FO}, \mathrm{FO})$*-homomorphism (f, h) from $D_{can}(S)$ to $D_{can}(T)$, f is a bfo-reduction from S to T.*

Proof. By definition f is an FO-reduction from S to T. We have to show that f is bounded in the above mentioned sense.

Let τ be the signature of S and let A and A' be two τ-structures which differ only in a single tuple t, say, $A' = \mathtt{insert}(t)(A)$. Because (f, h) is a homomorphism, $f(A)$ and $f(A')$ differ only in the tuples and constants affected by the operations in $h(\mathtt{insert}(t))$. The number of these operations is bounded and each operation affects only one tuple or constant. Since h does not depend on A and A', we can conclude that f is bounded and, therefore, a bfo-reduction. □

As stated in [11], canonical dynamic problems cannot be complete for dynamic complexity classes in our general setting. But their argument fails if we restrict to classes of canonical dynamic problems. By Theorem 13, problems complete for such classes under bounded $(\mathrm{FO}, \mathrm{FO})$-homomorphisms must be based on problems complete under bfo-reductions, i.e., dynREACH for acyclic graphs cannot be complete for $\textsc{Dyn}(\mathrm{NL}, \mathrm{FO})$ under bounded $(\mathrm{FO}, \mathrm{FO})$-homomorphisms because dynREACH is not complete for NL under bfo-reductions [17].

7 A LOGCFL-Complete Problem with First-Order Dynamic Complexity

In this section we turn to the question of the maximum possible static complexity of a problem with first-order updates. In dealing with this question one has to distinguish between "redundant" and "non-redundant" problems.

It was observed in [16, 17] that by blowing up the encoding of a problem its dynamic complexity can be decreased and that each problem in P has a padded version in $\textsc{Dyn}(\mathrm{P}, \mathrm{FO})$. To be more precise, let, for a (string) problem S, $\mathrm{PAD}(S)$ be its *padded version* $\mathrm{PAD}(S) = \{w^{|w|} \mid w \in S\}$. If it is a set of graphs then instances of $\mathrm{PAD}(S)$ consist of n disjoint graphs of n-vertex graphs. It was shown in [17] that $\mathrm{PAD}(\mathrm{REACH}_a)$, the padded version of the alternating reachability problem is in $\textsc{Dyn}(\mathrm{P}, \mathrm{FO})$. Note that, for graphs with n nodes it needs n operations to make a significant change, i.e., this result is just a restatement of the fact that REACH_a is in $\mathrm{FO}[n]$ [13]. As noted by the authors in [16], this result is based on a redundant encoding. They defined a notion of non-redundant problems by completeness under a certain kind of reduction. But this notion does not seem to be applicable to complexity classes below P.

We are interested in first-order updates to non-redundant problems. The complexity-wise highest problems known so far to have such updates are complete for NL, e.g., the reachability problem on acyclic graphs.

In this section we will improve this result by establishing first-order updates for a non-redundant problem complete for LOGCFL.

The class LOGCFL consists of all problems that are logspace-reducible to a context free language and is placed between NL and AC^1. More on LOGCFL can be found in [8].

The problem we consider is the canonical dynamic problem for a reachability problem on labeled, acyclic graphs: D_2LREACH(acyclic). The labels are symbols drawn from $\Sigma = \{a, b, \bar{a}, \bar{b}\}$. The problems asks for a path between two nodes s and t that is labeled by a string in D_2, the Dyck language with two types of parentheses given by the following context free grammar.

$$D_2 : \qquad S \to aS\bar{a}S \mid bS\bar{b}S \mid \epsilon$$

Proposition 14. D_2LREACH(acyclic) *is complete for* LOGCFL.

This can be shown by a reduction similar to the one given in [9] for the P-completeness of D_2LREACH.

We represent D_2LREACH(acyclic) as a set of structures over vocabulary $\langle R_a^2, R_{\bar{a}}^2, R_b^2, R_{\bar{b}}^2, s, t\rangle$, i.e., for each symbol $\sigma \in \Sigma$ there is a binary relation R_σ^2 and we allow insertion and deletion of tuples in these relations as well as setting the constants. We assume that every edge has a unique label, i.e., to change the label of an edge, it first has to be deleted before it is inserted into another relation. We also assume that the operations preserve acyclicity. Both conditions can be checked by a first-order formula.

We can now state the main theorem of this section.

Theorem 15. $D_{can}(D_2\text{LREACH(acyclic)})$ *is in* $\textsc{Dyn}(\text{LOGCFL}, \text{FO})$.

We will sketch the proof in the remainder of this section. First, we introduce the auxiliary data structure we are going to use. Insertion and deletion of edges is considered afterwards.

7.1 An Auxiliary Data Structure

One might be tempted to use as auxiliary structure the set of pairs (u, v) such that there is a D_2-labeled path from u to v. As the example in Figure 1 indicates, this information is not sufficient. In this example, the data structure would contain only the tuples (u, u) for $u \in \{0, \ldots, 6\}$. This does not help to recognize new paths, e.g., the path from 0 to 6 after insertion of an edge $(2, 3)$ labeled a.

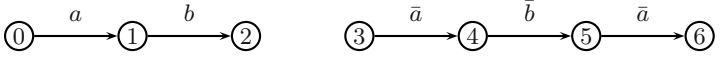


Fig. 1. Example of a labeled graph

A more helpful information is that the concatenation of the labels on the paths from 0 to 2 and from 4 to 6 is in D_2. This is exactly the kind of information

we are going to store. More precisely, we maintain a relation P of arity four which contains the tuples (u_1, v_1, u_2, v_2) such that there are paths from u_1 to v_1 labeled s_1 and from u_2 to v_2 labeled s_2 with $s_1 s_2 \in D_2$.

We also maintain the transitive closure T of the graph, ignoring the labels. Therefore, our auxiliary data structure is over vocabulary

$$\tau = \langle R_a^2, R_b^2, R_{\bar{a}}^2, R_{\bar{b}}^2, P^4, T^2, s, t \rangle.$$

The relation P stores information about concatenations of labels of two paths. To update P during insertion and deletion of edges, we will need the corresponding information for three or even four paths. Fortunately, this information can be extracted from P by a first-order formula.

Lemma 16. *For every $k \geq 1$ there is a first-order formula π_k over vocabulary τ, such that $\pi_k(u_1, v_1, \ldots, u_k, v_k)$ holds if and only if there are paths from u_i to v_i labeled with strings s_i for $i \leq k$, and $s_1 \cdots s_k \in D_2$.*

7.2 Inserting Edges

To update the relation P after an edge (x, y) was inserted, we have to find paths that use the new edge. All tuples referring to other paths have been in P before.

A tuple in P talks about two paths. (x, y) be used in both or only in the first or second path. These three cases are distinct in the following lemma.

Lemma 17. *For each $\sigma \in \Sigma$ there are* FO*-formulas $\varphi_{\sigma 1}$, $\varphi_{\sigma 2}$ and φ_σ, such that*

- *$\varphi_{\sigma 1}(u_1, v_1, u_2, v_2, x, y)$ is true iff there are paths from u_1 to x labeled s_1, from y to v_1 labeled s_2, and from u_2 to v_2 labeled s_3 such that $s_1 \sigma s_2 s_3$ is in D_2.*
- *$\varphi_{\sigma 2}(u_1, v_1, u_2, v_2, x, y)$ is true iff there are paths from u_1 to v_1 labeled s_1, from u_2 to x labeled s_2, and from y to v_2 labeled s_3 such that $s_1 s_2 \sigma s_3$ is in D_2.*
- *$\varphi_\sigma(u_1, v_1, u_2, v_2, x, y)$ is true iff there are paths from u_1 to x labeled s_1, from y to v_1 labeled s_2, from u_2 to x labeled s_3, and from y to v_2 labeled s_4 such that $s_1 \sigma s_2 s_3 \sigma s_4$ is in D_2.*

Proof (Sketch). We restrict to φ_{a1}. In the case of φ_{a1} we are interested in tuples(u_1, v_1, u_2, v_2) describing paths using the a-labeled edge (x, y) between u_1 and v_1. The idea in building φ_{a1} is to guess the corresponding edge (z, z') labeled $\bar{a}$ by existentially quantifying z and z'. Then P is used to check if there is a correctly labeled path using these two edges. There are two cases: (z, z') might be on the path form u_1 to v_1 behind (x, y) or on the path from u_2 to v_2. These considerations lead to the following formula:

$$\varphi_{a1}(u_1, v_1, u_2, v_2, x, y) \equiv \exists z, z' (R_{\bar{a}}(z, z') \wedge [(\pi_3(u_1, x, z', v_1, u_2, v_2) \wedge \pi_1(y, z)) \vee (P(u_1, x, z', v_2) \wedge P(y, v_1, u_2, z))]) \qquad \square$$

All that remains to update P is to put the old and new tuples together. Therefore, the update formulas for an operation $\texttt{insert}_{R_\sigma}(x, y)$, for $\sigma \in \Sigma$, are given by:

$$\begin{aligned} P'(u_1, v_1, u_2, v_2) \equiv\ & P(u_1, v_1, u_2, v_2) \vee \varphi_{\sigma 1}(u_1, v_1, u_2, v_2, x, y) \vee \\ & \varphi_{\sigma 2}(u_1, v_1, u_2, v_2, x, y) \vee \varphi_\sigma(u_1, v_1, u_2, v_2, x, y) \\ T'(u, v) \equiv\ & T(u, v) \vee (T(u, x) \wedge T(x, v)) \end{aligned}$$

7.3 Deleting Edges

Maintaining P and T under deletion of edges is more complicated. We basically need the following lemma.

Lemma 18. *For each $\sigma \in \Sigma$ there are* FO*-formulas $\psi_{\sigma 1}$, $\psi_{\sigma 2}$ and ψ_σ, such that*

- *$\psi_{\sigma 1}(u_1, v_1, u_2, v_2, x, y)$ expresses the following implication: If there is an edge (x, y) labeled σ and $\varphi_{\sigma 1}(u_1, v_1, u_2, v_2, x, y)$ is true, then there is a path from u_1 to v_1 that does not use (x, y) and a path from u_2 to v_2, so that the concatenation of their labels is in D_2.*
- *$\psi_{\sigma 2}(u_1, v_1, u_2, v_2, x, y)$ expresses the following implication: If there is an edge (x, y) labeled σ and $\varphi_{\sigma 2}(u_1, v_1, u_2, v_2, x, y)$ is true, then there is a path from u_1 to v_1 and a path from u_2 to v_2 that does not use (x, y), so that the concatenation of their labels is in D_2.*
- *$\psi_\sigma(u_1, v_1, u_2, v_2, x, y)$ is the following implication: If there is an edge (x, y) labeled σ and $\varphi_{\sigma 1}(u_1, v_1, u_2, v_2, x, y) \wedge \varphi_{\sigma 2}(u_1, v_1, u_2, v_2, x, y)$ is true, then there are paths from u_1 to v_1 and from u_2 to v_2 that do not use (x, y), so that the concatenation of their labels is in D_2.*

Proof (Sketch). Again, we restrict to the first formula. To build ψ_{a1}, we have to describe a path from u_1 to v_1 not using (x, y). To this end, we make use of a technique from [17]. Such a path exists if there is an edge (z, z') different from (x, y), such that there are a path from u_1 to z, a path from z' to v_1 and a path from z to x but no path from z' to x. In our context, we also need that the concatenation of labels along the path from u_1 via (z, z') to v_1 and a path from u_2 to v_2 is in D_2. This can be done by $\varphi_{\sigma 1}$, where σ is the label of (z, z'). Since we do not know σ, we have to consider all four possibilities.

$$\begin{aligned}
&\psi_{a1}(u_1, v_1, u_2, v_2, x, y) \equiv (R_a(x, y) \wedge \varphi_{a1}(u_1, v_1, u_2, v_2, x, y)) \\
&\rightarrow (\exists z, z'[T(u_1, z) \wedge T(z, x) \wedge E(z, z') \wedge \neg T(z', x) \wedge T(z', v_1) \wedge (z \neq x \vee z' \neq y) \wedge \\
&\qquad ((R_a(z, z') \wedge \varphi_{a1}(u_1, v_1, u_2, v_2, z, z')) \vee (R_{\bar{a}}(z, z') \wedge \varphi_{\bar{a}1}(u_1, v_1, u_2, v_2, z, z')) \vee \\
&\qquad (R_b(z, z') \wedge \varphi_{b1}(u_1, v_1, u_2, v_2, z, z')) \vee (R_{\bar{b}}(z, z') \wedge \varphi_{\bar{b}1}(u_1, v_1, u_2, v_2, z, z')))])
\end{aligned}$$

Here, $\varphi_E(z, z')$ expresses that there is an edge from z to z'. □

Consequently, the updates necessary for an operation $\texttt{delete}_{R_\sigma}(x, y)$ for $\sigma \in \Sigma$ can be described as follows, what concludes the proof of Theorem 15.

$$\begin{aligned}
P'(u_1, v_1, u_2, v_2) \equiv\ & P(u_1, v_1, u_2, v_2) \wedge \psi_{\sigma 1}(u_1, v_1, u_2, v_2, x, y) \wedge \\
& \psi_{\sigma 2}(u_1, v_1, u_2, v_2, x, y) \wedge \psi_\sigma(u_1, v_1, u_2, v_2, x, y) \\
T'(u, v) \equiv\ & T(u, v) \wedge (\neg T(u, x) \vee \neg T(y, v) \vee \exists z, z'[T(u, z) \wedge \varphi_E(z, z') \wedge \\
& T(z', v) \wedge T(z, x) \wedge \neg T(z', x) \wedge (z \neq x \vee z' \neq y)]).
\end{aligned}$$

We end this section with the following corollary, which holds because the auxiliary data structure for the empty graph can be described in first-order.

Corollary 19. *D_{can}(*$\mathrm{D_2LREACH}$(acyclic)*) is in* DynFO.

8 Conclusion

We have taken a step towards a dynamic complexity theory by presenting a more robust notion of dynamic problems and complexity classes. This allowed us to characterize the complexity of several dynamic problems more precisely, thus clarifying the role of precomputation in dynamic complexity which was an open problem of [17]. We also gave complete problems for dynamic complexity classes under a useful kind of reduction. Finally, we presented first-order updates to a first "non-redundant" LOGCFL-complete problem.

We want to give some directions for further research:

- It remains open if there is a non-redundant problem complete for P that allows efficient updates. D_2LREACH might be a candidate. Note that the result of [10] can not be applied to D_2LREACH in a straightforward way since one has to consider paths of exponential length.
- As stated in [11], canonical dynamic problems cannot be complete for dynamic complexity classes in general. Therefore, it might be interesting to look for complete problems for classes of canonical dynamic problems.
- A further issue is to establish connections between algorithmic results on dynamic problems and dynamic complexity theory.

References

1. G. Dong, L. Libkin, and L. Wong. Incremental recomputation in local languages. *Information and Computation*, 181(2):88–98, 2003.
2. G. Dong and J. Su. First-order incremental evaluation of datalog queries. In *Proc. of DBPL-4*, Workshops in Computing, pages 295–308. Springer, 1993.
3. G. Dong and J. Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Information and Computation*, 120(1):101–106, 1995.
4. G. Dong and J. Su. Deterministic FOIES are strictly weaker. *Annals of Mathematics and Artificial Intelligence*, 19(1-2):127–146, 1997.
5. G. Dong and J. Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *Journal of Computer and System Sciences*, 557(3):289–308, 1998.
6. K. Etessami. Dynamic tree isomorphism via first-order updates to a relational database. In *Proc. of the 17th PODS*, pages 235–243. ACM Press, 1998.
7. A. Fiat and G. J. Woeginger, editors. *Online Algorithms, The State of the Art*, volume 1442 of *LNCS*. Springer, 1998.
8. G. Gottlob, N. Leone, and F. Scarcello. Computing LOGCFL certificates. *Theoretical Computer Science*, 270(1-2):761–777, 2002.
9. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.
10. W. Hesse. The dynamic complexity of transitive closure is in $DynTC^0$. *Theoretical Computer Science*, 296(3):473–485, 2003.
11. W. Hesse and N. Immerman. Complete problems for dynamic complexity classes. In *Proc. of the 17th LICS*, pages 313–. IEEE Computer Society, 2002.
12. J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *Proc. of the 30th STOC*, pages 79–89. ACM, 1998.

13. N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.
14. L. Libkin and L. Wong. Incremental recomputation of recursive queries with nested sets and aggregate functions. In *Proc. of DBPL-6*, volume 1369 of *LNCS*, pages 222–238. Springer, 1998.
15. L. Libkin and L. Wong. On the power of incremental evaluation in SQL-like languages. In *Proc. of DBPL-7*, volume 1949 of *LNCS*, pages 17–30. Springer, 2000.
16. P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203–236, 1994.
17. S. Patnaik and N. Immerman. Dyn-FO: A parallel, dynamic complexity class. *Journal of Computer and System Sciences*, 55(2):199–209, 1997.
18. L. Roditty and U. Zwick. A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In *Proc. of the 36th STOC*, pages 184–191. ACM, 2004.

Parametric Duality and Kernelization: Lower Bounds and Upper Bounds on Kernel Size

Jianer Chen[1], Henning Fernau[2], Iyad A. Kanj[3], and Ge Xia[1]

[1] Department of Computer Science, Texas A&M University,
College Station, TX 77843-3112
{chen, gexia}@cs.tamu.edu*
[2] The University of Newcastle,
School of Electrical Engineering and Computer Science,
University Drive, NSW 2308 Callaghan, Australia
Universität Tübingen, Wilhelm-Schickard-Institut für Informatik,
Sand 13, D-72076 Tübingen, Germany
fernau@informatik.uni-tuebingen.de
[3] School of Computer Science, Telecommunications and Information Systems,
DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604-2301
ikanj@cs.depaul.edu†

Abstract. We develop new techniques to derive lower bounds on the kernel size for certain parameterized problems. For example, we show that unless $\mathcal{P} = \mathcal{NP}$, PLANAR VERTEX COVER does not have a problem kernel of size smaller than $4k/3$, and PLANAR INDEPENDENT SET and PLANAR DOMINATING SET do not have kernels of size smaller than $2k$. We derive an upper bound of $67k$ on the problem kernel for PLANAR DOMINATING SET improving the previous $335k$ upper bound by Alber et al.

Keywords: Kernelization, parameterized complexity.

1 Introduction

Many problems which are parameterized tractable become intractable when the parameter is "turned around" (see [9, 12, 14]). As an example, consider the VERTEX COVER and INDEPENDENT SET problems. If n denotes the number of vertices in the whole graph G, then it is well-known that (G, k) is a YES-instance of VERTEX COVER if and only if (G, k_d), where $k_d = n - k$, is a YES-instance of INDEPENDENT SET. In this sense, INDEPENDENT SET is the parametric dual problem of VERTEX COVER. While VERTEX COVER is fixed-parameter tractable on general graphs, INDEPENDENT SET is not. Similarly, while DOMINATING SET is fixed-parameter intractable on general graphs, its parametric dual, called NONBLOCKER, is fixed-parameter tractable.

* This research was supported in part by the NSF under grants CCF-0430683 and CCR-0311590.
† This work was supported in part by DePaul University Competitive Research Grant.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 269–280, 2005.

The landscape changes when we turn our attention towards special graph classes, e.g., problems on *planar graphs* [2]. Here, for example, both INDEPENDENT SET and DOMINATING SET are fixed-parameter tractable. In fact, and in contrast to what was stated above, there are quite many problems for which both the problem itself and its dual are parameterized tractable.

The beauty of problems which are together with their dual problems fixed-parameter tractable, is that this constellation allows from an algorithmic standpoint for a two-sided attack on the original problem. This two-sided attack enabled us to derive lower bounds on the kernel size for such problems (under classical complexity assumptions). For example, we show that unless $\mathcal{P} = \mathcal{NP}$, PLANAR VERTEX COVER does not have a kernel of size smaller than $4k/3$, and PLANAR INDEPENDENT SET and PLANAR DOMINATING SET do not have kernels of size smaller than $2k$. To the authors' knowledge, this is the first group of results establishing lower bounds on the kernel size of parameterized problems.

Whereas the lower bounds on the kernel size for PLANAR VERTEX COVER and PLANAR INDEPENDENT SET come close to the known upper bounds of $2k$ and $4k$ on the kernel size for the two problems, respectively, the lower bound derived for PLANAR DOMINATING SET is still very far from the $335k$ upper bound on the problem kernel (computable in $O(n^3)$ time), which was given by Alber et al. [1]. To bridge this gap, we derive better upper bounds on the problem kernel for PLANAR DOMINATING SET. We improve the reduction rules proposed in [1], and introduce new rules that *color* the vertices of the graph enabling us to observe many new combinatorial properties of its vertices. These properties allow us to prove a much stronger bound on the number of vertices in the reduced graph. We show that the PLANAR DOMINATING SET problem has a kernel of size $67k$ that is computable in $O(n^3)$ time. This is a significant improvement over the results in [1].

2 Preliminaries

A *parameterized problem* P is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a fixed alphabet and $\mathbb{N}$ is the set of all non-negative integers. Therefore, each instance of the parameterized problem P is a pair (I, k), where the second component k is called the *parameter*. The language $L(P)$ is the set of all YES-instances of P. We say that the parameterized problem P is *fixed-parameter tractable* [7] if there is an algorithm that decides whether an input (I, k) is a member of $L(P)$ in time $f(k)|I|^c$, where c is a fixed constant and $f(k)$ is a recursive function independent of the input length $|I|$. The class of all fixed parameter tractable problems is denoted by FPT.

A mapping $s : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ is called a *size function* for a parameterized problem P if:

- $0 \leq k \leq s(I, k)$,
- $s(I, k) \leq |I|$, and
- $s(I, k) = s(I, k')$ for all appropriate k, k' *(independence)*. Hence, we can also write $s(I)$ for $s(I, k)$.

A problem P together with its size function s are denoted (P, s). The *dual problem* P_d of P is the problem whose corresponding language (i.e., set of YES-instances) $L(P_d) = \{(I, s(I) - k) \mid (I, k) \in L(P)\}$. The dual of the dual of a problem (with a given size function) is again the original problem. We give some examples below.

d-HITTING SET
Given: A hypergraph $G = (V, E)$ with *edge degree* bounded by d, i.e., $\forall e \in E, |e| \leq d$
Parameter: k
Question: Is there a *hitting set* of size at most k, i.e.,

$$\exists C \subseteq V, |C| \leq k, \forall e \in E, C \cap e \neq \emptyset?$$

The special case in which $d = 2$ corresponds to the VERTEX COVER problem in undirected graphs. Let $L(d\text{-HS})$ denote the language of d-HITTING SET. Taking as size function $s(G) = |V|$, it is clear that the dual problem obeys $(G, k_d) \in L(d\text{-HS}_d)$ if and only if G has an *independent set* of cardinality k_d.

DOMINATING SET
Given: A (simple) graph $G = (V, E)$
Parameter: k
Question: Is there a *dominating set* of size at most k, i.e.,

$$\exists D \subseteq V, |D| \leq k,\ \forall v \in V \setminus D\ \exists d \in D, (d, v) \in E?$$

Taking as size function $s(G) = |V|$, it is clear that the dual problem obeys $(G, k_d) \in L(\text{DS}_d)$ if and only if G has a *nonblocker set* (i.e., the complement of a dominating set) of cardinality k_d.

Generally speaking, it is easy to "correctly" define the dual of a problem for selection problems as formalized in [3].

A *kernelization* for a parameterized problem P with size function s is a polynomial-time computable reduction which maps an instance (I, k) onto (I', k') such that: (1) $s(I') \leq g(k)$ (g is a recursive function), (2) $k' \leq k$, and (3) $(I, k) \in L(P)$ if and only if $(I', k') \in L(P)$. I' is called the *problem kernel* of I. It is known (see [8]) that a parameterized problem is fixed-parameter tractable if and only if it has a kernelization. Of special interest to us in this paper are problems with *linear kernels* in which $g(k) = \alpha k$ for some constant $\alpha > 0$. Such small kernels are known, in particular, for graph problems restricted to planar graphs.

3 Lower Bounds on Kernel Size

Practice in the study of parameterized algorithms has suggested that improved kernelization can lead to improved parameterized algorithms. Many efforts have been made towards obtaining smaller kernels for well-known $\mathcal{NP}$-hard parameterized problems (see for example [1, 5, 8]). A natural question to ask along

this line of research, is about the limit of polynomial time kernelization. In this section we develop techniques for deriving lower bounds on the kernel size for certain well-known $\mathcal{NP}$-hard parameterized problems.

Theorem 1. *Let (P, s) be an $\mathcal{NP}$-hard parameterized problem. Suppose that P admits an αk kernelization, and its dual P_d admits an $\alpha_d k_d$ kernelization, where $\alpha, \alpha_d \geq 1$. If $(\alpha - 1)(\alpha_d - 1) < 1$ then $\mathcal{P} = \mathcal{NP}$.*

Proof. Suppose that the statement of the theorem is true, and let $r(\cdot)$ denote the assumed linear kernelization reduction for P. Similarly, let $r_d(\cdot)$ be the linear kernelization reduction for P_d. Consider the following reduction R, which on input (I, k) of P performs the following:

if $k \leq \frac{\alpha_d}{\alpha+\alpha_d} s(I)$ **then** compute $r(I, k)$;
else compute $r_d(I, s(I) - k)$.

Now if $k \leq \frac{\alpha_d}{\alpha+\alpha_d} s(I)$, then $s(I') \leq \alpha k \leq \frac{\alpha\alpha_d}{\alpha+\alpha_d} s(I)$. Otherwise:

$$\begin{aligned} s(I') &\leq \alpha_d k_d \\ &= \alpha_d (s(I) - k) \\ &< \alpha_d \left(s(I) - \frac{\alpha_d}{\alpha + \alpha_d} s(I) \right) \\ &= \frac{\alpha\alpha_d}{\alpha + \alpha_d} s(I). \end{aligned}$$

Since $(\alpha-1)(\alpha_d-1) < 1$, or equivalently $\frac{\alpha\alpha_d}{\alpha+\alpha_d} < 1$, by repeatedly applying R (at most polynomially-many times), the problem P can be solved in polynomial time. This completes the proof.

From the previous theorem, and assuming $\mathcal{P} \neq \mathcal{NP}$, we immediately obtain the following.

1. **Corollary 1.** For any $\epsilon > 0$, there is no $(4/3 - \epsilon)k$ kernel for PLANAR VERTEX COVER.

   Proof. The four-color theorem implies a $4k$-kernelization for PLANAR INDEPENDENT SET, which is the dual problem of PLANAR VERTEX COVER.

2. **Corollary 2.** For any $\epsilon > 0$, there is no $(2 - \epsilon)k$ kernel for PLANAR INDEPENDENT SET. This result remains true if we restrict the problem to graphs of maximum degree bounded by three, or even to planar graphs of maximum degree bounded by three (both problems are $\mathcal{NP}$-hard).

   Proof. The general VERTEX COVER problem, which is the dual of the INDEPENDENT SET problem, has a $2k$-kernelization [5]. This kernelization is both planarity and bounded-degree preserving.

3. **Corollary 3.** For any $\epsilon > 0$, there is no $(3/2-\epsilon)k$-kernelization for VERTEX COVER restricted to triangle-free planar graphs (this problem is still $\mathcal{NP}$-hard [15–Chapter 7]).

Proof. Based on a theorem by Grötzsch (which can be turned into a polynomial-time coloring algorithm; see [11]) it is known that planar triangle-free graphs are 3-colorable. This implies a $3k$ kernel for INDEPENDENT SET restricted to this graph class, which gives the result. Observe that the $2k$-kernelization for VERTEX COVER on general graphs preserves planarity and triangle-freeness, which implies that this restriction of the problem has a $2k$-kernelization.

4. **Corollary 4.** For any $\epsilon > 0$, there is no $(335/334 - \epsilon)k$ kernel for PLANAR NONBLOCKER.

   Proof. A $335k$ kernel for PLANAR DOMINATING SET was derived in [1].

5. **Corollary 5.** For any $\epsilon > 0$, there is no $(2 - \epsilon)k$ kernel for PLANAR DOMINATING SET. This remains true when further restricting the graph class to planar graphs of maximum degree three (the problem is still $\mathcal{NP}$-hard).

   Proof. In [10], a $2k$-kernelization for NONBLOCKER on general graphs which preserves planarity and degree bounds, was derived (see also [13–Theorem 13.1.3]).

The above results open a new line of research, and prompt us to ask whether we can find examples of problems such that the derived kernel sizes are optimal (unless $\mathcal{P} = \mathcal{NP}$), and whether we can close the gaps between the upper bounds and lower bounds on the kernel size more and more. According to our previous discussion, PLANAR VERTEX COVER on triangle-free graphs is our "best match:" we know how to derive a kernel of size $2k$, and (assuming $\mathcal{P} \neq \mathcal{NP}$) we know that no kernel smaller than $3k/2$ exists. On the other hand, the $335k$ upper bound on the kernel size for PLANAR DOMINATING SET [1] is very far from the $2k$ lower bound proved above. In the next section, we improve this upper bound to $67k$ in an effort to bridge the huge gap between the upper bound and lower bound on the kernel size for this problem.

4 Reduction and Coloring Rules for PLANAR DOMINATING SET

In this section we will only consider planar graphs. For a graph G, we denote by $\gamma(G)$ the size of a minimum dominating set in G. We present an $O(n^3)$ time preprocessing scheme that reduces the graph G to a graph G', such that $\gamma(G) = \gamma(G')$, and such that given a minimum dominating set for G', a minimum dominating set for G can be constructed in linear time. We will color the vertices of the graph G with two colors: black and white. Initially, all vertices are colored black. Informally speaking, white vertices will be those vertices that we know for sure when we color them that there exists a minimum dominating set for the graph excluding all of them. The black vertices are all other vertices. Note that it is possible for white vertices to be in some minimum dominating set, but the

point is that there exists at least one minimum dominating set that excludes all white vertices. We start with the following definitions that are adopted from [1] with minor additions and modifications.

For a vertex v in G denote by $N(v)$ the set of neighbors of v, and by $N[v]$ the set $N(v) \cup \{v\}$. By removing a vertex v from G, we mean removing v and all the edges incident on v from G. For a vertex v in G, we partition its set of neighbors $N(v)$ into three sets: $N_1(v) = \{u \in N(v) \mid N(u) - N[v] \neq \emptyset\}$; $N_2(v) = \{u \in N(v) - N_1(v) \mid N(u) \cap N_1(v) \neq \emptyset\}$; and $N_3(v) = N(v) - (N_1(v) \cup N_2(v))$. For two vertices v and w we define $N(v, w) = N(v) \cup N(w)$ and $N[v, w] = N[v] \cup N[w]$. We partition $N(v, w)$ into three sets: $N_1(v, w) = \{u \in N(v, w) \mid N(u) - N[v, w] \neq \emptyset\}$; $N_2(v, w) = \{u \in N(v, w) - N_1(v, w) \mid N(u) \cap N_1(v, w) \neq \emptyset\}$; and $N_3(v, w) = N(v, w) - (N_1(v, w) \cup N_2(v, w))$.

Definition 1. Let $G = (V, E)$ be a plane graph. A *region* $R(v, w)$ between two vertices v and w is a closed subset of the plane with the following properties:

1. The boundary of $R(v, w)$ is formed by two simple paths P_1 and P_2 in V which connect v and w, and the length of each path is at most three.
2. All vertices that are strictly inside (i.e., not on the boundary) the region $R(v, w)$ are from $N(v, w)$.

For a region $R = R(v, w)$, let $V[R]$ denote the vertices in R, i.e.,

$$V[R] := \{u \in V \mid u \text{ sits inside or on the boundary of } R\}.$$

Let $V(R) = V[R] - \{v, w\}$.

Definition 2. A region $R = R(v, w)$ between two vertices v and w is called *simple* if all vertices in $V(R)$ are common neighbors of both v and w, that is, $V(R) \subseteq N(v) \cap N(w)$.

We introduce the following definitions.

Definition 3. A region $R = R(v, w)$ between two vertices v and w is called *quasi–simple* if $V[R] = V[R'] \cup R^+$, where $R' = R'(v, w)$ is a simple region between v and w, and R^+ is a set of white vertices satisfying the following conditions:

1. Every vertex of R^+ sits strictly inside R'.
2. Every vertex of R^+ is connected to v and not connected to w, and is also connected to at least one vertex on the boundary of R' other than v.

A vertex in $V(R)$ is called a *simple* vertex, if it is connected to both v and w, otherwise it is called *non–simple*. The set of vertices R^+, which consists of the non-simple vertices in $V(R)$, will be referred to as $R^+(v, w)$.

For a vertex $u \in V$, denote by $B(u)$ the set of black vertices in $N(u)$, and by $W(u)$ the set of white vertices in $N(u)$. We describe next the reduction and coloring rules to be applied to the graph G. The reduction and coloring rules

are applied to the graph in the order listed below until the application of any of them does not change the structure of the graph nor the color of any vertex in the graph. The first two reduction rules, **Rule 1** and **Rule 2**, are slight modifications of Rule 1 and Rule 2 introduced in [1]. The only difference is that in the current paper they are only applied to black vertices, and not to all the vertices as in [1].

Rule 1 ([1]). If $N_3(v) \neq \emptyset$ for some black vertex v, then remove the vertices in $N_2(v) \cup N_3(v)$ from G, and add a new white vertex v' and an edge (v, v') to G.

Rule 2 ([1]). If $N_3(v, w) \neq \emptyset$ for two black vertices v, w, and if $N_3(v, w)$ cannot be dominated by a single vertex in $N_2(v, w) \cup N_3(v, w)$, then we distinguish the following two cases.

Case 1. If $N_3(v, w)$ can be dominated by a single vertex in $\{v, w\}$ then: (1.1) if $N_3(v, w) \subseteq N(v)$ and $N_3(v, w) \subseteq N(w)$, remove $N_3(v, w)$ and $N_2(v, w) \cap N(v) \cap N(w)$ from G and add two new white vertices z, z' and the edges $(v, z), (w, z), (v, z'), (w, z')$ to G; (1.2) if $N_3(v, w) \subseteq N(v)$ and $N_3(v, w) \not\subseteq N(w)$, remove $N_3(v, w)$ and $N_2(v, w) \cap N(v)$ from G and add a new white vertex v' and the edge (v, v') to G; and (1.3) if $N_3(v, w) \subseteq N(w)$ and $N_3(v, w) \not\subseteq N(v)$, remove $N_3(v, w)$ and $N_2(v, w) \cap N(w)$ from G and add a new white vertex w' and the edge (w, w') to G.

Case 2. If $N_3(v, w)$ cannot be dominated by a single vertex in $\{v, w\}$, then remove $N_2(v, w) \cup N_3(v, w)$ from G and add two new white vertices v', w' and the edges (v, v'), (w, w') to G.

Rule 3. For each black vertex v in G, if there exists a black vertex $x \in N_2(v) \cup N_3(v)$, color x white, and remove the edges between x and all other white vertices in G.

Rule 4. For every two black vertices v and w, if $N_3(v, w) \neq \emptyset$, then for every black vertex $x \in N_2(v, w) \cup N_3(v, w)$ that does not dominate all vertices in $N_3(v, w)$, color x white and remove all the edges between x and the other white vertices in G.

Rule 5. For every quasi-simple region $R = R(v, w)$ between two vertices v and w, if v is black, then for every black vertex $x \in N_2(v, w) \cup N_3(v, w)$ strictly inside R that does not dominate all vertices in $N_2(v, w) \cup N_3(v, w)$ strictly inside R, color x white and remove all the edges between x and the other white vertices in G.

Rule 6. For every two white vertices u and v, if $N(u) \subseteq N(v)$, and $u \in N_2(w) \cup N_3(w)$ for some black vertex w, then remove v.

Rule 7. For every black vertex v, if every vertex $u \in W(v)$ is connected to all the vertices in $B(v)$, then remove all the vertices in $W(v)$ from G.

Rule 8. For every two black vertices v and w, let $W(v,w) = W(v) \cap W(w)$. If $|W(v,w)| \geq 2$ and there is a degree-2 vertex $u \in W(v,w)$, then remove all vertices in $W(v,w)$ except u, add a new degree-2 white vertex u', and connect u' to both v and w.

A graph G is said to be *reduced* if every vertex in G is colored white or black, and the application of **Rules 1–8** leaves the graph G unchanged. That is, the application of any of the above rules does not change the color of any vertex in G, nor does it change the structure of G.

Theorem 2. *Let G be a graph with n vertices. Then in time $O(n^3)$ we can construct a graph G' from G such that: (1) G' is reduced, (2) $\gamma(G') = \gamma(G)$,* (3) *there exists a minimum dominating set for G' that excludes all white vertices of G', and* (4) *from a minimum dominating set for G' a minimum dominating set for G can be constructed in linear time.*

5 A Problem Kernel for PLANAR DOMINATING SET

Let G be a reduced graph, and let D be a minimum dominating set for G consisting of black vertices such that $|D| = k$. In this section, we will show that the number of vertices n in G is bounded by $67k$. The following definitions are adopted from [1]. The reader is referred to [1] for more details.

Given any dominating set D in a graph G, a *D-region* decomposition of G is a set $\Re$ of regions between pairs of vertices in D such that:

1. For any region $R = R(v,w)$ in $\Re$, no vertex in D is in $V(R)$. That is, a vertex in D can only be an endpoint of a region in $\Re$.
2. No two distinct regions $R_1, R_2 \in \Re$ intersect. However, they may touch each other by having common boundaries.

Note that all the endpoints of the regions in a D-region decomposition are vertices in D. For a D-region decomposition $\Re$, define $V[\Re] = \bigcup_{R \in \Re} V[R]$. A D-region decomposition is *maximal*, if there is no region R such that $\Re' = \Re \cup R$ is a D-region decomposition with $V[\Re] \subsetneq V[\Re']$.

For a D-region decomposition $\Re$, associate a planar graph $G_\Re(V_\Re, E_\Re)$ with possible multiple edges, where $V_\Re = D$, and such that there is an edge between two vertices v and w in $G_\Re$ if and only if $R(v,w)$ is a region in $\Re$. A planar graph with multiple edges is called *thin*, if there is a planar embedding of the graph such that for any two edges e_1 and e_2 between two distinct vertices v and w in the graph, there must exist two more vertices which sit inside the disjoint areas of the plane enclosed by e_1 and e_2.

Alber et al. [1] showed that the number of edges in a thin graph of n vertices is bounded by $3n-6$. They also showed that for any reduced plane graph G and a dominating set D of G, there exists a maximal D-region decomposition for G such that $G_\Re$ is thin. Since the maximal D-region decomposition in [1] starts with any dominating set D and is not affected by the color a vertex can have,

the same results in [1] hold true for our reduced graph G whose vertices are colored black/white, and with a minimum dominating set D consisting only of black vertices. The above discussion is summarized in the following proposition.

Proposition 1. *Let G be a reduced graph and D a dominating set of G consisting of black vertices. Then there exists a maximal D-region decomposition $\Re$ of G such that $G_{\Re}$ is thin.*

Corollary 1. *Let G be a reduced graph with a minimum dominating set D consisting of k black vertices, and let $\Re$ be a maximal D-region decomposition of G such that $G_{\Re}$ is thin. Then the number of regions in $\Re$ is bounded by $3k-6$.*

Proof. The number of regions in $\Re$ is the number of edges in $G_{\Re}$. Since $G_{\Re}$ has $|D| = k$ vertices, by [1], the number of edges in $G_{\Re}$ is bounded by $3k-6$.

In the remainder of this section, $\Re$ will denote a maximal D-region decomposition of G such that $G_{\Re}$ is thin. Let u and v be two vertices in G. We say that u and v are *boundary-adjacent* if (u, v) is an edge on the boundary of some region $R \in \Re$. For a vertex $v \in G$, denote by $N^*(v)$ the set of vertices that are boundary-adjacent to v. Note that for a vertex $v \in D$, since v is black, by **Rule 3**, all vertices in $N_2(v) \cup N_3(v)$ must be white.

Proposition 2. *Let $v \in D$. The following are true.*

(a) (Lemma 6, [1]) Every vertex $u \in N_1(v)$ is in $V[\Re]$.
(b) The vertex v is an endpoint of a region $R \in \Re$. That is, there exists a region $R = R(x, y) \in \Re$ such that $v = x$ or $v = y$.
(c) Every vertex $u \in N_2(v)$ which is not in $V[\Re]$ is connected only to v and to vertices in $N^*(v)$.

Let x be a vertex in G such that $x \notin V[\Re]$. Then by part (b) in Proposition 2, $x \notin D$. Thus, $x \in N(v)$ for some black vertex $v \in D \subseteq V[\Re]$. By part (a) in Proposition 2, $x \notin N_1(v)$, and hence, $x \in N_2(v) \cup N_3(v)$. By **Rule 3**, the color of x must be white. Let $R = R(v, w)$ be a region in $V[\Re]$ of which v is an endpoint (such a region must exist by part (b) of Proposition 2). We distinguish two cases.

Case A. $x \in N_3(v)$. Since v is black, by **Rule 1**, this is only possible if $deg(x) = 1$ and $N_2(v) = \emptyset$ (in this case x will be the white vertex added by the rule). In such case it can be easily seen that we can flip x and place it inside R without affecting the planarity of the graph.

Case B. $x \in N_2(v)$. Note that in this case $N_3(v) = \emptyset$, and x is only connected to v and $N^*(v)$ by part (c) in Proposition 2. If $deg(x) = 2$, by a similar argument to **Case A** above, x can be flipped and placed inside R.

According to the above discussion, it follows that the vertices in G can be classified into two categories: (1) those vertices that are in $V[\Re]$; and (2) those that are not in $V[\Re]$, which are those vertices of degree larger than two that

belong to $N_2(v)$ for some vertex $v \in D$, and in this case must be connected only to vertices in $N^*(v)$. To bound the number of vertices in G we need to bound the number of vertices in the two categories. We start with the vertices in category (2).

Let O denote the set of vertices in category (2). Note that all vertices in O are white, and no two vertices u and v in O are such that $N(u) \subseteq N(v)$. To see why the latter statement is true, note that every vertex in O must be in $N_2(w)$ for some black vertex $w \in D$. So if $N(u) \subseteq N(v)$, then by **Rule 6**, v would have been removed from the graph. To bound the number of vertices in O, we will bound the number of vertices in O that are in $N_2(v)$ where $v \in D$. Let us denote this set by $N^\dagger(v)$. Let $N^*_\dagger(v)$ be the set of vertices in $N^*(v)$ that are neighbors of vertices in $N^\dagger(v)$. Note that every vertex in $N^\dagger(v)$ has degree ≥ 3, is connected only to v and to $N^*_\dagger(v)$, and no two vertices x and y in $N^\dagger(v)$ are such that $N(x) \subseteq N(y)$.

Proposition 3. $|N^\dagger(v)| \leq 3/2|N^*_\dagger(v)|$.

Lemma 1. *The number of vertices in category (2) (i.e., the number of vertices not in $V[\Re]$) is bounded by $18k$.*

Proof. Let v and w be any two distinct vertices in D and observe the following. First, $N^\dagger(v) \cap N^\dagger(w) = \emptyset$, because if $u \in N^\dagger(v) \cap N^\dagger(w)$ then (v, u, w) would be a degenerated region with $u \notin V[\Re]$ contradicting the maximality of $\Re$. Second, from the first observation it follows that $w \notin N^*_\dagger(v)$ and $v \notin N^*_\dagger(w)$(in general no vertex $a \in D$ belongs to $N^*_\dagger(b)$ for any vertex $b \in D$); otherwise, there exists a vertex $u \in N^\dagger(v)$ that is connected to w, and hence u $N^\dagger(v) \cap N^\dagger(w)$, contradicting the first observation. Third, $N^*_\dagger(v) \cap N^*_\dagger(w) = \emptyset$; otherwise, there exists a vertex $u \in N^*_\dagger(v) \cap N^*_\dagger(w)$ that is connected to a category-(2) vertex $a \in N^\dagger(v)$ (or $b \in N^\dagger(w)$) and the degenerated region (v, a, u, w) (or (w, b, u, v)) would contain the vertex $a \notin \Re$ (or $b \notin \Re$), contradicting the maximality of $\Re$.

Let B be the number of vertices not in D that are boundary-adjacent to vertices in D (i.e., in $N^*(v) - D$ for some $v \in D$). Combining the above observations with Proposition 3, it follows that the number of category-(2) vertices is

$$\sum_{v \in D} |N^\dagger(v)| \leq \frac{3}{2} \sum_{v \in D} |N^*_\dagger(v)| \leq 3B/2$$

According to the definition of a region, each region in $\Re$ has at most six vertices on its boundary two of which are vertices in D. Thus, each region in $\Re$ can contribute with at most four vertices to B. By Corollary 1, the number of regions in $\Re$ is bounded by $3k - 6$. It follows that $B \leq 12k - 24$, and hence, the number of category-(2) vertices is bounded by $18k - 36 < 18k$. This completes the proof.

To bound the number of vertices in category (1), fix a region $R(v, w)$ between $v, w \in D$. We have the following lemma.

Lemma 2. *Let $R = R(v, w)$ be a region in $V[\Re]$. The number of vertices in $V(R)$ is bounded by 16.*

Theorem 3. *The number of vertices in the reduced graph G is bounded by $67k$.*

Proof. By Lemma 1, the number of category-(2) vertices in G is bounded by $18k$. According to the discussion before, if we use the $18k$ upper bound on the number of category-(2) vertices, then we can assume that each region in $\Re$ is nice (if this is not the case we obtain a better upper bound on the total number of vertices in G). By Corollary 1, the number of regions in $\Re$ is bounded by $3k - 6$. According to Lemma 2, the number of vertices in $V(R)$, where $R \in \Re$ is a nice region, is bounded by 16. It follows that the number of vertices in $V(\Re)$ is bounded by $48k - 96$. Thus, the number of vertices in $V[\Re]$, and hence, in category (1), is bounded by $48k - 96$ plus the number of vertices in D which are the endpoints of the regions in $\Re$. Therefore the number of vertices in $V[\Re]$ is bounded by $49k - 96$, and the total number of vertices in G is bounded by $67k - 96 < 67k$. This completes the proof.

Theorem 4. *Let G be a planar graph with n vertices. Then in time $O(n^3)$, computing a dominating set for G of size bounded by k can be reduced to computing a dominating set of size bounded by k, for a planar graph G' of $n' < n$ vertices, where $n' \leq 67k$.*

Proof. According to Theorem 2, in time $O(n^3)$ we can construct a reduced graph G' from G where $\gamma(G') = \gamma(G)$, and such that a dominating set for G can be constructed from a dominating set for G' in linear time. Moreover, the graph G' has no more than n vertices. If G has a dominating set of size bounded by k, then G' has a dominating set of size bounded by k (since $\gamma(G) = \gamma(G')$), and by Theorem 3, we must have $n' \leq 67k$, so we can work on computing a dominating set for G'. If this is not the case, then G does not have a dominating set of size bounded by k, and the answer to the input instance is negative. This completes the proof.

Theorem 4, together with Theorem 1, gives:

Corollary 2. *For any $\epsilon > 0$, there is no $(67/66 - \epsilon)k$ kernel for* PLANAR NON-BLOCKER.

References

1. J. Alber, M. Fellows, and R. Niedermeier, Polynomial-time data reduction for dominating set, *Journal of the ACM* **51-3**, (2004), pp. 363-384.
2. J. Alber, H. Fernau, and R. Niedermeier, Parameterized complexity: exponential speedup for planar graph problems, *Journal of Algorithms* **52**, (2004), pp. 26–56.
3. J. Alber, H. Fernau, and R. Niedermeier, Graph separators: a parameterized view, *Journal of Computer and System Sciences* **67**, (2003), pp. 808–832.

4. R. Bar-Yehuda and S. Even, A local-ratio theorem for approximating the weighted vertex cover problem, *Annals of Discrete Mathematics* **25**, (1985), pp. 27-46.
5. J. Chen, I. A. Kanj, and W. Jia, Vertex cover: further observations and further improvement, *Journal of Algorithms 41*, (2001), pp. 280-301.
6. I. Dinur, and S. Safra, On the importance of Being Biased (1.36 hardness of approximating Vertex-Cover), in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing* (STOC 2002), (2002), pp. 33-42. To appear in *Annals of Mathematics*.
7. R. Downey and M. Fellows, *Parameterized Complexity*, Springer-Verlag, 1999.
8. R. Downey, M. Fellows, and U. Stege, Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability, in *Contemporary Trends in Discrete Mathematics*, (R. Graham, J. Kratochvíl, J. Nešetřil, and F. Roberts eds.), proceedings of the DIMACS-DIMATIA Workshop, Prague 1997, *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **vol. 49**, (1999), pp. 49-99.
9. M. Fellows, Parameterized complexity: the main ideas and connections to practical computing, in *Electronic Notes in Theoretical Computer Science* **61**, (2002).
10. M. Fellows, C. McCartin, F. Rosamond, and U. Stege, Coordinatized Kernels and Catalytic Reductions: An Improved FPT Algorithm for Max Leaf Spanning Tree and Other Problems, *Lecture Notes in Computer Science* **1974**, (2000), pp. 240-251.
11. H. Grötzsch, Ein Dreifarbensatz für dreikreisfreie Netze auf der Kugel, *Wiss. Zeitschrift der Martin-Luther-Univ. Halle-Wittenberg, Math.-Naturwiss., Reihe* **8**, (1959), pp. 109-120.
12. S. Khot and V. Raman, Parameterized complexity of finding subgraphs with hereditary properties, *Theoretical Computer Science* **289**, (2002), pp. 997–1008.
13. O. Ore, Theory of Graphs, Colloquium Publications XXXVIII, *American Mathematical Society*, (1962).
14. E. Prieto and C. Sloper, Either/or: Using vertex cover structure in designing FPT-algorithms-the case of k-internal spanning tree, in *Proceedings of WADS 2003, Workshop on Algorithms and Data Structures*, *LNCS* **2748**, (2003), pp. 465–483.
15. R. Uehara, *Probabilistic Algorithms and Complexity Classes*, PhD thesis, Department of Computer Science and Information Mathematics, The University of Electro-Communications, Japan, March 1998.

Shortest Monotone Descent Path Problem in Polyhedral Terrain

Sasanka Roy, Sandip Das, and Subhas C. Nandy

Indian Statistical Institute, Kolkata - 700 108, India

Abstract. Given a polyhedral terrain with n vertices, the shortest monotone descent path problem deals with finding the shortest path between a pair of points, called source (s) and destination (t) such that the path is constrained to lie on the surface of the terrain, and for every pair of points $p = (x(p), y(p), z(p))$ and $q = (x(q), y(q), z(q))$ on the path, if $dist(s, p) < dist(s, q)$ then $z(p) > z(q)$, where $dist(s, p)$ denotes the distance of p from s along the aforesaid path. This is posed as an open problem in [3]. We show that for some restricted classes of polyhedral terrain, the optimal path can be identified in polynomial time. We also propose an elegant method which can return near-optimal path for the general terrain in polynomial time.

1 Introduction

The geodesic path problem on the surface of a polyhedron is an important area of research in Geographic Information System. The measures of the quality of a path include the Euclidean length, maximum altitude along the path, the maximum slope of the path, etc. Extensive studies have been made on the shortest geodesic path problem for both convex and non-convex polyhedral surfaces in 3D. In [13], an $O(n^3 \log n)$ algorithm is proposed for finding the geodesic shortest path between two points on the surface of a convex polyhedron. The generalized version of this problem is studied in [5, 8] where the restriction of convexity is removed; the time complexities of the proposed algorithms are $O(n^2 \log n)$ and $O(n^2)$ respectively. The best known algorithm for producing the optimal solution runs in $O(n \log^2 n)$ time [6], where n is the number of vertices of the polyhedron. In the weighted version of this problem, each face is attached with a positive weight. Here, the shortest weighted path can be computed in $O(n^8 \log n)$ time [9]. Efficient approximation algorithms are also available in [2, 7, 11, 12].

Several variations of the path finding problems in polyhedral terrain are studied in [3]. Given a polyhedral terrain $\mathcal{T}$ with n vertices, the proposed algorithm constructs a linear size data structure in $O(n \log n)$ time and can efficiently answer the following query: given a pair of points s and t on the surface of $\mathcal{T}$, and an altitude ξ, does there exist a path from s to t such that for each point p on the path, $z(p) \leq \xi$?

We address the problem of computing the shortest among all possible monotone descending paths (if exists) between a pair of points on the surface of a

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 281–292, 2005.

polyhedral terrain. This is a long-standing open problem. In [3], it is specifically mentioned that *no bound on the combinatorial or Euclidean length of the shortest monotone descent path between a pair of points on the surface of a polyhedral terrain can be given.* Some interesting observations of the problem lead to efficient polynomial time algorithm for solving this problem in the following special cases.

P1 the sequence of faces through which the optimum path passes, are in convex (resp. concave) position (see Section 3), provided such a path exists.
P2 the sequence of faces through which the optimum path passes, have their boundaries parallel to each other (but the faces are not all necessarily convex (resp. concave)).

In problem P1, the time and space complexities for preprocessing the faces of the terrain are $O(n^2 \log n)$ and $O(n^2)$ respectively, and the shortest path query can be answered in $O(k + \log n)$ time. In problem P2, if a sequence of k faces are given whose boundaries are parallel to each other, the problem can be solved in $O(k \log k)$ time. The solution technique of P2 indicates the hardness of handling the general terrain. Finally, we present an efficient heuristic method for solving the problem on general terrain; it runs in $O(n^4 \log n)$ time. An upper bound on the amount of deviation of our result from the optimal solution is also given.

The problem is motivated from the agricultural applications where the objective is to lay a canal of minimum length from the source of water at the top of the mountain to the ground for irrigation purpose. In particular, problem P2 finds its another important application in designing of fluid circulation system in automobiles or refrigerator/air-condition machines.

2 Preliminaries

A terrain $\mathcal{T}$ is a polyhedral surface in 3D space with a special property: the vertical line at any point on the XY-plane intersects the surface of $\mathcal{T}$ at most once. Thus, the projections of all the faces of a terrain on the XY-plane are mutually non-intersecting at their interior. Each vertex p of the terrain is specified by a triple $(x(p), y(p), z(p))$. Without loss of generality, we assume that all the faces of the terrain are triangles, and the source point s is a vertex of the terrain.

Definition 1. *[8] Let f and f' be a pair of faces of $\mathcal{T}$ sharing an edge e. The* planar unfolding *of face f' onto face f is the image of the points of f' when rotated about the line e onto the plane of f such that the points of f and the points of f' lie in two different sides of the edge e respectively (i.e., faces f' and f do not overlap after unfolding).*

Let $\{f_0, f_1, \ldots, f_m\}$ be a sequence of adjacent faces. The edge common to f_{i-1} and f_i is e_i. We define the *planar unfolding with respect to the edge sequence* $\mathcal{E} = \{e_1, e_2, \ldots, e_{m-1}\}$ as follows: obtain planar unfolding of face f_m onto face f_{m-1}, then get the planar unfolding of the resulting plane onto f_{m-2}, and so on; finally, get the planar unfolding of the entire resulting plane onto f_0. From now onwards, this event will be referred to as $U(\mathcal{E})$.

Let $\pi(s,t)$ be a simple path from a point s to a point t on the surface of the terrain. The geodesic distance $dist(p,q)$ between a pair of points p and q on $\pi(s,t)$ is the length of the simple path from p to q along $\pi(s,t)$. The path $\pi_{geo}(s,t)$ is said to be the *geodesic shortest path* if the distance between s and t along $\pi_{geo}(s,t)$ is minimum among all possible simple paths from s to t.

Definition 2. *A simple path $\pi(s,t)$ $(z(s) > z(t))$ is a monotone descent path if for every pair of points $p, q \in \pi(s,t)$, $dist(s,p) < dist(s,q)$ implies $z(p) \geq z(q)$.*

We will use $\pi_{md}(p,q)$ and $\delta(p,q)$ to denote the shortest monotone descent path from p to q and its length respectively. If $\pi_{geo}(p,q)$ corresponds to the line segment $[p,q]$ in the unfolded plane along an edge sequence and it satisfies monotone descent property, then $\pi_{md}(p,q) = [p,q]$, and q is said to be *straight line reachable* from p in unfolded plane. It can be shown that (i) $\pi_{md}(s,t)$ is a simple path from s to t, (ii) it does not pass through any face more than once, and (iii) the two end-points of each line-segment on this path must lie on two edges of a face of $\mathcal{T}$. Note that, a monotone descent path between a pair of points s and t may not exist. Again, if monotone descent path from s to t exists, then $\pi_{md}(s,t)$ may not coincide with $\pi_{geo}(s,t)$.

Definition 3. *Given a source point s on the surface of the terrain $\mathcal{T}$, the descent flow region of s (called $DFR(s)$) is the region on the surface of $\mathcal{T}$ such that each point $q \in DFR(s)$ is reachable from s through a monotone descent path.*

Theorem 1. *Given a polyhedral terrain $\mathcal{T}$ of n vertices, and a source vertex s, one can construct a data structure, called* **DFR***, in $O(n\log n)$ time and $O(n)$ space, which can answer the existence of a monotone descent path from s to a query point t on the surface of $\mathcal{T}$ in $O(\log n)$ time.*

3 Shortest Monotone Descent Path on Convex $DFR(s)$

Let f and f' be two adjacent faces of the terrain $\mathcal{T}$ sharing an edge e. The faces f and f' are said to be in convex (resp. concave) position if the angle between f and f' inside the terrain is less (resp. greater) than 180^0.

Given a terrain $\mathcal{T}$ and a source point s, $DFR(s)$ is said to be *convex* (resp. *concave*) if every pair of adjacent faces in $DFR(s)$ is in convex (resp. concave) position. Before going into the detail, we need the following two results which hold for arbitrary polyhedral terrain.

Result 1. *[8] For a pair of points α and β, if $\pi_{geo}(\alpha,\beta)$ passes through an edge-sequence $\mathcal{E}$ of a polyhedron, then in the planar unfolding $U(\mathcal{E})$, the path $\pi_{geo}(\alpha,\beta)$ is a straight line segment.*

Result 2. *Given a vertex s and a pair of points α and β on the terrain $\mathcal{T}$, $\pi_{md}(s,\alpha)$ and $\pi_{md}(s,\beta)$ can join, bifurcate or intersect at vertices of $\mathcal{T}$. Moreover, if a vertex v is common in both the paths then the length of the subpath from s to v on both $\pi_{md}(s,\alpha)$ and $\pi_{md}(s,\beta)$ are same.*

We now study the properties of a convex terrain, and propose an efficient algorithm for finding the shortest monotone descent path from s to a given query point $t \in DFR(s)$.

Observation 1. *If p_1, p_2 be two points on a face of $\mathcal{T}$, and p_3 is another point on the line segment $[p_1, p_2]$, then $z(p_1) > z(p_3)$ implies $z(p_2) < z(p_3)$.*

Lemma 1. *Let f and f' be two adjacent faces of a polyhedral terrain which are in convex position. The edge $e = [a, b]$ separates f and f'; $z(b) > z(a)$. Consider a point p on face f, and a point c on the edge e with $z(p) = z(c)$.*

(a) Now the edge e can be partitioned into two parts $[a, c]$ and $(c, b]$ such that there does not exist a monotone descent path from p to the face f' passing through the portion $(c, b]$ but such a path may exist which passes through the portion $[a, c] \in e$.

(b) Let q be a point in f' and q^* denote the image of the point q in the planar unfolding of f' onto f. Now, (i) if the line segment $[p, q^*]$ intersects the line segment $[a, c]$ $(\in e)$ in the unfolded plane, then $z(p) > z(q)$ and the geodesic shortest path from p to q through the edge e is the shortest monotone descent path from p to q, and (ii) if $[p, q^*]$ intersects the line segment $(c, b]$ and $z(p) > z(q)$ then $[p, c]+[c, q]$ will form the shortest monotone descent path from p to q through the edge e.

Proof: Part (a) of the lemma is trivial. We now prove part (b) of the lemma. Let $\pi_{geo}(p, q; e)$ denote the geodesic shortest path from p to q passing through the edge e. If the line segment $[p, q^*]$ (in the unfolded plane) intersects e (at a point, say η) in its interior, then by Result 1, the image of $\pi_{geo}(p, q; e)$ in the unfolded plane coincides with the line segment $[p, q^*]$. Now, two cases need to be considered.

$\mathbf{z}(\eta) \leq \mathbf{z}(\mathbf{p})$: By Observation 1, $z(q^*) < z(\eta)$. As the two faces f and f' are in convex position, $z(q) \leq z(q^*)$. Thus both the line segments $[p, \eta]$ and $[\eta, q]$ are monotone descent (see Fig. 1(a)), and part (i) of the lemma follows.

$\mathbf{z}(\eta) > \mathbf{z}(\mathbf{p})$: Here the line segment $[p, \eta]$ is not monotone descent in the plane f. Consider any monotone descent path from p to q which intersects the line segment $[a, c]$ (at a point, say η'). Note that, the length of such a path remains same as that of its image in the unfolded plane, and it attains minimum when $\eta' = c$ as illustrated in Fig. 1(b). This proves part (ii) of the lemma. □

Let v be a vertex of $\mathcal{T}$ and p be a point in $DFR(v)$ which is reachable from v through a sequence of edges $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$; the faces f_{i-1} and f_i, attached to edge e_i, are in convex position; $v \in f_0$, $p \in f_m$. Let R^* denote the region obtained by the planar unfolding $U(\mathcal{E})$, which is a polygonal region in the unfolded plane. By Result 1, we can prove the following lemma:

Lemma 2. *If p^* denotes the image of a point p in R^*, and the line segment $[v, p^*]$ completely lies inside R^*, then the path $\pi(v, p)$ on $\mathcal{T}$, whose image in R^* is the line segment $[v, p^*]$, is the shortest monotone descent path from v to p through the faces $\{f_0, f_1, f_2, \ldots, f_m\}$.*

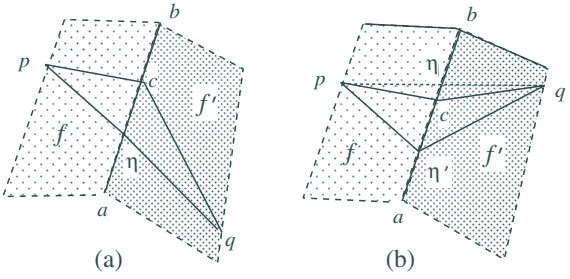


Fig. 1. Proof of Lemma 1

Definition 4. *If the shortest monotone descent path of a point p from a vertex v is obtained as in Lemma 2, then the point p is said to be straight-line reachable from the vertex v.*

The above discussions lead to a preprocessing step of $DFR(s)$ similar (but not exactly the same) to the method described in [8] for the problem of finding the geodesic shortest path from a fixed point to a given query point on the surface of a simple polyhedron. It (i) identifies the region $\mathcal{T}_s \in DFR(s)$ which is reachable from s through a sequence of faces in convex position, and then (ii) splits each face f of $\mathcal{T}_s$ into *homogeneous partitions* such that for every point p in a partition the shortest monotone descent path from s reaches p through the same edge sequence. Each homogeneous partition is a polygonal region on a single face and is bounded by the *h-segments* as defined below.

Definition 5. A segment $I = [a, b]$ on an edge e of $\mathcal{T}_s$ is said to be a *homogeneous segment* (or *h-segment* in short) if for every point $\alpha \in I$, the shortest monotone descent path from s to α passes through the same edge sequence. The end-points of an *h-segment* are referred to as *break-points*

Definition 6. *A point α on a line segment $[a, b]$ (portion of an edge) is said to be the frontier point with respect to a vertex v if α is straight line reachable from v through an edge sequence $\mathcal{E}$ and it is the closest point of v on the line segment $[a, b]$. It is easy to observe that α can be either a or b or the perpendicular projection of v on the line segment $[a, b]$ in the planar unfolding R^*.*

We follow *continuous Dijkstra* method to construct a hierarchical data structure *HDFR* for storing $\mathcal{T}_s$. It uses the break-points and frontier-points as event points. The leaf nodes of *HDFR* are the homogeneous partitions of the faces in $\mathcal{T}_s$, and non-leaf nodes are of two types: (i) *h-segment*s and (ii) vertices of $\mathcal{T}_s$. The root of *HDFR* is the vertex s. Each non-root node points to its predecessor in the hierarchy.

The query with respect to a point t is done by (i) locating a leaf node of *HDFR* whose corresponding face contains t using planar point location algorithm of [10], (ii) then it follows the predecessor links to report the edge sequence through which the shortest monotone descent path reaches from s to t. Following theorem

states the complexity results of finding the shortest monotone path of the query point t from the source point s.

Theorem 2. *Given a polyhedral terrain $\mathcal{T}$ with n vertices, and a source point s, our algorithm (i) creates the $HDFR$ data structure in $O(n^2\log n)$ time and $O(n^2)$ space, and (ii) for a given query point t, it outputs a shortest monotone descent path from s to t through a sequence of faces in convex position (if exists) in $O(k+\log n)$ time, where k is the number of line segments on the optimal path.*

4 Shortest Monotone Descent Path Through Parallel Edges

In this section, we shall consider a slightly different problem on a general terrain where each pair of adjacent faces are not restricted to only in convex position. Here source (s) and destination (t) points are given along with a sequence of faces $\mathcal{F} = \{f_0, f_1, \ldots, f_m\}$, $s \in f_0$, $t \in f_m$, and the objective is to find the shortest monotone descent path through $\mathcal{F}$. This problem in its general form seems to be difficult, and an efficient heuristic algorithm will be proposed in the next section. We now develope an efficient algorithm for finding the optimal solution in a restricted setup of this problem where the edge sequence $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$, separating the consecutive faces in $\mathcal{F}$, are parallel to each other. Note that, here we are deviating from the assumption that the faces in the terrain are triangular.

4.1 Properties of Parallel Edge Sequence

Lemma 3. *Let p and q be two points on two consecutive members e_i and e_{i+1} of $\mathcal{E}$ which bound a face f, and $z(p) = z(q)$. Now, if a line ℓ on face f is parallel to the line segment $[p, q]$, then (i) the length of the portion of ℓ lying in face f is equal to the length of the line segment $[p, q]$, and (ii) all the points on ℓ have same z-coordinate.*

Lemma 4. *Let e_i and e_{i+1} be two edges in $\mathcal{E}$ bounding a face f. For a pair of points $p, p' \in e_i$ and a pair of points $q, q' \in e_{i+1}$, if $z(p) > z(p')$ and $z(q) > z(q')$, then the line segments $[p, q]$ and $[p', q']$ do not intersect on face f; but the line segments $[p, q']$ and $[p', q]$ must intersect on face f.*

Theorem 3. *Let f_1 be a non-horizontal plane bounded by two parallel edges $e_1 = [a_1, b_1]$ and $e_2 = [a_2, b_2]$ ($z(a_i) < z(b_i)$, $i = 1, 2$); the point s appears in its adjacent face f_0 such that f_0 and f_1 are separated by the edge e_1. If there exists a pair of points $p \in e_1$ and $q \in e_2$ with $z(p) = z(q) < z(s)$, and s, p, q^* (q^* is the image of the point q in the planar unfolding $U(e_1)$) are collinear, then*

(i) for any point α in the interval $[q, a_2]$, shortest monotone descent path along e_1 is the inverse-image of the straight line segment $[s, \alpha^*]$ in the unfolded plane provided $[s, \alpha^*]$ intersects the edge e_1 in its interior.

(ii) for any point α in the interval $[b_2, q]$, shortest monotone descent path along e_1 is not an inverse-image of the straight line segment $[s, \alpha^*]$ in unfolded

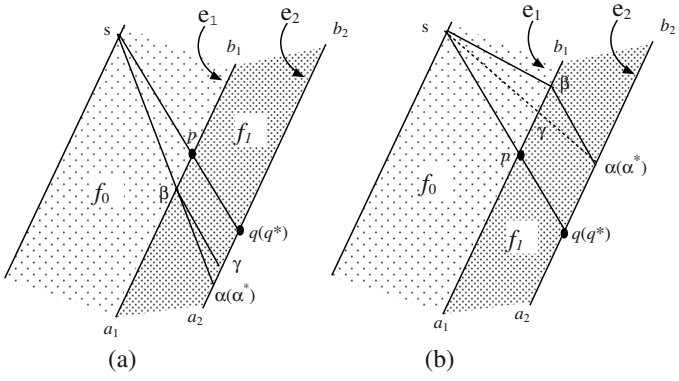


Fig. 2. Proof of Theorem 3

plane. Here $\pi_{md}(s, \alpha)$ will pass through a point $\beta \in [b_1, p]$ with $z(\beta) = z(\alpha)$ in the original terrain.

Proof: The line segment $[p, q]$ partitions the face f_1 into two parts, and the points b_1 and b_2 belong to the same side of $[p, q]$ (by Lemma 4). Consider a point $\alpha \in [q, a_2]$ on the edge e_2 (see Fig. 2(a)). In the planar unfolding $U(e_1)$, the straight line segment $[s, \alpha^*]$ intersects e_1 at a point, say β. The line segment $[\alpha, \beta]$ is below the line segment $[p, q]$ as shown in Fig. 2(a). Thus, if β is in the interior of the edge e_1 then $\beta \in [p, a_1]$. Let us consider a line segment $[\beta, \gamma]$ on the face f_1 which is parallel to $[p, q]$, and γ is on the edge e_2. Now consider the triangle $\Delta sq^*\alpha^*$ in the unfolded plane, where the point β lies on $[s, \alpha^*]$. As the line segment $[\beta, \gamma^*]$ is parallel to $[s, q^*]$, γ lies on $[q^*, \alpha^*]$. So, $z(\alpha) < z(\gamma) < z(q)$. By Lemma 3, $z(\gamma) = z(\beta)$. Hence part (i) of the lemma follows.

The proof of part (ii) follows from the following argument. Consider a point $\alpha \in [q, b_2]$ (See Fig. 2(b)); the line segment $[s, \alpha^*]$ intersects the edge e_1 at γ in the unfolded plane $U(e_1)$. Draw a line segment $[\alpha, \beta]$ on face f_1 which is parallel to $[p, q]$. As $z(\beta) = z(\alpha)$ (by Lemma 3), we have $z(\gamma) < z(\alpha)$. Thus, the shortest monotone descent path from s to α can not be the geodesic shortest path between them. As $z(\beta) = z(\alpha) < z(s)$, the shortest monotone descent path from s to α will be the concatenation of line segments $[s, \beta]$ and $[\beta, \alpha]$. □

We obtain the planar unfolding of the faces $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ onto face f_0, and use a two dimensional coordinate system for the entire unfolded plane such that the members in the edge-sequence $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ are ordered from left to right, and each of them is parallel to the y-axis. Each point in the unfolded plane is associated with the z-coordinate of the corresponding point in the original terrain. In this planar unfolding, let us represent an edge e_i of the terrain as $[a_i, b_i]$, with $y(a_i) < y(b_i)$. Now, $z(a_1) \leq z(b_1)$ then $z(a_i) \leq z(b_i)$ for all i (see Lemma 4). The source s is in f_0, then the monotone descent path passes through the edge sequence $\mathcal{E}$ to reach a point $t \in f_m$. If a path $\pi(s, t)$

enters into a face f_i along a line ℓ_i, then the *angle of incidence* of $\pi(s,t)$ in face f_i, denoted by θ_i, is the slope of the line ℓ_i in the unfolded plane.

Let e_1 and e_2 be two parallel boundaries of a face f. The *translation event* for face f, denoted by $T(f)$ is a linear translation of e_2 on e_1 such that the entire face f is merged to the line e_1 as follows:

- The points in the unfolded plane lying on the same side of s with respect to e_1 remain unchanged.
- Each point p lying in the proper interior of the face f is mapped to a point $q \in e_1$ such that $z(p) = z(q)$.
- Each point $p = (x_p, y_p)$ on the edge e_2 is mapped to a point $q = (x_q, y_q)$ on the edge e_1 such that $z(p) = z(q)$. Under this transformation $x_q = x_p + \alpha$, $y_q = y_p + \beta$, where the tuple (α, β) are constant, and they depend on the slope and width of the face f.
- Each point (x, y) in the unfolded plane lying on the other side of s with respect to e_2 is moved to the point $(x + \alpha, y + \beta)$.

The slope of the line containing (p, q) is referred to as *merging angle* of face f, and is denoted as $\phi(f)$. Theorem 3 indicates the following result.

Corollary 3.1. If the slope θ of a line segment ℓ in face f is such that (i) $\theta < \phi(f)$ then ℓ is strictly monotone descent, (ii) $\theta = \phi(f)$ then all points in ℓ have same z-coordinate, and (iii) $\theta > \phi(f)$ then ℓ is strictly monotone ascent.

Let $\pi(s,t)$ be the shortest monotone descent path from $s \in f_0$ to $t \in f_m$ passing through a sequence of parallel edges $\{e_1, e_2, \ldots, e_{m-1}\}$. Along this path there exists a set of faces $\{f_{j_i}, i = 1, 2, \ldots, k\}$ such that all the points of the path $\pi(s,t)$ in face f_{j_i} have same z-coordinate ξ_{j_i}; the portions of the path in all other faces are strictly monotone descent. Now, we have the following theorem.

Theorem 4. *If the translations $T(f_{j_1}), T(f_{j_2}), \ldots, T(f_{j_k})$ are applied (in any order) on the unfolded plane of faces $f_0, f_1, \ldots, f_m$ then the shortest monotone descent path $\pi(s,t)$ will become a straight line segment from s to t in the transformed plane.*

Proof: Let us first assume that $k = 1$, i.e., $\pi(s,t)$ passes through a face f with all points having same z-coordinate. Let f_a and f_b be its preceding and succeeding faces with separating edges e_a and e_b respectively. We also assume that $\pi(s,t)$ consists of three consecutive line segments $[s,a]$, $[a,b]$, $[b,t]$ lying in f_a, f and f_b respectively. Note that, all the points on $[a,b]$ have same z-coordinate. If we apply $T(f)$, the points b and t will be mapped to a and t'. Now, in the transformed plane, the shortest path from s to t' is the straight line segment $[s,t']$. We argue that $[s,t']$ will pass through a. On the contrary, assume that $[s,t']$ intersect e_a at a', and a' is the image of $b' \in e_b$ under $T(f)$, $b' \neq b$. Thus, $\overline{[s,a']} + \overline{[a',t']} < \overline{[s,a]} + \overline{[a,t']}$, where $\overline{[p,q]}$ indicates the length of the line segment $[p,q]$. Now, applying reverse transformation, $\overline{[s,a']} + \overline{[b',t]} < \overline{[s,a]} + \overline{[b,t]}$. From Lemma 3, $\overline{[s,a']} + \overline{[a',b']} + \overline{[b',t]} < \overline{[s,a]} + \overline{[a,b]} + \overline{[b,t]}$. This leads to a contradiction.

Let there exist several faces on the path $\pi(s,t)$ such that all the points of $\pi(s,t)$ in that face have same z-coordinate. If we apply T transformation on one face at a time, the above result holds. The order of choosing the face for applying the transformation T is not important due to the following argument: (i) a point p on the unfolded plane will be affected due to same set of transformation irrespective of in which order they are applied, and (ii) the effects of all the transformations affecting on a point are additive. □

Lemma 5. *If the shortest monotone descent path $\pi(s,t)$ is passing through a sequence of parallel edges, then all the line segments of $\pi(s,t)$, which are strictly monotone descent, are parallel on the unfolded plane of all faces.*

Theorem 5. *If the line-segments of shortest monotone descent path $\pi(s,t)$ in faces $f_{1^*}, f_{2^*}, \ldots, f_{k^*}$ are strictly monotone then their slopes are equal. The slope of the portions of $\pi(s,t)$ in all other faces are equal to the merging angle of the corresponding faces.*

4.2 Algorithm

Step 1: We compute the planar unfolding where the faces $f_1, f_2, \ldots, f_m$ are unfolded onto face f_0 containing s. We assume that the entire terrain is in first quadrant, and all the edges of $\mathcal{T}$ are parallel to the y-axis.

Step 2: We compute the merging angle for all the faces $f_i, i = 1, 2, \ldots, m$, and store them in an array Φ in ascending order. Each element contains its face-id.

Step 3: (* Merging phase*) Let θ be the slope of the line joining s and t in the unfolded plane. We sequentially inspect the elements of the array Φ from its first element onwards until an element $\Phi[k] > \theta$ is obtained. For each element $\Phi[i]$, $i < k$, the translation event takes place, and we do the following: Let $\Phi[i]$ correspond to a face f. We transform the entire terrain by merging the two boundaries of face f, i.e., compute the destination point t under the translation. The face f is marked. We update θ by joining s with the new position of t.

Compute the optimum path in transformed plane by the line joining s and t.

Step 4: The value θ, after the execution of Step 4, corresponds to the slope of the path segments which are strictly monotone along $\pi_{md}(s,t)$. We start from the point s at face f_0, and consider each face $f_i, i = 1, 2, \ldots, m$ in order. If face f_i is not marked, $\pi(s,t)$ moves in that face along a line segment of slope θ; otherwise, $\pi_{md}(s,t)$ moves along a line-segment of slope $\Phi[i]$.

Step 5: Finally, report the optimum path $\pi_{md}(s,t)$.

4.3 Correctness and Complexity Analysis of the Algorithm

Given a sequence of m faces of a polyhedral terrain bounded by parallel lines, and two query points s and t, Steps 1 and 2 of the algorithm computes the merging angles and sorts them in $O(m\log m)$ time. Step 3 needs $O(1)$ time. Each iteration of Step 4 needs $O(1)$ time, and we may need $O(n)$ such iterations for reporting the shortest monotone descent path from s and t. Thus we have the following theorem stating the time complexity result of our proposed algorithm.

Theorem 6. *Our algorithm correctly computes the shortest monotone descent path between two query points s and t through a sequence of faces of a polyhedral terrain bounded by parallel edges in $O(m\log m)$ time.*

5 Shortest Monotone Descent Path for General Terrain

Without loss of generality, we assume that for an edge $e_i = [a_i, b_i] \in \mathcal{T}$ $z(a_i) \leq z(b_i)$. In order to ensure a monotone descent path from s to t through an edge sequence $\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$, $z(b_i) \geq z(a_{i+1})$ for all $1 \leq i \leq k$. The following cases give an intution of the concept of *height level map* of [3].

$z(a_i) \geq z(a_{i+1})$ **&** $z(b_i) \geq z(b_{i+1})$**:** Here flow exists from any point of e_i on the edge e_{i+1}. Alternatively, any point of e_{i+1} is reachable from some point on e_i through a monotone descent path.

$z(a_i) < z(a_{i+1})$**:** In this case, monotone descent path does not exist from the edge segment $[a_i, \alpha) \in e_i$ to e_{i+1} where $z(\alpha) = z(a_{i+1})$.

$z(b_i) < z(b_{i+1})$**:** In this case, monotone descent path does not exist from e_i to the edge segment $(\beta, b_{i+1}] \in e_{i+1}$ where $z(\beta) = z(b_i)$.

The height level map introduces extra vertices (resp. edges) on the edges (resp. faces) of the terrain. The worst case number of such vertices and edges are both $O(n^2)$ [3]. These extra edges split few faces of the terrain; thus the terrain does not remain triangulated any more. We further triangulate the resulting terrain (with at most $O(n^2)$ vertices).

Definition 7. *A sequence of edges $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ is said to be* allowable *if every point on $e_1 \in \mathcal{E}$ is reachable from s using monotone descent path, from every point on the edge $e_m \in \mathcal{E}$, t is reachable using monotone descent path, and two consecutive edges in $e_i = [a_i, b_i]$, $e_{i+1} = [a_{i+1}, b_{i+1}] \in \mathcal{E}$ (sharing a common vertex) satisfies one of the following monotone descent properties.*

- *If $z(a_i) = z(b_i)$ and b_{i+1} is shared by e_i and e_{i+1}, then $z(a_{i+1}) \leq z(a_i)$.*
- *If $z(a_{i+1}) = z(b_{i+1})$ and a_i is shared by e_i and e_{i+1}, then $z(b_i) \geq z(b_{i+1})$.*
- *If $z(a_i) = z(a_{i+1})$ and $a_i \neq a_{i+1}$, then $b_i = b_{i+1}$.*
- *If $z(b_i) = z(b_{i+1})$ and $b_i \neq b_{i+1}$, then $a_i = a_{i+1}$.*

Lemma 6. *There always exists a monotone descent path from s to t through any allowable edge sequence.*

Result 3. *If $\pi^a_{geo}(s,t)$ denotes the shortest path from s to t considering all possible allowable edge sequences and $\pi^a_{md}(s,t)$ is the shortest monotone descent path through that allowable edge sequence through which $\pi^a_{geo}(s,t)$ passes, then $\pi_{geo}(s,t) \leq \pi^a_{geo}(s,t) \leq \pi_{md}(s,t) \leq \pi^a_{md}(s,t)$.*

We can obtain $\pi^a_{geo}(s,t)$ by minor tailoring the algorithm [8] for computing $\pi_{geo}(s,t)$. Next, we obtain a very closed approximation of $\pi^a_{md}(s,t)$ using the following heuristic steps. We shall refer it as $\pi^{a*}_{md}(s,t)$.

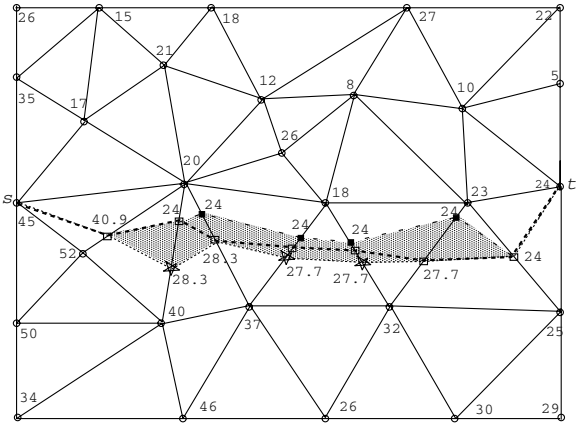


Fig. 3. A demonstrative example from [3]

Follow the path $\pi^a_{geo}(s,t)$ (along the allowable edge sequence $\mathcal{E}$) from s until a segment of the path is reached which is not monotone descent. From now onwards, traverse the remaining members of $\mathcal{E}$ maintaining equal height, until t is reached or a face (bounded by e_j and e_{j+1}) is encountered where this path crosses $\pi^a_{geo}(s,t)$. In the second case, we reach the point t' where $\pi^a_{geo}(s,t)$ intersects e_{j+1}. From, t' onwards, again follow $\pi^a_{geo}(s,t)$ down towards t. If the earlier situation occurs again, follow the same technique.

We apply the same technique, to compute a monotone ascent path from t to s, called $\pi^{a*}_{ma}(t,s)$. In Fig. 3, $\pi^{a*}_{md}(s,t)$, $\pi^{a*}_{ma}(t,s)$ and $\pi^a_{geo}(s,t)$ are shown using *dashed*, *dotted* and *bold* polylines respectively. We choose the shorter one among $\pi^{a*}_{md}(s,t)$ and $\pi^{a*}_{ma}(t,s)$ as the heuristic estimate of $\pi^a_{md}(s,t)$. The above algorithm is executed on the example of [3]. The length of $\pi^{a*}_{md}(s,t) = 242.5$, $\pi^{a*}_{ma}(t,s) = 230.8$, and $\pi^a_{geo}(s,t) = 223.7$.

Theorem 7. *The worst case time complexity of the proposed algorithm is* $O(n^4 \log n)$.

Proof: Follows from the fact that the number of Steiner vertices introduced by the height level map is $O(n^2)$, and (ii) the worst case running time of the shortest path algorithm in a polyhedral terrain of size n is $O(n^2 \log n)$ [8]. □

6 Conclusion

We have proposed polynomial time algorithms for finding the shortest monotone descent path in a polyhedral terrains in two special cases where (i) the path follows (i) a convex/concave face sequence, and (ii) a set of faces bounded by parallel edges. Our heuristic method for the general terrain first identifies the shortest path between the given pair of points (namely $\pi^a_{geo}(s,t)$) through an

allowable edge sequence. Next, it applies the heuristic method to find a monotone descent shortest path through the edge sequence where the shortest path has passed. The efficacy of our proposed method follows from the fact that it is very close to the shortest path between the given pair of points. The general problem is still unsolved since it is not known whether the actual shortest monotone descent path will pass through the aforesaid edge sequence. Even the shortest monotone descent path through an allowable edge sequence is difficult to compute.

References

1. P. K. Agarwal, S. Har-Peled and M. Karia, *Computing approximate shortest paths on convex polytopes*, Algorithmica, vol. 33, pp. 227-242, 2002.
2. L. Aleksandrov, A. Maheshwari and J.-R. Sack, *Approximation algorithms for geometric shortest path problems*, Proc. Symp. on Theory of Comput., pp. 286-295, 2000.
3. M. de Berg and M. van Kreveld, *Trekking in the Alps without freezing or getting tired*, Algorithmica, vol. 18, pp. 306-323, 1997.
4. B. Chazelle, *Efficient polygon triangulation*, Proc. of IEEE Symp. on Foundations of Computer Science, pp. 220–230, 1990.
5. J. Chen and Y. Han, *Shortest paths on a polyhedron*, Int. J. on Computational Geometry and Applications, vol. 6, pp. 127-144, 1996.
6. S. Kapoor, *Efficient computation of geodesic shortest paths*, Symp. on Theory of Computing, pp. 770-779, 1999.
7. M. Lanthier, A. Maheswari and J. -R. Sack, *Approximating weighted shortest paths on polyhedral surfaces*, Algorithmica, vol. 30, pp. 527-562, 2001.
8. J. S. B. Mitchell, D. M. Mount and C. H. Papadimitrou, *Discrete geodesic problem*, SIAM J. on Computing, vol. 16, pp. 647-668, 1987.
9. J. S. B. Mitchell and C. H. Papadimitrou, *The weighted region problem: Finding shortest paths through a weighted planar subdivision*, J. of the Association for Computing Machinary, vol. 38, pp. 18-73, 1991.
10. F. P. Preparata, *A new approach to planar point location*, SIAM J. Computing, vol. 10, pp. 473-482, 1981.
11. J.H. Reif and Z. Sun, *An efficient approximation algorithm for weighted region shortest path problem*, Proc. of the Workshop on Algorithmic Foundations of Robotics, pp. 191-203, 2000.
12. S. Roy, S. Das and S. C. Nandy, *A practical algorithm for approximating shortest weighted path between a pair of points on polyhedral surface*, Int. Conf. on Computational Science and Its Applications , pp. 42-52, 2004.
13. M. Sharir and A. Schorr, *On shortest paths in polyhedral space*, SIAM J. Computing, vol. 15, pp. 193-215, 1986.
14. K. R. Varadarajan, P. K. Agarwal, *Approximating shortest paths on a non-convex polyhedron*, SIAM J. Computing, vol. 30, pp. 1321-1340, 2001.

Packet Buffering: Randomization Beats Deterministic Algorithms

Markus Schmidt

Institut für Informatik, Albert-Ludwigs-Universität Freiburg,
Georges-Köhler-Allee 79, 79110 Freiburg, Germany
markus.schmidt@informatik.uni-freiburg.de

Abstract. We consider the problem of buffering unit value data packets in multi-queue network switches where each of the switch's input ports is equipped with a buffer of limited capacity. At these ports, packets arrive online and can be stored within the space limitations or must be discarded. Our objective is the maximization of the number of forwarded packets where, per time step, at most one packet from the set of buffers can be transmitted to the output port.

In this paper, we give a technique for transforming any randomized algorithm for unit buffers into a randomized algorithm for buffers with arbitrary capacities while maintaining the competitiveness. We present the first randomized online algorithm that beats the deterministic lower bound of $e/(e-1) \approx 1.58$. It is 3/2-competitive and thus nearly matches the randomized lower bound of 1.46. For buffers with 2 queues having large capacities, we show a lower bound of $16/13 \approx 1.23$ for any online algorithm and prove that the competitive ratio of greedy algorithms is $9/7 \approx 1.29$, improving the best previously known upper bound of 3/2.

1 Introduction

Due to the steady increase of traffic in today's networking systems, the fast and correct forwarding of data packets has become of major importance. As a result, switches that route packets arriving at the input ports to the appropriate output ports so that the packets can reach their resepective destinations have become critical elements for the performance of high-speed networks. Since data traffic may be bursty and packet loss is wished to be kept small, ports are equipped with buffers for temporary storage of the packets. The limitation of the buffer capacities entails the importance of effective buffer management strategies to maximize the throughput at a switch. Thus, the design and analysis of various buffer management policies have recently attracted considerable research interest [1–14].

In this framework, we consider a switch where for a given output port there are m serving input ports, each equipped with a buffer able to store up to B packets simultaneously and organized as a queue. At these input ports, new packets may arrive at any time step and can be appended to the respective buffers if space permits, otherwise the surplus ones must be dropped. During

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 293–304, 2005.

each time step the switch can select one populated queue and pass the packet at its head through the output port. Our goal is to maximize the *throughput*, i.e. the total number of forwarded packets. We thus consider all packets to be equally important, i.e. all of them have the same value. This model covers most current networks, particularly IP networks, for they treat packets from different data streams equally in their intermediate switches.

Since, usually, information on future packet arrivals is very limited or not available at all, we investigate an online setting where at any time future packet arrivals are unknown. Thus, we are interested in online buffer management strategies that have a provably good performance. Following [15], a deterministic online algorithm ALG is called c-competitive if $c \cdot\ T_{ALG}(\sigma) \geq T_{OPT}(\sigma)$ for all packet arrival sequences σ. Here $T_{ALG}(\sigma)$ and $T_{OPT}(\sigma)$ denote the throughputs achieved by ALG and, respectively, by an optimal offline algorithm OPT that knows the entire input σ in advance. If ALG is a randomized algorithm, then $T_{ALG}(\sigma)$ has to be replaced by the expected throughput $E[T_{ALG}(\sigma)]$.

Previous Work: Azar and Richter [4] showed that any work-conserving algorithm, which serves any non-empty queue, is 2-competitive and that for *unit buffers*, i.e. $B = 1$, no deterministic strategy can be better than $(2 - \frac{1}{m})$-competitive. They also considered randomization and presented a randomized algorithm that achieves a competitiveness of $e/(e-1) \approx 1.58$. Recently, Albers and Schmidt [2] gave a lower bound of 2 for the family of greedy algorithms, where it does not matter how ties are broken. They developed a variation of the greedy algorithm and showed that it is 1.89-competitive for $B \geq 2$. Moreover, they gave lower bounds of 1.46 and $e/(e-1)$ for randomized and deterministic algorithms, respectively.

A kind of dual problem is the question how much buffer space is needed in order not to lose any packets. In this context, Bar-Noy et al. [8] and Koga [14] studied buffer management policies when buffers have unlimited capacity and one wishes to minimize the maximum queue length. They presented $\Theta(\log m)$-competitive online strategies. Concerning the packet throughput, further results are known when packets have values and the goal is to maximize the total value of the transmitted packets. The focus of most of the previous work in this area has been on the single queue problem, i.e. the maintenance of only one buffer. Kesselman et al. [10] gave 2-competitive algorithms for various models allowing preemption. Recently, Bansal et al. [7] presented a 1.75-competitive algorithm when packets must be transmitted in the order they arrive. Aiello et al. [1] investigated single queue problems assuming that preemption is not allowed. For this scenario Andelman et al. [3] showed tight bounds of $\Theta(\log \alpha)$ where α is the ratio of the maximum to the minimum packet value.

Azar and Richter [4] presented a technique that transforms any c-competitive algorithm for a single queue into a $2c$-competitive algorithm for m queues. Using results from [3, 10] they derived 4-competitive preemptive and $2e\lceil \ln \alpha \rceil$-competitive non-preemptive algorithms. Recently, Azar and Richter [5] proved the zero-one principle for switching networks, saying that any comparison based

algorithm is c-competitive if and only if it is c-competitive for all sequences whose packet values are restricted to 0 and 1.

Our Contribution: We present a technique for transforming any randomized algorithm A for unit buffers into a randomized algorithm $\tilde{A}$ for multi-packet buffers while maintaining the competitiveness. If the multi-packet buffer consists of m queues, each able to buffer B packets, $\tilde{A}$ runs a simulation of A in a unit buffer with mB queues. The queues of the simulated unit buffer are partitioned into m sets, each consisting of B queues, and each such a set is assigned to one queue of the multi-packet buffer. The packets arriving at a queue of the multi-packet buffer are assigned in a round robin manner to the B queues assigned in the unit buffer. $\tilde{A}$ always serves that queue that the queue served by A is assigned to.

The main contribution of this paper is a new randomized algorithm, called *Random Permutation* (RP), for unit buffers that we prove to be 3/2-competitive. This result shows that randomized algorithms can beat the factor of $e/(e-1) \approx 1.5820$, being, as aforementioned, a lower bound for deterministic algorithms. The new RP algorithm chooses a permutation of the queues due to the uniform distribution. If there are populated queues, RP always serves that one that has the highest prority in the chosen permutation, i.e. that one being most to the front of the permutation vector. By applying the generalization technique described above to RP, we get a randomized algorithm $\tilde{RP}$ that is 3/2-competitive for all buffer sizes B.

In the lower bound constructions in previous work [2, 4], the number m of ports was assumed to be large compared to the buffer queue capacity B whereas, in practice, the port number is rather small and, thus, the packet capacity satisfies $B \gg m$. Moreover, greedy algorithms are very important in practice because they are fast, use little extra memory and reduce packet loss by always serving a longest queue. In the last part of the paper, we consider buffers with $m = 2$ queues, called *bicodal buffers*. For this setting, we show a lower bound of $16/13 \approx 1.2308$ for any online algorithm and prove that the competitive ratio of greedy algorithms is $9/7 \approx 1.2857$, improving the best previously known upper bound of $2 - 1/m = 3/2$ shown in [4].

This paper is organized as follows. In Section 2 we develop our generalization technique for unit buffer algorithms. The new RP algorithm is presented in Section 3. The analysis of the greedy algorithm for bicodal buffers is given in Section 4.

2 Generalizing Algorithms Without Loss of Competitiveness

Theorem 1. *If there is a deterministic (randomized) c-competitive algorithm A for $B = 1$, then there is a deterministic (randomized) c-competitive algorithm $\tilde{A}$ for all B.*

Proof. First, we consider the deterministic case. Let A be a deterministic c-competitive algorithm for $B = 1$. We use A in an algorithm $\tilde{A}$ for arbitrary B

and show that the competitiveness of $\tilde{A}$ cannot be worse than A's one. Let Γ_B be a buffer with m queues each able to buffer B packets and let Γ_1 be a buffer with mB queues each able to buffer one packet. We denote the queues in Γ_B and Γ_1 by $q_1, \ldots, q_m$ and $q_{1,0}, \ldots, q_{1,B-1}, \ldots, q_{m,0}, \ldots, q_{m,B-1}$, respectively. Moreover let $Q_i = \{q_{i,0}, \ldots, q_{i,B-1}\}$ for $1 \leq i \leq m$. Let σ be an arrival sequence for Γ_B and let σ_{it} denote the number of packets arriving at q_i at time step t. We transform σ into an arrival sequence $\tilde{\sigma}$ for Γ_1. Let $\zeta_{it} = \sum_{\tau \leq t} \sigma_{i\tau}$. In $\tilde{\sigma}$, there arrives one packet at $q_{i,(k \mod B)}$ for each $k = \zeta_{i,t-1} + 1, \ldots, \zeta_{it}$, i.e. the packets arriving at q_i are mapped to packets arriving at the queues in Q_i in a round robin manner with respect to the modulo B function.

Algorithm $\tilde{A}$:
For each time step t do:

- Arrival step:
  1. Accept as many packets as possible at each queue.
- Transmission step:
  2. Run a simulation of A on sequence $\tilde{\sigma}$.
  3. If, at time step t, A serves a queue in Q_i, serve q_i.
  4. Else if Γ_1 is empty, but there are still populated queues in Γ_B, serve one of them arbitrarily chosen.

Let OPT denote an optimal offline algorithm and let $T_{ALG}(\sigma)$ denote the packet throughput of algorithm ALG when processing sequence σ. We shall show that

$$T_{OPT}(\sigma) \leq T_{OPT}(\tilde{\sigma}) \tag{1}$$

$$T_{\tilde{A}}(\sigma) \geq T_A(\tilde{\sigma}). \tag{2}$$

This establishes our claim because $T_{OPT}(\sigma) \leq T_{OPT}(\tilde{\sigma}) \leq c \cdot T_A(\tilde{\sigma}) \leq c \cdot T_{\tilde{A}}(\sigma)$.

First, we prove that algorithm $\tilde{A}$ is well-defined, i.e. that if the condition in line 3 is satisfied, then q_i is populated. Let N_{it} and $\tilde{N}_{it}$ denote the numbers of packets buffered by $\tilde{A}$ at q_i in Γ_B and by A at Q_i in Γ_1, respectively, at the beginning of time step t. We show by induction on t that the inequality $N_{it} \geq \tilde{N}_{it}$ holds for each t. The claim is obvious for $t = 1$. Let it hold at the beginning of time step t. If $\tilde{A}$ accepts all σ_{it} arriving packets, the claim still holds because A accepts at most σ_{it} of them. Otherwise, there are exactly B packets in q_i after the arrival step, and at most B queues in Q_i can be populated. If q_i is not served at the transmission step of t, the claim is established for $t+1$. Otherwise, either a queue in Q_i is served and the number of packets decreases by 1 both in q_i and Q_i or Γ_1 is completely empty and $\tilde{N}_{it} = 0$. Since line 4 only affects $T_{\tilde{A}}(\sigma)$, we have inequality (2) as an immediate result.

Now, we consider OPT. We prove by induction on t that there exists a feasible schedule S for $\tilde{\sigma}$ such that $T_S(\tilde{\sigma}) = T_{OPT}(\sigma)$, establishing inequality (1) because $T_S(\tilde{\sigma}) \leq T_{OPT}(\tilde{\sigma})$. Let b_{it} denote the number of packets buffered by OPT at q_i at the beginning of time step t and let the next packet accepted by OPT at q_i

be mapped to $q_{i,k_{it}}$. We show by induction on t that there is a schedule S for $\tilde{\sigma}$ such that precisely b_{it} queues out of Q_i are populated in S at the beginning of time step t and that these queues form a consecutive (wrt. the modulo B function) block B_{it} that ends at $q_{i,((k_{it}-1) \mod B)}$, i.e. the populated queues in Q_i are $q_{i,((k_{it}-b_{it}) \mod B)}, \ldots, q_{i,((k_{it}-1) \mod B)}$. The claim is obvious for $t = 1$. Let it hold at the beginning of time step t. If OPT accepts all σ_{it} arriving packets, then $\sigma_{it} \leq B - b_{it}$. Since $q_{i,(k_{it} \mod B)}, \ldots, q_{i,((k_{it}+B-b_{it}-1) \mod B)}$ are empty, S accepts all the σ_{it} single packets arriving at $q_{i,(k_{it} \mod B)}, \ldots, q_{i,((k_{it}+\sigma_{it}-1) \mod B)}$. If there is an overflow at q_i, then OPT only accepts $B - b_{it}$ packets and these packets fit in the empty queues in Q_i in S. First, we consider the case that there are less than B packets in q_i after the arrival step. The next packet arriving at q_i is accepted by OPT and mapped to $q_{i,((k_{it}+\sigma_{it}) \mod B)}$, which is the first empty queue in Q_i beyond $B_{i,t+1}$. If OPT serves q_i at time step t, then S serves the queue at the beginning of B_{it}. Hence, the populated queues in Q_i still form a consecutive block and the claim holds for $t + 1$. Now, we consider the case of a buffer overflow. All B queues in Q_i are populated. If OPT does not serve q_i at time step t, the claim holds for $t + 1$ by putting the beginning of $B_{i,t+1}$ to $q_{i,(k_{i,t+1} \mod B)}$. Otherwise, S serves $q_{i,(k_{i,t+1} \mod B)}$, and the claim is established for $t + 1$ as well.

Finally, we consider the randomized case. Let R be a randomized c-competitive algorithm for $B = 1$. Hence, there is a probability space Ω with distribution P due to which R chooses deterministic algorithms for $B = 1$ out of $\{A(\omega) : \omega \in \Omega\}$. Above, we have shown that for each sequence σ for Γ_B we can construct a sequence $\tilde{\sigma}$ such that inequalities (1) and (2) are satisfied by each deterministic algorithm. Let $\tilde{R}$ be the randomized algorithm for buffer size B that chooses deterministic algorithms for B out of $\{\tilde{A}(\omega) : \omega \in \Omega\}$ due to the same probability distribution P where $\tilde{A}(\omega)$ is the algorithm obtained by applying the generalization technique above to $A(\omega)$. Since R is c-competitive, there holds

$$E[T_R(\tilde{\sigma})] = \int_{\Omega} T_{A(\omega)}(\tilde{\sigma}) dP \geq T_{OPT}(\tilde{\sigma})/c. \tag{3}$$

By using the fact that R and $\tilde{R}$ choose their underlying deterministic algorithms due to the same probability distribution P and subsequently using inequalities (2), (3) and (1), we eventually derive that

$$E[T_{\tilde{R}}(\sigma)] = \int_{\Omega} T_{\tilde{A}(\omega)}(\sigma) dP \geq \int_{\Omega} T_{A(\omega)}(\tilde{\sigma}) dP \geq T_{OPT}(\tilde{\sigma})/c \geq T_{OPT}(\sigma)/c. \qquad \square$$

3 A Random Permutation Algorithm

Algorithm RP: Let $\mathcal{P}$ be the set of permutations of $\{1, \ldots, m\}$. Choose $\pi \in \mathcal{P}$ according to the uniform distribution and fix it. In each transmission step choose among the populated queues that one whose index is most to the front in π.

We call a packet p arriving at queue q a *non-initialization packet* if p is the $i^{th}, i > B$, packet arriving there. Because of the generalization technique in Section 2, we shall henceforth assume that $B = 1$.

Lemma 1. *Let there be m queues and $B = 1$ buffer per queue and let $p_j^{(m)}$ denote the probability of RP's accepting the j^{th} non-initialization packet. If $i \leq m$, then*

$$p_i^{(m)} \geq \frac{1}{m!} \sum_{j=1}^{i} \binom{i}{j} (m-j)!(-1)^{j-1}.$$

Proof. We use the following notation: Let P_i be the i^{th} non-initialization packet and let P_i arrive at queue q_{a_i} at time t_i. We assume that OPT can accept each non-initialization packet. Hence OPT must have served q_{a_i} before t_i. Since OPT can serve at most one queue per unit of time, OPT can accept at most t non-initialization packets till time t, yielding $t_i \geq i$. The earlier P_i arrives, the greater is the probability that RP has not served q_{a_i} since the last packet arrival there. Furthermore, the greater the number of queues where at least one packet has already arrived, the less is p_i. Hence, we can assume –without loss of generality– that $t_i = i$ and that at least one packet has arrived at each queue. Henceforth, we shall restrict ourselves to arrival sequences σ satisfying this assumption. Since $i \leq m$, OPT can either decide $a_i = a_j$ for some $j < i$ or $a_i \neq a_j$ for all $j < i$.

The following two lemmata establish that the latter choice is more disadvantageous for RP and are stated without proofs due to space limitation.

Lemma 2. *There holds the following invariant: At each time t, there exists $1 \leq n_t \leq m$ such that in RP's configuration the queues $q_{\pi_1}, \ldots, q_{\pi_{n_t}}$ are empty after the transmission step at t, whereas the queues $q_{\pi_{n_t+1}}, \ldots, q_{\pi_m}$ are populated.*

We can interpret n_t as a separation mark in π, positioned between π_{n_t} and π_{n_t+1}, separating the empty queues from the populated ones.

Lemma 3. *Let $Q = \{q_1, \ldots, q_m\}$ be the set of queues. Let I_j^+ denote the set of queues where at least one of the first j non-initialization packets has arrived and let $I_j^- = Q \setminus I_j^+$. For $1 \leq i \leq m-1$, let p_{i+1}^+ and p_{i+1}^- be the probabilities of RP's accepting the $(i+1)^{st}$ non-initialization packet at a queue out of I_i^+ and I_i^-, respectively. There holds $p_{i+1}^- \leq p_{i+1}^+$.*

So, we shall consider an arrival sequence σ where, after the initial burst of one packet at each queue, one packet arrives at each unit of time and the first m non-initialization packets arrive at distinct queues. Without loss of generality, we may assume that $\sigma_{ij} = \delta_{ij}$ (Kronecker symbol) for all $1 \leq i \leq m$, i.e. at time step i, there is a single packet arriving at q_i. Henceforth, $p_i^{(m)}, i \leq m$, denotes the probability of RP's accepting P_i in this special σ.

In order to complete the proof of Lemma 1, we show another useful lemma, first.

Lemma 4. *There holds the following recursion:*

$$p_1^{(m)} = \frac{1}{m}, \quad p_i^{(m)} = p_{i-1}^{(m)} + \frac{1}{m}\left(1 - p_{i-1}^{(m-1)}\right) \text{ for } 1 < i \leq m.$$

Proof. Let $i = 1$. $p_1^{(m)}$ corresponds to the probability that $\pi_1 = 1$. Since π is chosen due to the uniform distribution, each queue has the same probability of leading π, hence $p_1^{(m)} = \frac{1}{m}$. q_i is empty at time i if it has been empty at time $i-1$ or it is served at time i. The former probability is given by $p_{i-1}^{(m)}$. Now we consider the serving of q_i at time i. This is the case if and only if the separation mark n in π is moved on at time i and reaches i. Since i is put to each position in π with the same probability, n reaches i with probability $\frac{1}{m}$, given that the mark moves. This move takes place if and only if $i-1$ is located beyond the mark at time $i-1$, i.e. q_i is preferred to q_{i-1} in π. Since we fix the position in π to n_{i-1}, the situation corresponds to a buffer consisting of queues $q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_m$ where the first $n_{i-1}-1$ positions in the permutation are the same, and the remaining ones except that for i are shifted one position to the head, i.e. $\pi'_k = \pi_k$ if $k \leq n_{i-1} - 1$, and $\pi'_k = \pi_{k+1}$ if $k \geq n_{i-1}$. The probability of $i-1$ there being located beyond the separation mark is the counter probability of $i-1$ there being served during the first $i-1$ time steps. Hence, the separation mark is moved on with probability $\left(1 - p_{i-1}^{(m-1)}\right)$. □

Now, we continue to prove Lemma 1. The claim is shown by induction on m. $p_1^{(1)} = 1 = \frac{1}{1!}\sum_{j=1}^{1}\binom{1}{j}(1-j)!(-1)^{j-1}$. Assume that the claim holds for $m-1$. From Lemma 4, we derive $p_1^{(m)} = \frac{1}{m} = \frac{(m-1)!}{m!} = \frac{1}{m!}\sum_{j=1}^{1}\binom{1}{j}(m-j)!(-1)^{j-1}$. Assume that the claim holds for $p_{i-1}^{(m)}$. If $i > 1$, we use the recursion formula from Lemma 4: $p_i^{(m)} = p_{i-1}^{(m)} + \frac{1}{m}\left(1 - p_{i-1}^{(m-1)}\right) = p_{i-1}^{(m)} + \frac{1}{m} - \frac{1}{m}p_{i-1}^{(m-1)}$. Since the claim holds for $m-1$, we deduce that

$$p_i^{(m)} = p_{i-1}^{(m)} + \frac{(m-1)!}{m!} - \frac{1}{m}\frac{1}{(m-1)!}\sum_{j=1}^{i-1}\binom{i-1}{j}((m-1)-j)!(-1)^{j-1}$$

$$= p_{i-1}^{(m)} + \frac{1}{m!}\sum_{j=0}^{i-1}\binom{i-1}{j}(m-1-j)!(-1)^{j} = p_{i-1}^{(m)} + \frac{1}{m!}\sum_{j=1}^{i}\binom{i-1}{j-1}(m-j)!(-1)^{j-1}.$$

Using the claim for $p_{i-1}^{(m)}$ gives us

$$p_i^{(m)} = \frac{1}{m!}\sum_{j=1}^{i-1}\binom{i-1}{j}(m-j)!(-1)^{j-1} + \frac{1}{m!}\sum_{j=1}^{i}\binom{i-1}{j-1}(m-j)!(-1)^{j-1}$$

$$= \frac{1}{m!}\sum_{j=1}^{i-1}\left(\binom{i-1}{j} + \binom{i-1}{j-1}\right)(m-j)!(-1)^{j-1} + \frac{1}{m!}\binom{i-1}{i-1}(m-i)!(-1)^{i-1}.$$

The application of Pascal's formula eventually completes our proof:

$$p_i^{(m)} = \tfrac{1}{m!}\sum_{j=1}^{i-1}\binom{i}{j}(m-j)!(-1)^{j-1} + \tfrac{1}{m!}\binom{i}{i}(m-i)!(-1)^{i-1}$$
$$= \tfrac{1}{m!}\sum_{j=1}^{i}\binom{i}{j}(m-j)!(-1)^{j-1}. \qquad \square$$

Due to space limitation, the proofs of the following three lemmata are omitted.

Lemma 5. *If $m \geq 3$, then $\frac{5}{8} \leq p_m^{(m)} \leq \frac{2}{3}$.*

Lemma 6. *If $1 \leq i \leq m$, then the throughput ratio of RP after i non-initialization packets is less than $\frac{3}{2}$.*

Lemma 7. *Let $\beta_m = \sum_{i=1}^{m} p_i^{(m)}$. There holds $\beta_m = (m+1)\sum_{j=0}^{m+1}\frac{(-1)^j}{j!}$.*

Theorem 2. *The competitive ratio of RP is at most $\frac{3}{2}$.*

Proof. If $m \leq 2$, the claim is obvious because *RP* is a work-conserving algorithm and each work-conserving algorithm is $(2-\frac{1}{m})$-competitive, as shown in [4]. Hence we may assume $m \geq 3$. The expected number of hits for *RP* during the first m time steps is β_m. So, the expected number of faults is $m - \beta_m$. Since there cannot be more than $m-1$ faults –each fault results in one queue empty in *RP*'s configuration, but populated in *OPT*'s configuration–, the expected number of further faults is at most $\beta_m - 1$. Let there be n non-initialization packets. If $n \leq m$, our claim follows from Lemma 6. Let $n > m$. We partition the arrival sequence into σ^0 comprising the initialization packets and the first m non-initialization packets and σ^1 comprising the remaining packets. Let p_m denote $p_m^{(m)}$. Hence $p_i^{(m)} \geq p_m$ for $i > m$. Moreover let $T_0 = E[T_{RP}(\sigma^0)]$, $T_1 = E[T_{RP}(\sigma^1)], L_0 = 2m - T_0, L_1 = n - m - T_1, \varrho_m = \frac{2m}{m+\beta_m}$. L_0 and L_1 are the expected numbers of packets lost by *RP* when serving σ^0 and σ^1, respectively, ϱ_m is the throughput ratio after m non-initialization packets. Let $s_n = \sum_{j=0}^{n}\frac{(-1)^j}{j!}$. Note that $p_m^{(m)} = 1 - s_m$. By Lemma 5, $2s_m + s_{m+1} \geq 3 \cdot \frac{1}{3} = 1$. Hence, $m < m(2s_m + s_{m+1}) + s_{m+1}$, yielding $2m(1-s_m) < m + (m+1)s_{m+1}$. We derive that

$$\varrho_m p_m = \frac{2m(1-s_m)}{m+(m+1)s_{m+1}} < 1. \tag{4}$$

There holds $r := \frac{T_{OPT}(\sigma)}{E[T_{RP}(\sigma)]} = \frac{T_0+L_0+T_1+L_1}{T_0+T_1}$. Note that $T_0 + L_0 \leq \varrho_m T_0$, $T_1 \geq (n-m)p_m$, $L_1 \leq (1-p_m)(n-m)$ and $L_1 \leq \beta_m - 1$, hence $L_1 \leq \min\{(1-p_m)(n-m), \beta_m - 1\}$. We conclude that

$$r \leq \frac{\varrho_m T_0 + T_1 + L_1}{T_0 + T_1} = \varrho_m + \frac{L_1 - (\varrho_m - 1)T_1}{T_0 + T_1} \leq \varrho_m + \frac{L_1 - (\varrho_m - 1)p_m(n-m)}{m + \beta_m + p_m(n-m)}.$$

First, we consider $n \leq m + \frac{\beta_m - 1}{1-p_m}$. Then, $r \leq \varrho_m + \frac{(1-p_m)(n-m) - (\varrho_m - 1)p_m(n-m)}{m+\beta_m+p_m(n-m)} = \varrho_m + \frac{(1-\varrho_m p_m)(n-m)}{m+\beta_m+p_m(n-m)}$, which is increasing in n because $1 - \varrho_m p_m \geq 0$ due to inequality (4). Hence

$$r \le \varrho_m + \frac{(1-\varrho_m p_m)\frac{\beta_m-1}{1-p_m}}{m+\beta_m+p_m\frac{\beta_m-1}{1-p_m}}.$$

Now, we consider $n \ge m + \frac{\beta_m-1}{1-p_m}$. Then, $r \le \varrho_m + \frac{\beta_m-1-(\varrho_m-1)p_m(n-m)}{m+\beta_m+p_m(n-m)}$, which is decreasing in n. Hence

$$r \le \varrho_m + \frac{\beta_m-1-(\varrho_m-1)p_m\frac{\beta_m-1}{1-p_m}}{m+\beta_m+p_m\frac{\beta_m-1}{1-p_m}} = \varrho_m + \frac{(1-\varrho_m p_m)\frac{\beta_m-1}{1-p_m}}{m+\beta_m+p_m\frac{\beta_m-1}{1-p_m}}.$$

By using the definition of ϱ_m, we deduce that in either case

$$r \le \frac{\varrho_m(m+\beta_m+p_m\frac{\beta_m-1}{1-p_m}-p_m\frac{\beta_m-1}{1-p_m})+\frac{\beta_m-1}{1-p_m}}{m+\beta_m+p_m\frac{\beta_m-1}{1-p_m}} = \frac{\varrho_m(m+\beta_m)+\frac{\beta_m-1}{1-p_m}}{m+\beta_m+p_m\frac{\beta_m-1}{1-p_m}}$$
$$= \frac{2m+\frac{\beta_m-1}{1-p_m}}{m+\beta_m+p_m\frac{\beta_m-1}{1-p_m}} = 2 - \frac{\beta_m-2p_m+1}{m(1-p_m)+\beta_m-p_m}.$$

Let $\delta_n = s_n - s_{n-1}$. Due to Lemma 5, $s_m \ge 0$. Hence $(m+1)\delta_{m+1}+s_m \ge 0$ if $\delta_{m+1} \ge 0$. Assume that $\delta_{m+1} < 0$. Then, $\delta_{m+1} = -\frac{1}{(m+1)!}$ and, using inequality $|s_m - \frac{1}{e}| \le \frac{1}{m!}$, $(m+1)\delta_{m+1}+s_m = s_m - \frac{1}{m!} \ge (\frac{1}{e}-\frac{1}{m!})-\frac{1}{m!} = \frac{1}{e}-\frac{2}{m!} \ge \frac{1}{e}-\frac{1}{3} > 0$. Hence, due to Lemma 7,

$$\beta_m = (m+1)s_{m+1} = (m+1)(s_m+\delta_{m+1}) = ms_m + (m+1)\delta_{m+1} + s_m \ge ms_m.$$

Since, due to Lemma 5, $p_m \le \frac{2}{3}$, we derive that $\beta_m + 2 \ge ms_m + 3p_m = m(1-p_m)+3p_m$, yielding $2(\beta_m - 2p_m + 1) \ge m(1-p_m)+\beta_m-p_m$. Using this inequality for r results in

$$r \le 2 - \frac{\beta_m-2p_m+1}{m(1-p_m)+\beta_m-p_m} \le 2 - \frac{1}{2} = \frac{3}{2}. \qquad \square$$

The following theorem is an immediate result of Theorem 1 and Theorem 2.

Theorem 3. *Let $\tilde{RP}$ be the randomized algorithm resulting from the application of the generalization technique to algorithm RP. $\tilde{RP}$ is $\frac{3}{2}$-competitive for any buffer size B.*

4 Bicodal Buffers

We consider the unit value packet throughput maximization problem for the case of $m = 2$ queues each having buffer size $B \gg m$.

The following theorem is stated without proof due to space limitation.

Theorem 4. *The competitive ratio of each deterministic online algorithm ALG is at least $\frac{16}{13} \approx 1.2308$.*

Algorithm Greedy: The greedy algorithm GR always serves the queue currently having maximum load. Ties are broken arbitrarily.

In our analysis we assign to each request sequence σ the sequence $l = (l_1, l_2, \ldots)$ of packet losses where l_i denotes the number of packets lost by GR during its i^{th} packet loss. We shall deduce lower bounds for the number of packets transmitted by GR for any loss sequence and thus derive an upper bound for the competitive ratio by assuming that the optimal offline algorithm OPT can accept all arriving packets. GR's transmission schedule can be partitioned in several phases where each phase ends when all of GR's queues are empty and the subsequent phase starts upon the next packet arrival. If we postpone this arrival until OPT has emptied its queues as well, the competitive ratio does not decrease. Since the total throughput ratio is bounded above by the greatest phase throughput ratio, it is sufficient only to consider sequences σ where not until σ has terminated does GR's buffer become unpopulated. Henceforth, σ and l always denote such an arrival sequence and its loss sequence, respectively. Let $S(l)$ denote $\sum_{i=1}^{n} l_i$. Moreover, let T_G and T_O denote the throughputs of GR and OPT, respectively.

Due to space limitation, the proof of the following lemma is omitted.

Lemma 8. *If σ yields $n \geq 3$ losses $l_1, \ldots, l_n$, we can transform it to $\tilde{\sigma}$ with $n-1$ losses $\tilde{l}_1, \ldots, \tilde{l}_{n-1}$ such that $\frac{T_O(\sigma)}{T_G(\sigma)} \leq \frac{T_O(\tilde{\sigma})}{T_G(\tilde{\sigma})}$.*

Corollary 1. *It is sufficient to consider sequences with at most 2 losses.*

Lemma 9. *If there is a unique packet loss l_1, then*

1. $T_G \geq B + 2l_1$
2. $l_1 \leq B/2$
3. $T_O \leq \frac{5}{4} T_G$.

Proof. Wlog. we may assume that the packet loss occurs in q_1. Let g_k and h_k denote the loads of GR and OPT in q_k, $k \in \{1, 2\}$, just before this loss. From the imminent packet loss, we derive that $g_1 = h_1 + l_1$ and, hence, $g_2 = h_2 - l_1$. Hence, there are l_1 time steps $t_1 < \ldots < t_{l_1}$ where GR serves q_2 while OPT serves q_1. There holds $g_{2,t_j} \geq g_{1,t_j}$. In each time step after the last different service, OPT serves the same queue as GR does and the load difference l_1 is kept. We consider t_{l_1}: there holds $g_1 = h_1 + l_1, g_2 = h_2 - l_1, g_2 \geq g_1$. Hence, $g_2 \geq h_1 + l_1 \geq l_1$. Since GR has served q_2 at least l_1 times before, at least $2l_1$ packets must have arrived at q_2. Due to the packet loss in q_1, GR has accepted at least B packets in q_1, establishing claim 1. Claim 2 results from the fact that $B \geq h_2 = g_2 + l_1 \geq 2l_1$, yielding $\frac{T_O}{T_G} = 1 + \frac{l_1}{T_G} \leq 1 + \frac{l_1}{B+2l_1} = \frac{B+3l_1}{B+2l_1} = \frac{3}{2} - \frac{B}{2B+4l_1} \leq \frac{3}{2} - \frac{B}{2B+2B} = \frac{5}{4}$. □

Lemma 10. *If there are 2 losses l_1 and l_2, then $T_O \leq \frac{9}{7} T_G$.*

Proof. First, we consider the case that the two losses occur in different queues. Since $l_2 \leq (B - l_1)/2$, there holds $r(l) = \frac{l_1+l_2}{B+3l_1+l_2} = 1 - \frac{B+2l_1}{B+3l_1+l_2} \leq 1 - \frac{B+2l_1}{B+3l_1+(B-l_1)/2} = 1 - \frac{2B+4l_1}{3B+5l_1} = \frac{B+l_1}{3B+5l_1} = \frac{1}{5} + \frac{2B}{15B+25l_1}$. If $l_1 \geq B/3$, then $r(l) \leq \frac{1}{5} + \frac{2B}{15B+25l_1} \leq \frac{1}{5} + \frac{6B}{45B+25B} = \frac{1}{5} + \frac{3}{35} = \frac{2}{7}$. Now assume that $l_1 \leq B/3$. Since $B - l_1 - l_2 \leq l_1$ implies that $B \leq 2l_1 + l_2 \leq 2l_1 + (B - l_1)/2$ and, thus, $2B \leq B + 3l_1$, we conclude that $l_1 \leq B - l_1 - l_2$. Hence $2l_1 + l_2 \leq B$. Since $l_1 + 2l_2 \leq B$, we get $l_1 + l_2 \leq \frac{2}{3}B$, yielding $\frac{l_1+l_2}{B+2l_1+2l_2} = \frac{1}{2} - \frac{B}{2B+4(l_1+l_2)} \leq \frac{1}{2} - \frac{B}{2B+4\cdot\frac{2}{3}B} = \frac{1}{2} - \frac{3}{6+8} = \frac{7-3}{14} = \frac{2}{7}$. Hence

$$\frac{l_1 + l_2}{B + 2l_1 + 2l_2} \leq \frac{2}{7}. \tag{5}$$

We distinguish 3 cases:

1. $G_1 \leq l_1 \leq B - l_1 - l_2$:
   (a) $l_2 \leq G_1$: $r(l) = \frac{l_1+l_2}{B+3l_1+l_2} = 1 - \frac{B+2l_1}{B+3l_1+l_2} \leq 1 - \frac{B+2l_1}{B+4l_1} = \frac{2l_1}{B+4l_1} = \frac{1}{2} - \frac{B}{2B+8l_1} \leq \frac{1}{2} - \frac{B}{2B+\frac{8}{3}B} = \frac{1}{2} - \frac{3}{6+8} = \frac{7-3}{14} = \frac{2}{7}$.
   (b) $l_2 > G_1$: Here $l_2 - G_1$ additional packets have to be taken into account. Hence $r(l) \leq \frac{l_1+l_2}{(B+3l_1+l_2)+(l_2-G_1)} = \frac{l_1+l_2}{B+2l_1+2l_2+(l_1-G_1)} \leq \frac{l_1+l_2}{B+2l_1+2l_2} \leq \frac{2}{7}$ due to inequality (5).
2. $l_1 \leq G_1 \leq B - l_1 - l_2$:
   (a) $l_2 \leq l_1$: like Subcase 1a.
   (b) $l_1 \leq l_2 \leq G_1$: Here, $G_1 - l_1$ additional packets have to be taken into account. Hence $r(l) \leq \frac{l_1+l_2}{(B+3l_1+l_2)+(G_1-l_1)} = \frac{l_1+l_2}{B+2l_1+l_2+G_1} \leq \frac{l_1+l_2}{B+2l_1+2l_2} \leq \frac{2}{7}$ due to inequality (5).
   (c) $l_1 \leq G_1 \leq l_2$: Here, $(G_1-l_1)+(l_2-G_1) = l_2-l_1$ additional packets have to be taken into account. Hence $r(l) \leq \frac{l_1+l_2}{(B+3l_1+l_2)+(l_2-l_1)} = \frac{l_1+l_2}{B+2l_1+2l_2} \leq \frac{2}{7}$ due to inequality (5).
3. $l_1 \leq B-l_1-l_2 \leq G_1$: Here, $G_1-l_1 \geq B-l_1-l_2-l_1 = B-2l_1-l_2$ additional packets have to be taken into account. Hence $r(l) \leq \frac{l_1+l_2}{(B+3l_1+l_2)+(B-2l_1-l_2)} = \frac{l_1+l_2}{2B+l_1} \leq \frac{l_1+(B-l_1)/2}{2B+l_1} = \frac{B+l_1}{4B+2l_1} = \frac{1}{2} - \frac{B}{4B+2l_1} \leq \frac{1}{2} - \frac{B}{4B+\frac{2}{3}B} = \frac{1}{2} - \frac{3}{12+2} = \frac{7-3}{14} = \frac{2}{7}$.

Now, we consider the case that the two losses occur in the same queue. If $l_1 \geq B/3$, the claim is established as in the different queues case. Hence, we may assume that $l_1 \leq B/3$. If $G_2 > l_1$, $G_2 - l_1$ additional packets must be taken into account. So must $l_2 - G_2$ ones if $l_2 > G_2$. We distinguish 4 cases:

1. $l_2 \leq G_2 \leq l_1$: like Subcase 1a of different queues.
2. $l_1 \leq G_2, l_2 \leq G_2$:
   (a) $l_2 \leq l_1$: like Subcase 1a of different queues.
   (b) $l_1 \leq l_2$: like Subcase 2b of different queues.
3. $G_2 \leq l_1, G_2 \leq l_2$: like Subcase 1b of different queues.
4. $l_1 \leq G_2 \leq l_2$: like Subcase 2c of different queues. □

Combining Lemma 8, Lemma 9 and Lemma 10 yields the following theorem.

Theorem 5. *GR is* $\frac{9}{7}$*-competitive.*

Theorem 6. *The competitive ratio of GR is* $\frac{9}{7} \approx 1.2857$.

Proof. Let $t_0 = 1, t_1 = 1 + B/3, t_2 = 1 + B$ and $\sigma_{t_0} = (B/3, 2B/3), \sigma_{t_1} = (B, 0)$, $\sigma_{t_2} = (0, B)$, and $\sigma_t = (0, 0)$ for all $t \notin \{t_0, t_1, t_2\}$. □

References

1. W. Aiello, Y. Mansour, S. Rajagopolan and A. Rosén, Competitive queue policies for differentiated services . *Proc. INFOCOM*, 431–440, 2000.
2. S. Albers and M. Schmidt. On the performance of greedy algorithms in packet buffering. *Proc. 36th ACM Symposium on Theory of Computing*, 35–44, 2004.
3. N. Andelman, Y. Mansour and A. Zhu. Competitive queueing policies in QoS switches. *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms*, 761–770, 2003.
4. Y. Azar and Y. Richter. Management of multi-queue switches in QoS Networks. *Proc. 35th ACM Symposium on Theory of Computing*, 82–89, 2003.
5. Y. Azar and Y. Richter. The zero-one principle for switching networks. *Proc. 36th ACM Symposium on Theory of Computing*, 64–71, 2004.
6. A. Aziz, A. Prakash and V. Ramachandran. A new optimal scheduler for switch-memory-switch routers. *Proc. 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures* 343– 352, 2003.
7. N. Bansal, L.K. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber and M. Sviridenko. Further Improvements in Competitive Guarantees for QoS Buffering. *Proc. 31st International Colloquium on Automata, Languages and Programming*, Springer LNCS, Volume 3142, 196 - 207 , 2004.
8. A. Bar-Noy, A. Freund, S. Landa and J. Naor. Competitive on-line switching policies. *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, 525–534, 2002.
9. E.L. Hahne, A. Kesselman and Y. Mansour. Competitive buffer management for shared-memory switches. *Proc. 13th ACM Symposium on Parallel Algorithms and Architectures*, 53–58, 2001.
10. A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber and M. Sviridenko. Buffer overflow management in QoS switches. *Proc. 31st ACM Symposium on Theory of Computing*, 520–529, 2001.
11. A. Kesselman and Y. Mansour. Loss-bounded analysis for differentiated services. *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms*, 591–600, 2001.
12. A. Kesselman, Y. Mansour and R. van Stee. Improved competitive guarantees for QoS buffering *Proc. 11th European Symposium on Algorithms*, Springer LNCS, Volume 2832, 361–372, 2003.
13. A. Kesselman and A. Rosén. Scheduling policies for CIOQ switches. *Proc. 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 353–361, 2003.
14. H. Koga. Balanced scheduling towards loss-free packet queueing and delay fairness. *Proc. 12th Annual International Symposium on Algorithms and Computation*, 61–73, 2001.
15. D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202-208, 1985.

Solving Medium-Density Subset Sum Problems in Expected Polynomial Time

Abraham D. Flaxman[1] and Bartosz Przydatek[2]

[1] Department of Mathematical Sciences,
Carnegie Mellon University, Pittsburgh, PA 15213, USA
abie@cmu.edu
[2] Department of Computer Science,
ETH Zurich, 8092 Zurich, Switzerland
przydatek@inf.ethz.ch

Abstract. The subset sum problem (SSP) (given n numbers and a target bound B, find a subset of the numbers summing to B), is a classic NP-hard problem. The hardness of SSP varies greatly with the density of the problem. In particular, when m, the logarithm of the largest input number, is at least $c \cdot n$ for some constant c, the problem can be solved by a reduction to finding a short vector in a lattice. On the other hand, when $m = \mathcal{O}(\log n)$ the problem can be solved in polynomial time using dynamic programming or some other algorithms especially designed for dense instances. However, as far as we are aware, all known algorithms for dense SSP take at least $\Omega(2^m)$ time, and no polynomial time algorithm is known which solves SSP when $m = \omega(\log n)$ (and $m = o(n)$).

We present an expected polynomial time algorithm for solving uniformly random instances of the subset sum problem over the domain $\mathbb{Z}_M$, with $m = \mathcal{O}((\log n)^2)$. To the best of our knowledge, this is the first algorithm working efficiently beyond the magnitude bound of $\mathcal{O}(\log n)$, thus narrowing the interval of hard-to-solve SSP instances.

1 Introduction

The subset sum problem (SSP), one of the classical NP-hard problems, is defined as follows: given n numbers and a target bound B, find a subset of the numbers whose sum equals B.

In this paper, we consider a case arising commonly in cryptographic applications where the numbers are represented by m-bit integers, and the sums are computed modulo M, where M is another m-bit integer. In other words, the addition is performed in $\mathbb{Z}_M$. More formally, the subset sum problem of dimensions n and m is:

GIVEN: n numbers $a_1, \ldots, a_n$, with $a_i \in \mathbb{Z}_M$, and a target $B \in \mathbb{Z}_M$, where M is an m-bit integer
FIND: a subset $S \subset \{1, \ldots, n\}$, such that

$$\sum_{i \in S} a_i \equiv B \mod M \,.$$

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 305–314, 2005.

We focus on *random* instances of the problem, where both the input numbers and the bound are picked uniformly at random. Similar random instances (with different parameters than we will eventually select) were shown by Chvatal [Chv80] to be hard instances for a class of knapsack algorithms.

The hardness of random SSP instances varies significantly with the choice of parameters, in particular the magnitude of m as a function of n (cf. [IN96]):

$m > n$**:** such instances are "almost 1-1" (each subset has a different sum), and are efficiently solvable by a reduction to a short vector in a lattice when $m \geq c \cdot n$, for some constant c [LO85, Fri86, CJL$^+$92].

$m < n$**:** such instances are "almost onto" (with multiple solutions for most targets), and are efficiently solvable by various techniques in high-density case, i.e., for $m = \mathcal{O}(\log n)$ (e.g., by dynamic programming, or using methods of analytical number theory [CFG89, GM91]).

Despite various efficient approaches to dense instances, as far as we are aware, all these algorithms take at least $\Omega(M)$ time, and so none of them works in polynomial time when $m = \omega(\log n)$.

1.1 Contributions

In this work we propose an expected polynomial time algorithm which solves uniformly random SSP instances with m up to $(\log n)^2/16$. Our algorithm starts by dividing the input instance into small, efficiently solvable subinstances. The solutions of subinstances lead to a reduced instance (simpler than the original input), which we solve recursively. Finally, the solution of the reduced instance is combined with the solutions of subinstances to yield a solution of the original instance.

To the best of our knowledge, this is the first algorithm working efficiently beyond the magnitude bound of $\mathcal{O}(\log n)$, thus narrowing the interval with hard-to-solve SSP instances.

1.2 Related Work

Our algorithm bears some similarity to an approach developed by Blum *et al.* [BKW03] in the context of computational learning theory. By employing a recursive approach much like ours, they provide an algorithm for learning an XOR function in the presence of noise.

Beier and Vöcking [BV03] presented an expected polynomial time algorithm for solving random knapsack instances. Knapsack and subset sum have some compelling similarities, but the random instances considered there are quite different from ours, and this leads to the development of quite a different approach.

1.3 Notation and Conventions

A tuple $(a_1, \ldots, a_n; B, M)$ denotes an instance of SSP with input numbers a_i and target B to be solved over $\mathbb{Z}_M$.

For the clarity of presentation we occasionally neglect the discrete nature of some terms in summations to avoid the use of rounding operations (floors and ceilings). However, this simplification does not compromise the validity of our results. All asymptotic notation is with respect to n, we write $f(n) \sim g(n)$ to mean $f(n)/g(n) \to 1$ as $n \to \infty$. All logarithms are base 2.

2 The New Algorithm

We begin with a special case, an algorithm applicable when M is a power of 2. Then we present another special case, an algorithm applicable when M is odd. In general, we apply a combination of the two special cases. Given any modulus M we write $M = \bar{M} \cdot M'$, with $\bar{M} = 2^{\bar{m}}$ and M' odd. We use the first algorithm to reduce the original problem $(a_1, \ldots, a_n; B, M)$ to a problem $(a'_1, \ldots, a'_{n'}; B', M')$, and then use the second algorithm to solve the reduced problem.

In the algorithms below ℓ is a parameter whose value will later be set to $(\log n)/2$. For simplicity, the description presented below focuses on the core part of the algorithms, which can fail on some inputs. Later we show that the failures have sufficiently low probability so that upon failure we can run a dynamic programming algorithm (which takes exponential time) and obtain an *expected* polynomial time algorithm.

2.1 Subset Sum Modulo Power of 2

Given an instance $(a_1, \ldots, a_n; B, M)$, with $M = 2^m$ and $B \neq 0$, we transform it to an equivalent instance with target zero, i.e., $(a_1, \ldots, a_n, a_{n+1}; 0, M)$, where $a_{n+1} = M - B$ and we require that a valid solution contain this additional element a_{n+1}. To solve the target-zero instance we proceed as follows: we find among the input numbers a maximum matching containing a_{n+1}, where two numbers a_i, a_j can be matched if the sum $(a_i + a_j)$ has its ℓ least significant bits equal to zero, (in other words, if $(a_i + a_j) \equiv 0 \mod 2^\ell$.) From the matching we generate a "smaller" instance of SSP, which we solve recursively: given a matching of size s, $((a_{i_1}, a_{j_1}), \ldots, (a_{i_s}, a_{j_s}))$, where wlog. $a_{i_s} = a_{n+1}$, we generate an instance $((a_{i_1}+a_{j_1})/2^\ell, \ldots, (a_{i_s}+a_{j_s})/2^\ell; 0, 2^{m-\ell})$, and we require that a valid solution of this instance must contain the last element. Note that the instance to be solved recursively is indeed smaller. It has at most $(n+1)/2$ input numbers, and both the modulus and the input numbers are shorter by ℓ bits. When the recursion reaches the bottom, we extract a solution of the original problem in a straightforward way. Figure 1 presents the algorithm in pseudocode. Note that the algorithm returns a set $\mathcal{S}$ of disjoint subsets, where the the last subset is a solution to the input problem, and all remaining subsets sum up to zero modulo 2^m. These extra subsets are used in the combined algorithm in Sect. 2.3.

We remark that the above method can be used to solve instances of SSP with some other moduli, for example when M is a power of small primes, or when M is "smooth" (meaning the product of small primes). However, the method does not generalize easily to arbitrary moduli, and in particular gives no obvious

procedure SSPmod2$(a_1, \ldots, a_n, B, m, \ell)$
  $a_{n+1} := -B$
  $\mathcal{S} :=$ SSPmod2rec$(a_1, \ldots, a_{n+1}, m, \ell)$
  /** *wlog.* $\mathcal{S} = (S_1, \ldots, S_s)$ *and* $(n+1) \in S_s$ **/
  return $(S_1, \ldots, S_{s-1}, S_s \setminus \{n+1\})$

procedure SSPmod2rec$(a_1, \ldots, a_{n+1}; m, \ell)$
  $\mathcal{S} := ()$
  $V := \{1, \ldots, n, n+1\}$
  $E := \{(i,j) : (a_i + a_j) \equiv 0 \mod 2^\ell\}$
  $E' :=$ maximum matching in $G = (V, E)$ containing vertex $(n+1)$
  /** *wlog.* $E' = (e_1, \ldots, e_s)$*, with* e_s *containing* $(n+1)$ **/
  if E' is non-empty **then**
    if $\ell < m$ **then**
      $\forall e_k \in E'$, $e_k = (i_k, j_k)$, **let** $a'_k := (a_{i_k} + a_{j_k})/2^\ell$
      $\mathcal{S}' :=$ SSPmod2rec$(a'_1, \ldots, a'_s; m - \ell, \ell)$
      if $\mathcal{S}'$ is not empty **then**
        /** *wlog.* $\mathcal{S}' = (S'_1, \ldots, S'_t)$*, with each* $S'_i \subseteq \{1 \ldots s\}$*, and* $s \in S'_t$ **/
        $\forall S'_i \in \mathcal{S}'$ **let** $S_i := \bigcup_{e_k : k \in S'_i, e_k = (i_k, j_k)} \{i_k, j_k\}$
        $\mathcal{S} := (S_1, \ldots S_t)$
    else
      $\forall e_k \in E'$, $e_k = (i_k, j_k)$, **let** $S_k := \{i_k, j_k\}$
      $\mathcal{S} := (S_1, \ldots S_s)$
  return $\mathcal{S}$

Fig. 1. The proposed algorithm for solving dense SSP instances modulo a power of 2

way to handle a large prime modulus. In the next section we describe a different algorithm, which works with high probability for arbitrary *odd* moduli.

2.2 Subset Sum with an Odd Modulus

The algorithm for SSP with an odd modulus has on a high level the same strategy as the algorithm from the previous section, i.e., it successively reduces the size of the numbers by matching them in pairs. However, it differs in one significant detail. Instead of working on least significant bits, it zeros out the most significant bits at each step of the recursion.

Given an instance $(a_1, \ldots, a_n; B, M)$, with M odd and $B \neq 0$, we begin, as in the previous case, by transforming it to an equivalent instance with target 0. However, this time we use a different transformation. To each input number we add the value $\Delta := (-B/2^t) \mod M$, where $t = \lceil \log_2 M/\ell \rceil$, so the modified instance is $(a'_1, \ldots, a'_n; 0, M)$, where $a'_i = a_i + \Delta$.

Our motivation for making this transformation becomes clear when we reveal our plan to make sure that any solution returned by our algorithm contains exactly 2^t elements. Since the sum of the solution of the modified instance is

zero modulo M, the sum of the corresponding numbers in the original instance is B, as each number of the solution contributes an extra Δ to the sum and

$$2^t \cdot \Delta \equiv -B \mod M.$$

The fact that M is odd is required to ensure that such a Δ exists.

Now it is convenient to view elements from $\mathbb{Z}_M$ as numbers from the interval $I = \{-(M-1)/2, \ldots, (M-1)/2\}$, following the transformation

$$a \to \begin{cases} a, & \text{if } a \le (M-1)/2; \\ a - M, & \text{otherwise.} \end{cases} \tag{1}$$

Given a target-zero instance $(a'_1, \ldots, a'_n; 0, M)$ with M odd, we find a solution of cardinality 2^t as follows: we find a maximum matching among the input numbers, where two numbers a'_i, a'_j can be matched iff there exists an integer k so that when viewed as elements of the interval I, as in (1), $a'_i \in [kM/2^{\ell+1}, (k+1)M/2^{\ell+1}]$ and $a'_j \in [-(k+1)M/2^{\ell+1}, -kM/2^{\ell+1}]$. Again, from the matching we generate a "smaller" instance of SSP, which we solve recursively. Given a matching of size s,

$$((a'_{i_1}, a'_{j_1}), \ldots, (a'_{i_s}, a'_{j_s})),$$

---

procedure SSPmodOdd$(a_1, \ldots, a_n; B, M, \ell)$
  $t := \lceil \log_2 M/\ell \rceil$
  $\Delta := (-B/2^t) \mod M$
  return SSPmodOddRec$(a_1 + \Delta, \ldots, a_n + \Delta; M, \ell, 1)$

procedure SSPmodOddRec$(a_1, \ldots, a_n; M, \ell, d)$
  /** *we view a_i's as numbers from $I = \{-(M-1)/2, \ldots, (M-1)/2\}$* **/
  $S := \{\}$
  $V := \{1, \ldots, n\}$
  $E := \{(i,j) : \exists k \in \mathbb{Z}, a_i \in [kM/2^{d\ell+1}, (k+1)M/2^{d\ell+1}],$
    $a_j \in [-(k+1)M/2^{d\ell+1}, -kM/2^{d\ell+1}]\}$
  $E' :=$ maximum matching in $G = (V, E)$
  /** *wlog.* $E' = (e_1, \ldots, e_s)$ **/
  if E' is non-empty **then**
    if $d \cdot \ell < \lceil \log_2 M \rceil$ **then**
      $\forall e_k \in E'$, $e_k = (i_k, j_k)$, **let** $a'_k := (a_{i_k} + a_{j_k})$
      $S' :=$ SSPmodOddRec$(a'_1, \ldots, a'_s; M, \ell, d+1)$
      if S' is not empty **then**
        /** $S' \subseteq \{1 \ldots s\}$ **/
        $S := \bigcup_{e_k : k \in S', e_k = (i_k, j_k)} \{i_k, j_k\}$
    else
      $S := \{i_1, j_1\}$, where $e_1 \in E'$, $e_1 = (i_1, j_1)$.
  return S

Fig. 2. The proposed algorithm for solving dense SSP instances with an odd modulus

procedure DenseSSP$(a_1, \ldots, a_n; B, M, \ell)$
  $S := \{\}$
  find M' and $\bar{M} = 2^{\bar{m}}$ such that M' is odd and $M = 2^{\bar{m}} \cdot M'$
  $\mathcal{S} :=$ SSPmod2$(a_1, \ldots, a_n, B, \bar{m}, \ell)$ /** *here* $\mathcal{S} = (S_1, \ldots, S_s)$, *with* **/
    /** $\sum_{j \in S_i} a_j \equiv 0 \mod \bar{M}$ *for* $i = 1..(s-1)$, *and* **/
    /** $\sum_{j \in S_s} a_j \equiv B \mod \bar{M}$ **/
  if $\mathcal{S}$ is not empty **then**
    $\forall i = 1..(s-1)$ **let** $a'_i := (\sum_{j \in S_i} a_j)/\bar{M}$
    $B' := B - \sum_{j \in S_s} a_j$
    $S' :=$ SSPmodOdd$(a'_1, \ldots, a'_{s-1}; B', M', \ell)$ /** *here* $S' \subseteq \{1, \ldots, n'\}$ **/
    if S' is not empty **then**
      $S := S_s \cup \left(\bigcup_{j \in S'} S_j\right)$
  return S

Fig. 3. The combined algorithm for solving dense SSP instances

we generate an instance $((a'_{i_1} + a'_{j_1}), \ldots, (a'_{i_s} + a'_{j_s}); 0, M)$. By the property of the matched numbers, the input numbers of the new instance are smaller in the sense that they are closer to 0 when viewed as elements of interval I. Figure 2 presents in pseudocode the algorithm for odd moduli.

2.3 Combined Algorithm

As mentioned above, given an instance $(a_1, \ldots, a_n; B, M)$, for any (m-bit) modulus M, we write $M = \bar{M} \cdot M'$, with $\bar{M} = 2^{\bar{m}}$ and M' odd, and apply both algorithms described above, one for $\bar{M}$ and one for M'.

First, we solve the instance $(a_1, \ldots, a_n; B, \bar{M})$ using procedure SSPmod2. As a solution, we obtain a sequence $\mathcal{S} = (S_1, \ldots, S_s)$ of disjoint subsets of $\{1, \ldots, n\}$, where for each $i = 1..(s-1)$ we have $\sum_{j \in S_i} a_j \equiv 0 \mod \bar{M}$, and $\sum_{j \in S_s} a_j \equiv B \mod \bar{M}$(i.e., the last subset is a solution for target B). From this solution we generate an instance for the second algorithm, $(a'_1, \ldots, a'_{n'}; B', M')$, where $n' = s - 1$, $a'_i = (\sum_{j \in S_i} a_j)/\bar{M}$ for $i = 1..n'$, and $B' = B - \sum_{j \in S_s} a_j$. The second algorithm returns a solution $S' \subseteq \{1, \ldots, n'\}$, from which we derive our answer

$$S_s \cup \left(\bigcup_{j \in S'} S_j \right)$$

Figure 3 presents the combined algorithm in pseudocode.

3 Analysis

3.1 Correctness

We need to argue that any non-empty subset returned by the algorithm is a valid solution.

First consider computation modulo $\bar{M}$ which is a power of 2. At each level of recursion we match pairs of input numbers so that ℓ least significant bits are zeroed, while respecting the constraint, that the last input number is matched. Therefore, in recursive call, we have zeroed the least significant bits of the resulting numbers, so it follows by induction that all the subsets returned by SSPmod2rec sum up to $0 \mod \bar{M}$.

Moreover, we need to argue that the last subset returned by SSPmod2rec determines a solution for the given target B. Indeed, if S_s is the last subset returned by SSPmod2rec, then $(n+1) \in S_s$ and $\sum_{i \in S_s} a_i \equiv 0 \mod \bar{M}$. Since $a_{n+1} = -B$, this implies that $\sum_{i \in S \setminus \{n+1\}} a_i \equiv B \mod \bar{M}$, as desired.

To prove the correctness of the computation modulo an odd number M', note that the transformation to a target-zero instance gives the desired result: any solution is created bottom-up, by first matching two input numbers, than matching two pairs matched previously, and so on, i.e., at each recursion level the number of the numbers in a solution is doubled, so the size of the solution subset is equal 2^t, where t is the depth of recursion, which, in turn, equals $\lceil \log_2 M'/\ell \rceil$. Therefore, any solution with target zero will have exactly $2^t \cdot \Delta \equiv -B \mod M'$ of "extra" sum, i.e., the corresponding original numbers sum up to $B \mod M'$.

Further, since the algorithm matches the numbers which have opposite signs but "close" magnitudes, at each level of recursion a portion of ℓ *most* significant bits is zeroed, while avoiding the problems of overflows and wrap-arounds when adding the numbers. Hence, by induction, the subset returned by SSPmodOddRec sums up to $0 \mod M'$.

The correctness of the combined argument follows from the above arguments and from the Chinese Reminder Theorem, since $M = \bar{M} \cdot M'$, where $\bar{M}$ and M' are relatively prime.

3.2 Success Probability

We consider the cases with magnitudes m up to $(\log n)^2/16$ and we set $\ell = (\log n)/2$. At a given level in the recursion, in both cases (power of 2 and odd), the success of the algorithm at that level depends on the number of numbers in the recursion being "enough". And, the number of numbers in the recursion at a given level is equal to the number of edges in the matching at the previous level. We will argue by induction. Let t_k denote the number of numbers available at the beginning of level k of the recursion, and let s_k denote the number of edges in the matching at level k.

Lemma 1. *For s_k and t_k defined as above, Let $\mathcal{A}_k$ denote the event that $s_k \geq t_k/4$. Then*

$$\Pr[\mathcal{A}_k \mid \mathcal{A}_1, \ldots, \mathcal{A}_{k-1}] \leq \exp\left(-n^{3/4}/32\right).$$

Proof. If $\mathcal{A}_1, \ldots, \mathcal{A}_{k-1}$ occur (meaning, in every previous level of the recursion, we have managed to keep at least $1/4$ of the numbers), then we begin level k with at least $n(1/4)^{(\log n)/8} = n^{3/4}$ numbers (since there are at most $m/\ell \leq (\log n)/8$ levels of recursion total).

Lemma 1 is easier to argue when the modulus is a power of 2. Then the subinstances are formed by zeroing least significant bits, and so the reduced numbers are independent and uniformly random. When the modulus is an odd, the reduced numbers are independent but not uniformly random. Fortunately, they are distributed symmetrically, in the sense that $\Pr[a'_i = a] = \Pr[a'_i = -a]$. We argue this by induction: Suppose $\Pr[a_i = a] = \Pr[a_i = -a]$ for all i. Then, since each edge $(i, j) \in E$ yields an $a'_k = a_i + a_j$, we have

$$\begin{aligned}\Pr[a'_k = a] &= \sum_b \Pr[a_i = b] \Pr[a_j = a - b] \\ &= \sum_b \Pr[a_i = -b] \Pr[a_j = -(a - b)] \\ &= \Pr[a'_k = -a].\end{aligned}$$

This symmetry property is all that we need to show s_k is very likely to exceed $t_k/4$. We can pretend the t_k input numbers are generated by a two-step process: first, we pick the absolute value of the numbers constituting the instance, and then we pick the sign of each number. Since the distribution is symmetric, in the second step each number in the instance becomes negative with probability 1/2.

Let T_i denote the number of numbers picked in the first step with absolute value in the interval $[(i-1)M/L^d, iM/L^d]$, where $L = 2^\ell$ and $i = 1 \dots L$. Then the number of *negative* numbers in interval i is a random variable $X_i \sim \mathrm{Bi}(T_i, 1/2)$, and we can match all but $Y_i := |X_i - (T_i - X_i)|$ numbers in interval i. Further,

$$\mathrm{E}[Y_i] = \sum_{k=1}^{T_i} \Pr[Y_i \geq k] = \sum_{k=1}^{T_i} \Pr[|X_i - T_i/2| \geq k/2],$$

and by Azuma's inequality, this is at most

$$\sum_{k=1}^{T_i} 2e^{-k^2/(2T_i)} \leq \int_{x=0}^{\infty} 2e^{-x^2} \sqrt{2T_i} dx = \sqrt{2\pi T_i}.$$

Let Y denote the total discrepancy of all the bins,

$$Y = \sum_{i=1}^{L} Y_i.$$

By linearity of expectation, we have that we expect to match all but

$$\mathrm{E}[Y] = \mathcal{O}(\sqrt{T_1} + \dots + \sqrt{T_L})$$

numbers. This sum is maximized when $T_1 = \dots = T_L$, and minimized when $T_i = t$ for some i (and all other T_j's are zero), hence

$$\mathcal{O}(\sqrt{t}) \leq \mathrm{E}[Y] \leq \mathcal{O}(\sqrt{tL}) \ . \tag{2}$$

Changing a single number in the instance can change the discrepancy by at most 2, so we use Azuma's inequality in a convenient formulation given by McDiarmiad [McD89] (see also Bollobás [Bol88]) and the fact that $L = 2^\ell = \sqrt{n}$ and $t \geq n^{3/4}$.

$$\begin{aligned}\Pr[s \leq t/4] &= \Pr[Y \geq t/2] \\ &\leq \Pr[Y \geq \mathrm{E}[Y] + t/4] \\ &\leq e^{-t/32} \\ &\leq \exp\left(-n^{3/4}/32\right).\end{aligned}$$

□

Then, in the case of an odd modulus, the failure probability is bounded by

$$\Pr[\text{failure}] \leq \sum_{k=1}^{m/\ell} \Pr[\mathcal{A}_k \mid \mathcal{A}_1, \ldots, \mathcal{A}_k] \leq (\log n) \exp -n^{3/4}/32 = \mathcal{O}(e^{-\sqrt{n}}).$$

If the modulus is a power of 2, we must also account for the possibility failure due to not matching the special number a_{n+1} at some stage. Let $\mathcal{B}_k$ denote the event that the special number is not matched at stage k. This type of failure only occurs if all $t_k - 1$ other numbers are different from the special number. Since the mod 2 reductions keep the numbers at stage k uniformly distributed among $m - k\ell$ possibilities, the probability of $\mathcal{B}_k$ given t_k is $\left(1 - \frac{1}{m-k\ell}\right)^{t_k - 1}$ and if $\mathcal{A}_1, \ldots, \mathcal{A}_{k-1}$ hold, this is at most $\exp\left(-(\log n)^{-2} n^{3/4}\right)$. So again, the probability of failure is $\mathcal{O}(e^{-\sqrt{n}})$.

3.3 Running Time

The running time of the algorithm above is dominated by the time required to solve all the subinstances, which is bounded by $(n-1)/(\ell-1) \cdot \mathcal{O}(2^\ell) = \mathcal{O}(n^{3/2})$.

In the case of failure, we can solve the instance by dynamic programming in time $O(2^{(\log n)^2})$. Since (when n is sufficiently large) the failure probability is much less than $2^{-(\log n)^2}$, combining the algorithm above with a dynamic programming backup for failures yields a complete algorithm that runs in expected polynomial time.

3.4 Choice of Parameters

The parameters above are not optimized, but there is a curious feature in the proof of Lemma 1 that puts a restriction on the range of coefficients c that would work for $m = c(\log n)^2$. Similarly, the range of constants c' that would work for $\ell = c' \log n$ is restricted in a way that does not seem natural. For $\ell = (\log n)/2$ and $m = (\log n)^2/16$, the number of stages of recursion is small enough that each stage has sufficiently many numbers to succeed with high probability. But for $\ell = (\log n)/2$ and $m = (\log n)^2/8$, McDiarmid's version of Azuma's inequality will not work in the way we have used it.

4 Conclusions and Open Problems

We presented an expected polynomial time algorithm for solving uniformly random subset sum problems of medium density over $\mathbb{Z}_M$, with m bounded by $\mathcal{O}((\log n)^2)$, where n is the number of the input numbers. As far as we are aware, this is the first algorithm for dense instances that works efficiently beyond the magnitude bound of $\mathcal{O}(\log n)$, thus narrowing the interval with hard-to-solve SSP instances. A natural open question is whether the bound on the magnitude can be further extended, e.g. up to $(\log n)^z$ for some $z > 2$.

Finally, recall that DenseSSP is a deterministic algorithm which can fail with non-zero probability. Since this probability is very low, upon failure we can run a dynamic programming algorithm and still obtain expected polynomial time in total. A different way of handling failures might be to run DenseSSP again on randomly permuted input. Note however that such multiple trials are not fully independent, thus complicating the analysis. It is an interesting problem to compare this alternative approach with the one we have analyzed.

Acknowledgement

We would like to thank Krzysztof Pietrzak for useful discussions.

References

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.

[Bol88] B. Bollobás. Martingales, isoperimetric inequalities and random graphs. In A. Hajnal, L. Lovász, and V. T. Sós, editors, *Combinatorics*, number 52 in Colloq. Math. Soc. János Bolyai, pages 113–139. North Holland, 1988.

[BV03] René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. In *Proc. 35th ACM STOC*, pages 232–241, 2003.

[CFG89] M. Chaimovich, G. Freiman, and Z. Galil. Solving dense subset-sum problems by using analytical number theory. *J. Complex.*, 5(3):271–282, 1989.

[Chv80] V. Chvatal. Hard knapsack problems. *Operations Research*, 28:1402–1411, 1980.

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Comput. Complex.*, 2(2):111–128, 1992.

[Fri86] Alan Frieze. On the Lagarias-Odlyzko algorithm for the subset sum problem. *SIAM J. Comput.*, 15(2):536–539, 1986.

[GM91] Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.*, 20(6):1157–1189, 1991.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, Fall 1996.

[LO85] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

[McD89] Colin McDiarmid. On the method of bounded differences. In *London Mathematical Society Lecture Note Series*, volume 141, pages 148–188. Cambridge University Press, 1989.

Quantified Constraint Satisfaction, Maximal Constraint Languages, and Symmetric Polymorphisms

Hubie Chen

Departament de Tecnologia, Universitat Pompeu Fabra, Barcelona, Spain
hubie.chen@upf.edu

Abstract. The constraint satisfaction problem (CSP) is a convenient framework for modelling search problems; the CSP involves deciding, given a set of constraints on variables, whether or not there is an assignment to the variables satisfying all of the constraints. This paper is concerned with the quantified constraint satisfaction problem (QCSP), a more general framework in which variables can be quantified both universally and existentially. We study the complexity of restricted cases of the QCSP where the types of constraints that may appear are restricted by a constraint language. We give a complete complexity classification of maximal constraint languages, the largest possible languages that can be tractable. We also give a complete complexity classification of constraint languages arising from symmetric polymorphisms.

1 Introduction

The *constraint satisfaction problem (CSP)* is widely acknowledged as a convenient framework for modelling search problems. An instance of the CSP consists of a set of variables, a domain, and a set of constraints; each constraint consists of a tuple of variables paired with a relation (over the domain) which contains permitted values for the variable tuple. The question is to decide whether or not there is an assignment mapping each variable to a domain element that satisfies all of the constraints. Alternatively, the CSP may be viewed as the problem of deciding, given an ordered pair of relational structures, whether or not there exists a homomorphism from the first structure to the second. Canonical examples of CSPs include boolean satisfiability and graph coloring problems.

All of the variables in a CSP can be viewed as being implicitly existentially quantified. A natural and useful generalization of the CSP is the *quantified constraint satisfaction problem (QCSP)*, where variables may be both universally and existentially quantified. An instance of the QCSP can be viewed as a game between two players which take turns setting variables occurring in a set of constraints; the question is to decide whether or not a specified player can always succeed in satisfying all of the constraints, despite the moves of the other player. While the CSP captures the complexity of deciding whether or not a combinatorial object of desirable type exists in a large search space, the QCSP

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 315–326, 2005.

is a prototypical PSPACE reasoning problem which captures the complexity of many problems involving *interaction* among multiple agents. Such problems arise naturally in a wide variety of domains, for example, combinatorics, logic, game theory, and artificial intelligence.

In their general formulation, the CSP and QCSP are intractable, being NP-complete and PSPACE-complete, respectively; however, it is possible to parameterize these problems by restricting the *constraint language*, or the types of constraints that are permitted in problem instances. This is the form of restriction which was studied by Schaefer in his now classic dichotomy theorem [30], and has seen intense investigation over the past decade in several different contexts. This paper continues the recently initiated study of the complexity of the QCSP under constraint language restrictions [3, 13, 14]. Our contributions are the complete classification of *maximal constraint languages*, as well as the complete classification of idempotent *symmetric polymorphisms*.

1.1 Background

Complexity Classification Theorems. In 1978, Schaefer proved that every constraint language over a two-element domain gives rise to a case of the CSP that is either in P or is NP-complete [30]. The non-trivial tractable cases given by this result are 2-SAT, Horn SAT, and XOR-SAT (where each constraint is a linear equation in the field with two elements). Over the past decade, many more complexity *dichotomy theorems* in the spirit of Schaefer's have been established [15], including dichotomies in the complexity of model checking for circumscription [26], "inverse" satisfiability [24], and computing an assignment maximizing the number of satisfied constraints [25]. All of these dichotomy theorems are for constraint languages over a two-element domain. Particularly relevant here is the dichotomy theorem for QCSP in domain size two [15, 16], which shows that the only tractable cases in this context are Quantified 2-SAT [1], Quantified Horn SAT [12], and Quantified XOR-SAT [15], reflecting exactly the non-trivial tractable constraint languages given by Schaefer's theorem. All other constraint languages give rise to a PSPACE-complete QCSP.

A considerable limitation of the mentioned classification theorems is that they only address constraint languages that are over a two-element domain. Extensions of these theorems that classify all constraint languages over *finite* domains would be extremely valuable. Such extensions would identify *all* of the ways tractability can arise from constraint language restrictions. Given a restricted case of the QCSP that is a candidate for being tractable, one would have the ability to immediately identify whether or not tractability of the case can be deduced from its constraint language; if not, it would follow that other features of the case need to be utilized in a proof of tractability. Indeed, the tractable and intractable cases identified by classification theorems give crisp theoretical results constituting an extremely convenient starting point for performing complexity analysis.

Much attention has recently been directed towards achieving a full classification theorem for the CSP, and has resulted in the identification of many new tractable cases of the CSP; see, for example, the papers [23, 19, 21, 18, 27, 9, 17,

11, 6, 7]. One spectacular result produced by this line of work is Bulatov's dichotomy theorem on CSP complexity classifying all constraint languages over a *three*-element domain [5], which resolved a question from Schaefer's original paper [30] that had been open for over two decades.

Algebra and Polymorphisms. A powerful algebraic approach to studying complexity and the relative expressiveness of constraint languages was introduced in [23, 20] and further studied, for example, in [21, 22, 18, 10, 9, 17, 11, 5, 6, 7]. In this approach, a dual perspective on the various constraint languages is given by studying the set of functions under which a constraint language is *invariant*, called the *polymorphisms* of a constraint language. In particular, sets of polymorphisms are linked to constraint languages (which are sets of relations) via a Galois connection, and two different constraint languages having the same polymorphisms give rise to cases of the CSP (and QCSP) with *exactly* the same complexity. The program of classifying constraint languages as either tractable or intractable can thus be rephrased as a classification question on polymorphisms; as it turns out, this rephrasing makes the classification program amenable to attack by insights and tools from universal algebra. This dual viewpoint was used heavily by Bulatov to obtain his dichotomy theorem [5], and can also be used in conjunction with Post's classification theorem [28, 2] to succinctly derive Schaefer's theorem: see, for example, [2].

Quantified Constraint Satisfaction. Recently, the issue of QCSP complexity based on constraint languages in domains of size greater than two was studied by Börner, Bulatov, Krokhin and Jeavons [3] and the present author [13, 14]. Both of these works used the algebraic approach in a central manner. The contributions of [3] included development of the algebraic viewpoint for the QCSP, the identification of some intractable and tractable classes of constraint languages, and a complete complexity classification theorem for a restricted class of constraint languages. The main contribution of [13] was general technology for demonstrating the tractability of constraint languages, while [14] studied the complexity of 2-semilattice polymorphisms.

1.2 Contributions of This Paper

In this paper, we study QCSP complexity by adopting the approach used to study CSP complexity in [9, 11, 4, 8]: we seek the most general tractability results possible by focusing on *maximal constraint languages*. Maximal constraint languages are those constraint languages that can express any constraint with the help of any relation not contained in the language. Because a constraint language that can express all relations is intractable, maximal constraint languages are the largest constraint languages that could possibly be tractable. The investigation of such languages has played a key role in understanding tractability in the CSP setting: all of the tractability results identified by Schaefer's theorem apply to maximal constraint languages, and the investigation of maximal constraint languages in domains of size larger than two has resulted in the identification of new tractable cases of the CSP [9, 11, 4, 8].

Classification of Maximal Constraint Languages. We give a *full classification theorem* on maximal constraint languages, showing that each gives rise to a case of the QCSP that is either polynomial-time decidable, or intractable (by which we mean NP-hard or coNP-hard). In order to obtain this theorem, we make use of a theorem of Rosenberg [29] from universal algebra which yields an algebraic description of maximal constraint languages, showing that any maximal constraint language has a polymorphism of one of five types. Most of the effort in obtaining our classification theorem is concentrated in studying one of the five types of maximal constraint languages, namely, those having a binary idempotent polymorphism. We remark that in the CSP setting, a classification of maximal constraint languages was only recently obtained [8], and there the difficult case was also this particular type.

Binary Idempotent Polymorphisms. Intriguingly, we show that maximal constraint languages invariant under a binary idempotent polymorphism give rise to *four* modes of behavior in the context of quantified constraint satisfaction. In our study, we consider such binary polymorphisms in two classes, those that act as a projection on some two-element domain, and those that do not. Those that do act as a projection give rise to cases of the QCSP that are either NP-complete or PSPACE-complete. Those that do not act as a projection can be assumed to be commutative, and give rise to cases of the QCSP that are either in P or coNP-hard; our demonstration of this fact generalizes the main result of [14]. We leave the exact complexity analysis of the coNP-hard cases as a fascinating issue for future research; we conjecture that these cases are contained in coNP, and are hence coNP-complete.

The fact that the binary polymorphisms investigated here fall into four different regimes of complexity can be contrasted with complexity results on the CSP, where all constraint languages that have been studied have been shown to be either in P or NP-complete. We believe that our results give evidence that, relative to study of the CSP, study of the QCSP is likely to require the utilization of a greater diversity of techniques, and be much richer from a complexity-theoretic standpoint.

Symmetric Polymorphisms and Set Functions. Our study of commutative binary polymorphisms is in fact carried out in a more general setting, that of idempotent symmetric polymorphisms. We *fully classify* idempotent symmetric polymorphisms, showing that for any such polymorphism, the corresponding case of the QCSP is either reducible to its CSP restriction or is coNP-hard, and that an algebraic criterion determines which of the two cases holds (Theorem 17).

Significantly, we show that the ideas used to study symmetric polymorphisms can be deployed to give a *full classification* of idempotent *set functions* in the QCSP. Set functions have been studied in CSP complexity [18] and guarantee tractability in the CSP setting. As with symmetric polymorphisms, we show that set functions beget cases of the QCSP that are either P or coNP-hard. This classification result resolves an open question naturally arising from [18], namely, to take the class of problems shown therein to be tractable in the CSP, and to give a complexity classification of these problems in the QCSP.

Algebra and Complexity. We believe our results to be beautiful examples of the fascinating interplay between algebra and complexity taking place in the setting of constraint satisfaction–an interplay that we feel to be deserving of more attention.

We remark that much of our study concerns *idempotent* polymorphisms. Idempotent polymorphisms give rise to constraint languages containing all *constant relations*, by which we mean arity one relations of size one. Such constraint languages have the desirable robustness property that for any (QCSP or CSP) instance over the constraint language, when a variable is instantiated with a value, the resulting instance is still over the constraint language.

2 Preliminaries

We use the notation $[n]$ to denote the set containing the first n positive integers, $\{1, \ldots, n\}$. We use t_i to denote the ith coordinate of a tuple $\bar{t}$.

2.1 Quantified Constraint Satisfaction

We now set the basic terminology of quantified constraint satisfaction to be used. Our definitions and notation are fairly standard, and similar to those used in other papers on (quantified) constraint satisfaction. Throughout, we use D to denote a domain, which here is a nonempty set of finite size.

Definition 1. *A* relation *(over D) is a subset of D^k. A* constraint *(over D) is an expression of the form $R(\overline{w})$, where R is a relation over D and $\overline{w}$ is a tuple of variables with the same arity as R. A* constraint language *is a set of relations, all of which are over the same domain.*

Intuitively, a constraint restricts the permissible values that can be given to a set of variables; the variable tuple specifies the variables that are restricted, while the corresponding relation specifies the values that the variable tuple may take on. Formally, we consider an arity k constraint $R(w_1, \ldots, w_k)$ to be *satisfied* under an interpretation f defined on the variables $\{w_1, \ldots, w_k\}$ if $(f(w_1), \ldots, f(w_k)) \in R$.

Definition 2. *A* quantified formula *is an expression of the form $Q_1 v_1 \ldots Q_n v_n \mathcal{C}$ where for all $i \in [n]$, Q_i is a quantifier from the set $\{\forall, \exists\}$ and v_i is a variable; and, $\mathcal{C}$ is a* constraint network*, that is, a finite set of constraints over the same domain, with variables from $\{v_1, \ldots, v_n\}$. A quantified formula is said to be over a constraint language Γ if every constraint in its constraint network $\mathcal{C}$ has relation from Γ.*

Note that, in this paper, we only consider quantified formulas without free variables. Truth of a quantified formula is defined just as in first-order logic; the constraint network $\mathcal{C}$ is interpreted as the conjunction of constraints it contains. The QCSP is the problem of deciding, given a quantified formula, whether or not it is true; the CSP can be defined as the restricted version of the QCSP where

all quantifiers appearing must be existential. We are interested in the following parameterized version of the QCSP.

Definition 3. *Let Γ be a constraint language. The* $\mathsf{QCSP}(\Gamma)$ *decision problem is to decide, given as input a quantified formula over Γ, whether or not it is true. We define the* $\mathsf{CSP}(\Gamma)$ *decision problem as the restriction of* $\mathsf{QCSP}(\Gamma)$ *to instances having only existential quantifiers.*

The present paper is a contribution to the long-term research goal of classifying the complexity of $\mathsf{QCSP}(\Gamma)$ for *all* constraint languages Γ.

2.2 Expressibility and Polymorphisms

We now explain how the set of relations expressible by a constraint language, and the polymorphisms of a constraint language, characterize the complexity of the constraint language. Our presentation is based on the papers [23, 20], to which we refer the reader for more information.

Definition 4. *(see [20] for details) When Γ is a constraint language over D, define $\langle\Gamma\rangle$, the set of relations* expressible *by Γ, to be the smallest set of relations containing $\Gamma \cup \{=_D\}$ and closed under permutation, extension, truncation, and intersection. (Here, $=_D$ denotes the equality relation on D.)*

The more relations that a constraint language Γ can express, the higher in complexity it is.

Proposition 5. *Let Γ_1, Γ_2 be constraint languages where Γ_1 is finite. If $\langle\Gamma_1\rangle \subseteq \langle\Gamma_2\rangle$, then* $\mathsf{QCSP}(\Gamma_1)$ *reduces to* $\mathsf{QCSP}(\Gamma_2)$.[1]

From Proposition 5, we can see that two finite constraint languages that express exactly the same relations are reducible to one another, and hence of the same complexity. (Up to certain technicalities that are not essential for understanding the new results of this paper, all of our discussion also holds for the case of infinite constraint languages.)

We now introduce the notion of polymorphism. An operation μ of rank k is a polymorphism of a relation R if, for any choice of k tuples $\overline{t_1}, \ldots, \overline{t_k}$ from R, the tuple obtained by acting on the tuples $\overline{t_i}$ in a coordinate-wise manner by μ, is also contained in R.

Definition 6. *An operation $\mu : D^k \to D$ is a* polymorphism *of a relation $R \subseteq D^m$ if for all tuples $\overline{t_1}, \ldots, \overline{t_k} \in R$, the tuple $(\mu(t_{11}, \ldots, t_{k1}), \ldots, \mu(t_{1m}, \ldots, t_{km}))$ is in R. An operation μ is a polymorphism of a constraint language Γ if μ is a polymorphism of all relations $R \in \Gamma$. When μ is a polymorphism of R (Γ), we also say that R (Γ) is* invariant *under μ.*

[1] Note that the only form of reduction we consider in this paper is many-one polynomial-time reduction.

We will be interested in the set of all polymorphisms of a constraint language Γ, as well as the set of all relations invariant under all operations in a given set.

Definition 7. *Let $\mathcal{O}_D$ denote the set of all finite rank operations over D, and let $\mathcal{R}_D$ denote the set of all finite arity relations over D.*

When $\Gamma \subseteq \mathcal{R}_D$ is a set of relations (that is, a constraint language), we define

$$\mathsf{Pol}(\Gamma) = \{\mu \in \mathcal{O}_D \mid \mu \text{ is a polymorphism of } \Gamma\}.$$

When $F \subseteq \mathcal{O}_D$ is a set of operations, we define

$$\mathsf{Inv}(F) = \{R \in \mathcal{R}_D \mid R \text{ is invariant under all operations } \mu \in F\}.$$

When f is a single operation, we use $\mathsf{Inv}(f)$ as notation for $\mathsf{Inv}(\{f\})$, and $\mathsf{QCSP}(f)$ ($\mathsf{CSP}(f)$) as notation for $\mathsf{QCSP}(\mathsf{Inv}(f))$ ($\mathsf{CSP}(\mathsf{Inv}(f))$).

Theorem 8. *For any constraint language Γ, it holds that $\langle\Gamma\rangle = \mathsf{Inv}(\mathsf{Pol}(\Gamma))$.*

From Theorem 8, we see that the complexity of a constraint language depends only on its polymorphisms, since its polymorphisms determine the set of relations that it can express, which as we have discussed, characterizes its complexity.

Proposition 9. *Let Γ_1, Γ_2 be constraint languages where Γ_1 is finite.*
If $\mathsf{Pol}(\Gamma_2) \subseteq \mathsf{Pol}(\Gamma_1)$, then $\mathsf{QCSP}(\Gamma_1)$ reduces to $\mathsf{QCSP}(\Gamma_2)$. Moreover, if both Γ_1 and Γ_2 are finite and $\mathsf{Pol}(\Gamma_1) = \mathsf{Pol}(\Gamma_2)$, then $\mathsf{QCSP}(\Gamma_1)$ and $\mathsf{QCSP}(\Gamma_2)$ are equivalent in that they reduce to one another.

3 Maximal Constraint Languages

We study the most general forms of constraint language restrictions by considering *maximal constraint languages*, the largest possible constraint languages that cannot express all relations.

Definition 10. *A constraint language Γ is* maximal *if $\langle\Gamma\rangle$ is not the set of all relations, but for any relation R not contained in Γ, $\langle\Gamma \cup \{R\}\rangle$ is the set of all relations.*

A theorem of Rosenberg [29] demonstrates that all maximal constraint languages are invariant under an operation of one of five particular forms. In order to state this theorem, we require some new terminology. An operation $f : D^k \to D$ is a *majority* operation if $k = 3$ and for all $a, a' \in D$ the equalities $f(a, a, a') = f(a, a', a) = f(a', a, a) = a$ hold; an *affine* operation if $k = 3$ and for all $a_1, a_2, a_3 \in D$ it is the case that $f(a_1, a_2, a_3) = a_1 * a_2^{-1} * a_3$ where $*$ is a binary operation and $^{-1}$ is a unary operation such that $(D, *, ^{-1})$ is an Abelian group; a *projection* if there exists $i \in [k]$ such that for all $a_1, \ldots, a_k \in D$, $f(a_1, \ldots, a_k) = a_i$; and a *semiprojection* if $k \geq 3$, f is not a projection, and there exists $i \in [k]$ such that for all $a_1, \ldots, a_k \in D$, $|\{a_1, \ldots, a_k\}| < k$ implies $f(a_1, \ldots, a_k) = a_i$.

Theorem 11. *(follows from [29]) If Γ is a maximal constraint language, then $\Gamma = \mathsf{Inv}(f)$ for an operation f having one of the following five types: a unary operation which is either a bijection or acts identically on its range, a semiprojection, a majority operation, an affine operation, a binary idempotent operation that is not a projection.*

The first two types of operations in Theorem 11 give rise to hard cases of the $\mathsf{QCSP}(\Gamma)$ problem.

Theorem 12. *Let Γ be a maximal constraint language of the form $\mathsf{Inv}(f)$, where f is a unary operation which is either a bijection or acts identically on its range. Then $\mathsf{QCSP}(\Gamma)$ is PSPACE-complete.*

Theorem 13. *[11] Let Γ be a maximal constraint language of the form $\mathsf{Inv}(f)$, where f is a semiprojection. Then $\mathsf{QCSP}(\Gamma)$ is NP-hard.*

The next two types of operations in Theorem 11 have previously been shown to be tractable.

Theorem 14. *[13, 3] Let Γ be a constraint language invariant under a majority operation or invariant under an affine operation. Then $\mathsf{QCSP}(\Gamma)$ is in P.*

The following is the statement of our classification theorem on maximal constraint languages.

Theorem 15. *(Classification of maximal constraint languages) Let Γ be a maximal constraint language. One of the following five conditions holds:*

- *$\mathsf{QCSP}(\Gamma)$ is in P.*
- *$\mathsf{QCSP}(\Gamma)$ is coNP-hard and $\Gamma = \mathsf{Inv}(f)$ for f a binary commutative idempotent operation that is not a projection.*
- *$\mathsf{QCSP}(\Gamma)$ is NP-complete.*
- *$\mathsf{QCSP}(\Gamma)$ is NP-hard and $\Gamma = \mathsf{Inv}(f)$ for f a semiprojection.*
- *$\mathsf{QCSP}(\Gamma)$ is PSPACE-complete.*

Proof. For the first four types of maximal constraint languages in Theorem 11, the result holds by Theorems 12, 13, and 14. Otherwise, consider a maximal constraint language $\Gamma = \mathsf{Inv}(f)$ for f a binary idempotent operation that is not a projection. Using a technique similar to that of the proof of [11–Lemma 3], it can be shown that $\Gamma = \mathsf{Inv}(f')$ for f' a binary idempotent operation such that, for every two-element subset $\{a, b\}$ of D, either $f'(a, b) = f'(b, a)$, or f' acts as a projection on $\{a, b\}$.

- If there is no two-element subset $\{a, b\}$ such that f' acts as a projection on $\{a, b\}$, the function f' is commutative and $\mathsf{QCSP}(\Gamma)$ either reduces to $\mathsf{CSP}(\Gamma)$ or is coNP-hard by Theorem 17 (Section 4). In the former case, $\mathsf{CSP}(\Gamma)$, and hence $\mathsf{QCSP}(\Gamma)$, is in P by [8–Theorem 5].
- Otherwise, there exists at least one two-element subset $\{a, b\}$ such that f' acts as a projection on $\{a, b\}$, and $\mathsf{QCSP}(\Gamma)$ is either NP-complete or PSPACE-complete by Theorem 22 (Section 5). □

4 Symmetric Polymorphisms

In this section, we develop algebraic theory that permits us to present a complete classification of idempotent symmetric polymorphisms in the QCSP, as well as a complete classification of idempotent set functions. We say that a polymorphism (or function) $f : D^k \to D$ is *symmetric* if for all $a_1, \ldots, a_k \in D$ and for all permutations $\pi : [k] \to [k]$, it holds that $f(a_1, \ldots, a_k) = f(a_{\pi(1)}, \ldots, a_{\pi(k)})$. Note that in this section, we consider symmetric polymorphisms of all arities, and not just binary polymorphisms.

Let $f : D^k \to D$ be an idempotent symmetric function. We say that an element $a \in D$ can *f-hit* an element $b \in D$ in one step if there exist elements $a_1, \ldots, a_{k-1} \in D$ such that $f(a, a_1, \ldots, a_{k-1}) = b$. Let us say that $a \in D$ can f-hit $b \in D$ (in m steps) if there exist elements $d_0, \ldots, d_m \in D$ such that $a = d_0$, $b = d_m$ and for all $i = 0, \ldots, m-1$ it holds that d_i can hit d_{i+1} in one step. Notice that the "can f-hit" relation is reflexive and transitive.

Define a set $C \subseteq D$ to be *coherent* with respect to f if it is nonempty and for all $a_1, \ldots, a_k \in D$, the following holds: if $\{a_1, \ldots, a_k\} \setminus C$ is nonempty, then $f(a_1, \ldots, a_k) \notin C$ (equivalently, if $f(a_1, \ldots, a_k) \in C$, then $a_1, \ldots, a_k \in C$). When $S \subseteq D$ is a nonempty set that is not coherent, if $\{a_1, \ldots, a_k\} \setminus S$ is nonempty and $f(a_1, \ldots, a_k) \in S$, we say that $(a_1, \ldots, a_k)$ is a *witness to the non-coherence of S*.

Observe that the union of any two coherent sets is also a coherent set. In addition, when C_1, C_2 are two coherent sets with a non-trivial intersection, their intersection $C_1 \cap C_2$ is also a coherent set. We use $\langle d \rangle$ to denote the smallest coherent set containing d.

Lemma 16. *All elements of $\langle d \rangle$ can hit d.*

Let us say that a coherent set is *minimal* if it is minimal (among coherent sets) with respect to the subset $\subseteq$ ordering. We show that whether or not an idempotent symmetric function f has a *unique* minimal coherent set in fact determines the complexity of $\mathsf{QCSP}(f)$.

Theorem 17. *Let f be an idempotent symmetric function. If there is a unique minimal coherent set with respect to f, then $\mathsf{QCSP}(f)$ reduces to $\mathsf{CSP}(f)$; otherwise, $\mathsf{QCSP}(f)$ is coNP-hard.*

Similar techniques can be used to establish a classification result on idempotent *set functions*. We consider a *set function* (on domain D) to be a mapping from $\wp(D) \setminus \{\emptyset\}$ to D, where $\wp(D)$ denotes the power set of D. A set function $f : \wp(D) \setminus \{\emptyset\} \to D$ is considered to be *idempotent* if for all $d \in D$, it holds that $f(\{d\}) = d$. We consider a relation R of arity k to be *invariant* under a set function $f : \wp(D) \setminus \{\emptyset\} \to D$ if for any non-empty subset $S \subseteq R$, it holds that the tuple $(f(\{t_1 : \bar{t} \in S\}), \ldots, f(\{t_k : \bar{t} \in S\}))$ is in R. Equivalently, R is invariant under $f : \wp(D) \setminus \{\emptyset\} \to D$ if it is invariant under all of the functions $f_i : D^i \to D$ defined by $f_i(d_1, \ldots, d_i) = f(\{d_1, \ldots, d_i\})$, for $i \geq 1$. Set functions were studied in the context of CSP complexity by Dalmau and Pearson [18];

among other results, they showed that any problem of the form $\mathsf{CSP}(f)$, for f a set function, is polynomial-time decidable.

We can define notions similar to those in the beginning of this section for set functions. In particular, when f is a set function, we say that an element $a \in D$ can *f-hit* an element $b \in D$ in one step if there exists a subset $A \subseteq D$ such that $f(\{a\} \cup A) = b$. We define a set $C \subseteq D$ to be *coherent* with respect to f if it is nonempty and for all nonempty $A \subseteq D$, the following holds: if $A \setminus C$ is nonempty, then $f(A) \notin C$. Using these notions, it is possible to give proofs analogous to those for the previous results of this section, yielding the following classification theorem.

Theorem 18. *Let $f : \wp(D) \setminus \{\emptyset\} \to D$ be an idempotent set function. If there is a unique minimal coherent set with respect to f, then $\mathsf{QCSP}(f)$ reduces to $\mathsf{CSP}(f)$ (and is hence in P by [18]); otherwise, $\mathsf{QCSP}(f)$ is coNP-hard.*

5 Commutative-Projective Operations

As we have indicated (proof of Theorem 15), all binary operations giving rise to maximal constraint languages can be seen as having one of two types. The previous section was concerned with a generalization of the first type, *commutative* binary operations; this section studies the second type, *commutative-projective* operations.

Definition 19. *A* commutative-projective *operation is a binary idempotent operation $f : D^2 \to D$ such that for every two-element subset $\{a, b\}$ of D, either $f(a, b) = f(b, a)$, or f acts as a projection on $\{a, b\}$; and, there exists a two-element subset $\{a, b\}$ of D such that f acts as a projection on $\{a, b\}$.*

Fix a commutative-projective operation $f : D^2 \to D$. Based on f, we define two directed graphs G_1, G_2 as follows. Both of these graphs have vertex set D. Let there be an edge from a to b in both G_1 and G_2 if there exists $d \in D$ such that $f(a, d) = f(d, a) = b$. In addition, let there be an edge from a to b as well as an edge from b to a in G_i if on the two-element set $\{a, b\}$ the operation f acts as the projection onto the ith coordinate. We have the following results.

Lemma 20. *Suppose that there exists $d_0 \in D$ such that for both $i \in \{1, 2\}$ and for every $d \in D$, there is a path from d_0 to d in G_i. Then, $\mathsf{QCSP}(f)$ is NP-complete.*

For each $i \in \{1, 2\}$, define $\leq_i$ to be the partial ordering on strongly connected components of G_i where $C \leq_i C'$ if there exist vertices $v \in C, v' \in C'$ such that there is a path from v to v' in G_i. We define a component C to be minimal in G_i if for all components C' such that $C' \leq_i C$, it holds that $C' = C$.

Lemma 21. *Suppose that one (or both) of the graphs G_1, G_2 has more than one minimal component. Then, $\mathsf{QCSP}(f)$ is PSPACE-complete.*

Theorem 22. *Let $f : D^2 \to D$ be a commutative-projective operation such that $\mathsf{Inv}(f)$ is a maximal constraint language. The problem $\mathsf{QCSP}(f)$ is either NP-complete or PSPACE-complete.*

Acknowledgements. The author thanks the anonymous referees for their helpful comments.

References

1. Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.
2. E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with boolean blocks, part II: constraint satisfaction problems. *ACM SIGACT-Newsletter*, 35(1):22–35, 2004.
3. F. Börner, A. Bulatov, A. Krokhin, and P. Jeavons. Quantified constraints: Algorithms and complexity. In *Computer Science Logic 2003*, 2003.
4. Andrei Bulatov. Combinatorial problems raised from 2-semilattices. Manuscript.
5. Andrei Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proceedings of 43rd IEEE Symposium on Foundations of Computer Science*, pages 649–658, 2002.
6. Andrei Bulatov. Malt'sev constraints are tractable. Technical Report PRG-RR-02-05, Oxford University, 2002.
7. Andrei Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of 18th IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 321–330, 2003.
8. Andrei Bulatov. A graph of a relational structure and constraint satisfaction problems. In *Proceedings of 19th IEEE Annual Symposium on Logic in Computer Science (LICS'04)*, 2004.
9. Andrei Bulatov and Peter Jeavons. Tractable constraints closed under a binary operation. Technical Report PRG-TR-12-00, Oxford University, 2000.
10. Andrei Bulatov, Andrei Krokhin, and Peter Jeavons. Constraint satisfaction problems and finite algebras. In *Proceedings 27th International Colloquium on Automata, Languages, and Programming – ICALP'00*, volume 1853 of *Lecture Notes In Computer Science*, pages 272–282, 2000.
11. Andrei Bulatov, Andrei Krokhin, and Peter Jeavons. The complexity of maximal constraint languages. In *ACM Symposium on Theory of Computing*, pages 667–674, 2001.
12. Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117(1):12–18, 1995.
13. Hubie Chen. Collapsibility and consistency in quantified constraint satisfaction. In *AAAI*, 2004.
14. Hubie Chen. Quantified constraint satisfaction and 2-semilattice polymorphisms. In *CP*, 2004.
15. Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classification of Boolean Constraint Satisfaction Problems.* SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2001.
16. Victor Dalmau. Some dichotomy theorems on constant-free quantified boolean formulas. Technical Report LSI-97-43-R, Llenguatges i Sistemes Informàtics - Universitat Politècnica de Catalunya, 1997.
17. Victor Dalmau. A new tractable class of constraint satisfaction problems. In *6th International Symposium on Artificial Intelligence and Mathematics*, 2000.

18. Victor Dalmau and Justin Pearson. Closure functions and width 1 problems. In *CP 1999*, pages 159–173, 1999.
19. Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.
20. Peter Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.
21. Peter Jeavons, David Cohen, and Martin Cooper. Constraints, consistency, and closure. *Articial Intelligence*, 101(1-2):251–265, 1998.
22. Peter Jeavons, David Cohen, and Justin Pearson. Constraints and universal algebra. *Annals of Mathematics and Artificial Intelligence*, 24(1-4):51–67, 1998.
23. P.G. Jeavons, D.A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.
24. D. Kavvadias and M. Sideri. The inverse satisfiability problem. *SIAM Journal on Computing*, 28(1):152–163, 1998.
25. Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2001.
26. L. M. Kirousis and P. G. Kolaitis. The complexity of minimal satisfiability problems. In *Proceedings 18th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2010 of *Lecture Notes in Computer Science*, pages 407–418. Springer-Verlag, 2001.
27. Ph.G. Kolaitis and M.Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000.
28. Emil L. Post. *The Two-Valued Iterative Systems of Mathematical Logic*. Princeton University Press, 1941.
29. I.G. Rosenberg. Minimal clones I: the five types. In *Lectures in Universal Algebra (Proc. Conf. Szeged 1983)*, volume 43 of *Colloq. Math. Soc. Janos Bolyai*, pages 405–427. North-Holland, 1986.
30. Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

Regular Tree Languages Definable in FO

Michael Benedikt[1] and Luc Segoufin[2]

[1] Bell Laboratories, 2701 Lucent Lane, Lisle, IL 60532, USA
benedikt@research.bell-labs.com
[2] INRIA-Futurs, 4 re Jacques Monod, Orsay Cedex 91893, France
http://www-rocq.inria.fr/~segoufin

Abstract. We consider regular languages of ranked labeled trees. We give an algebraic characterization of the regular languages over such trees that are definable in first-order logic in the language of labeled graphs. These languages are the analog on ranked trees of the "locally threshold testable" languages on strings. We show that this characterization yields a decision procedure for determining whether a regular collection of trees is first-order definable: the procedure is polynomial time in the minimal automaton presenting the regular language.

Keywords: Tree automata, Logic.

1 Introduction

This paper is concerned with the relationship between regularity (acceptance by an automaton) and definability in first-order logic. Over strings this relationship is well-understood. A fundamental result in formal language theory [Bc60] states that a language of strings is regular – that is, equal to the language of strings accepted by a finite string automaton – exactly when it is definable in monadic-second-order logic (MSO) over the vocabulary consisting of the successor relation on strings and the labels. By restricting to first-order logic (FO) rather than MSO, we can obtain two proper subcollections of the family of regular languages. The languages that are definable in first-order logic over the transitive closure of the successor relation and the label predicates, which we denote FO($<$), are exactly the star-free or, equivalently, the aperiodic languages [MP71, Sch65]. The languages that are definable in first-order logic over the successor relation and the label predicates, which we denote by FO, correspond to locally threshold testable languages (see [Tho97]). Using a fundamental result of Thérien and Weiss [TW85], Beauquier and Pin [BP89] gave an algebraic characterization of the FO languages. They are exactly the languages for which the corresponding monoid satisfies one of a family of sets of identities. Put another way, they show that the monoids corresponding to FO-definable languages form a pseudo-variety within the collection of all finite monoids. Both the characterization of FO($<$)-definability via aperiodicity and the characterization of FO-definability of Beauquier and Pin lead to effective algorithms for checking whether a regular

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 327–339, 2005.

language is FO(<) (resp. FO) definable. A complete overview of the string case can be found in [Tho97] or in [Str94].

We now consider the situation over ranked trees – labeled trees with a fixed bound on branching. Regularity is now defined as acceptance by a (nondeterministic top-down or deterministic bottom-up) tree automaton, and regularity is shown to be equivalent to definability in monadic-second-order logic in the vocabulary of labeled graphs [Don70, TW68]. In this context we use FO(<) to denote first-order logic over the labels and the transitive closure of the graph relation and FO to denote first-order logic over the graph relation and the labels. The notions of aperiodicity and star-freeness have natural extensions to the tree context, but here FO(<) is strictly weaker than aperiodicity and star-freeness [PT93, Heu91, Pot95]. Finding a decidable characterization of FO(<) within the regular tree languages is a longstanding open problem; partial results (see below) are given in [EW03, BW04]. As in the string case, FO definability is known to be strictly weaker than FO(<) definability, but surprisingly an effective characterization of FO-definability was also lacking. [Wil96] gives an algebraic characterization and decision procedure for the frontier testable languages, a subclass of the FO definable languages. [BW04] provides a decision procedure for two fragments of FO(<) defined using existential path quantification; none of these fragments exactly matches the expressiveness of FO. [EW03] gives a characterization of the FO(<) definable languages in terms of an algebraic structure (the "syntactic pre-clone") associated with the language; this characterization is not known to be effective.

In this work we settle the issue of effective characterization of FO-definability. Over trees FO still corresponds to the Local Threshold Testable (LTT) languages, but this characterization does not yield a decision procedure. Our main result is an effective characterization of FO within the regular tree languages that uses equations over an algebra associated with the language. These equations characterize the algebra of pointed trees associated with the formula, rather than the monoid that is used to characterize definability in the string case; this algebraic approach is similar to that proposed in [Wil96]. As strings are FO definable within trees, our proof yields a new proof of the string characterization of [BP89]. The current proofs of the characterization theorem in the string case use either fundamental (and difficult) results in the theory of monoids [BP89] or difficult results within the theory of finite categories [Str94]. Our proof of the characterization theorem for trees uses an alternative approach, exploiting locality results [Lib04] and an inductive rewriting argument. We believe that, when restricted to the special case of strings, our proof is much simpler than the previous above mentioned ones. Nevertheless several of the technical lemmas remain identical in inspiration if not in notation to the earlier proofs.

Organization: Section 2 gives the formal framework for the main result, and reviews relevant facts about trees and first-order definability. Section 3 presents and illustrates the equations that define our algebraic characterization, and states the main characterization theorem of the paper, while Section 4 is de-

voted to the proof. Section 5 gives the decision procedure that follows from the characterization theorem, and analyzes its complexity, while Section 6 gives conclusions and open issues. Proofs omitted due to space constraints will appear in the journal version.

2 Notations

We fix a finite alphabet Σ and a rank $k \geq 2$. We denote by *tree* a rooted acyclic finite graph with labels in Σ such that each node has at most k children. The number of children of a node n is denoted $deg(n)$. Trees by default will also be *ordered*, which means that each non-root node n has, in addition to its label, an associated integer $i \leq k$, denoted $ind(n)$, where every node n has one child c with $ind(c) = i$ for each $i \leq deg(n)$. For trees we can thus speak of the first/second ... child of a node. We will also deal with *unordered trees*, where there is no such additional integer to distinguish children with the same label. The set of trees is denoted by T and the set of unordered trees by UT. In this paper we will prove results for both ordered and unordered trees, but we will concentrate the exposition on the ordered case; the unordered case requires only slight modifications. Thus we will often write "tree" for "ordered tree" below.

A *pointed tree* is a tree with a designated (unlabeled) leaf which acts as a hole. The concatenation of two pointed trees Δ and Δ' is denoted by $\Delta \cdot \Delta'$ and is the pointed tree constructed from Δ by plugging Δ' to its hole. The set of pointed trees is denoted by T^1. A *j-pointed tree* is a tree with j designated leaves, while an *extended pointed tree* is a j-pointed tree for some j. We assume that the j holes are indexed by $1 \dots j$, according to their lexicographical order in the order tree case, arbitrarily otherwise. For any extended pointed tree Δ and any (pointed or not) trees $t_1 \cdots t_j$, $\Delta[t_1 \cdots t_j]$ denotes the (extended pointed) tree constructed from Δ by plugging t_1 in its first hole, t_2 in its second hole, etc. We also denote by $\Delta \cdot_i t$ the operation which consists of plugging t in the i^{th} hole of Δ. The set of j-pointed trees is denoted by T^j. We can similarly talk about pointed and j-pointed unordered trees, defining concatenation and substitution in the same way on them.

A (deterministic bottom-up) tree automaton is defined in the usual way, consisting of a set of states and a transition function associating a unique state to any pair (i-tuple of states for $i \leq k$,label). The semantics of tree automata, as well as basic properties, can be found in [Tho97]. A tree automaton defines a collection of trees, and a tree automaton whose transition function is invariant under permuting the ordering of the states can be considered to define a collection of unordered trees.

When a tree automaton A is fixed, each pointed tree can be viewed as a function from states to states associating the state reached by A at its root when a state is assumed at its hole. We then write $\Delta \simeq \Delta'$ to denote the fact that the functions associated to the pointed trees Δ and Δ' are equal. When A is fixed a pointed tree is said to be *idempotent* if the function it defines is idempotent ($f^2 = f$).

MSO and FO are defined over (ordered) trees in the standard way over the signature containing one unary predicate P_a per letter $a \in \Sigma$, the binary tree successor relation G, and predicates C_i that hold of node n iff n is the i^{th} child of its parent. Notice that $<$, the ancestor relation, is *not* included in the signature, therefore FO does not have access to it (of course it is definable from G in MSO). When the trees are unordered the predicates C_i are omitted in the signature. It is known that the languages defined by MSO formulas are exactly the regular tree languages, in both the ordered and unordered case. We wish to give a characterization of languages definable in FO among the regular tree languages. As mentioned in the introduction, a language L is FO exactly when it is Locally Threshold Testable (LTT) – this means that there are integers i and d such that membership of a tree T in L depends only on the number of occurrences of each i-depth tree as a subtree of T, where the number is calculated only up to d. This does not give an algorithm, since there are infinitely many i and d to check.

For any formula $\varphi \in$ FO, its quantifier rank $\mathrm{qr}(\varphi)$ is defined as the nesting depth of the quantifiers of φ as usual. The elementary equivalence up to depth n is denoted by $\equiv^n$: for any two trees $t, t' \in T$ we say that $t \equiv^n t'$ if t and t' satisfy exactly the same FO sentences of quantifier rank less than n.

3 Main Result

Let L be a regular tree language and A be the minimal bottom-up automata for L. We say that L satisfies (†) if the following holds:

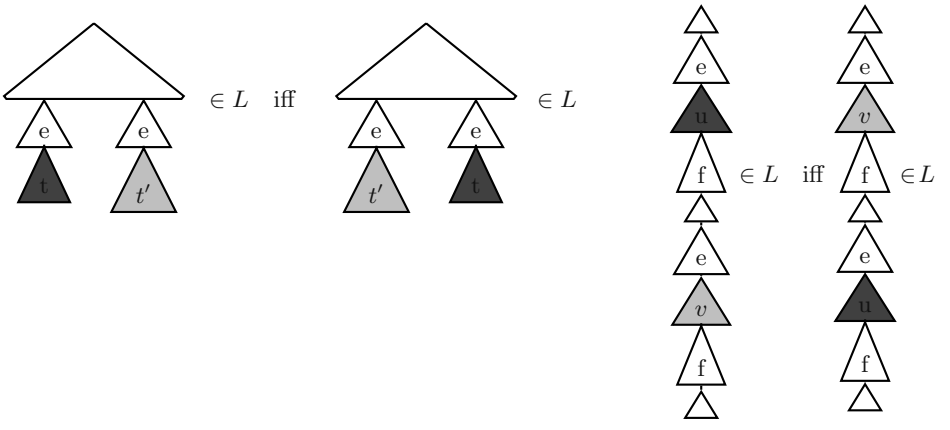
1. **Horizontal swap** (see the left part of Figure 1). For any $\Delta \in T^2$, $e \in T^1$ idempotent and any $t, t' \in T$
$$\Delta[e \cdot t, e \cdot t'] \in L \quad \text{iff} \quad \Delta[e \cdot t', e \cdot t] \in L$$
2. **Vertical swap** (see the right part of Figure 1). For any $s, s', u, v \in T^1$, $e, f \in T^1$ idempotents, and $t \in T$
$$s \cdot e \cdot u \cdot f \cdot s' \cdot e \cdot v \cdot f \cdot t \in L \quad \text{iff} \quad s \cdot e \cdot v \cdot f \cdot s' \cdot e \cdot u \cdot f \cdot t \in L$$
3. **Aperiodicity**. There exists a l such that for any $s, u \in T^1$ and any $t \in T$
$$s \cdot u^l \cdot t \in L \quad \text{iff} \quad s \cdot u^{l+1} \cdot t \in L$$

We want to show the following:

Theorem 1. *Let L be a regular tree language. L verifies (†) iff L is definable in FO. The same holds for L a regular language of unordered trees.*

Part 2 and 3 of (†) are the straightforward extensions to trees of the characterization of FO definable *string* languages among regular *string* languages [BP89, Str94]. They are no longer sufficient when dealing with trees.

Proof. of Theorem 1. ($\Leftarrow$) This is a classical locality argument. If e is idempotent, e can be replaced by e^m for any m without affecting membership in L. Take m bigger than the locality rank (see [Lib04] for a definition) of an FO formula defining L. All right-hand-side and left-hand-side strings considered in part 1 and



Condition 1 of (†) Condition 2 of (†)

Fig. 1. Swap conditions of (†)

2 of (†) now have the same local neighborhood and thus the same membership in L. We obtain the same result for part 3 by choosing for l the same m.

The opposite direction follows from the following theorem, which will be quite involved:

Theorem 2. *For any regular language L satisfying* (†)*, there exists a K such that for any $t, t' \in T$ we have:* $t \equiv^K t' \quad \Rightarrow \quad t \in L \quad \textit{iff} \quad t' \in L$

Before proving Theorem 2 we show how Theorem 1 follows from it. From Theorem 2 we know that if L satisfies (†), L is a union of equivalence classes of $\equiv^K$ for some K. Standard arguments of finite model theory ([Lib04]) show that $\equiv^K$ has only finitely many equivalence classes and that each of them is definable in FO. Therefore L is definable in FO as a disjunction of such formulas for the corresponding classes.

4 Proof of Theorem 2

We will prove this in the case of ordered ranked trees, but a simple modification proves the result in the unordered case. For the rest of the section, we fix a regular language L and let A_L be the minimal deterministic bottom-up tree automaton recognizing L. We fix l for (†)3, and we refer to the following transformations corresponding to (†):

1. For any $\Delta \in T^2$, $e \in T^1$ idempotent and any $t, t' \in T$
$$\Delta[e \cdot t, e \cdot t'] \rightsquigarrow \Delta[e \cdot t', e \cdot t]$$
2. For any $s, s', u, v \in T^1$, $e, f \in T^1$ idempotents, and $t \in T$
$$s \cdot e \cdot u \cdot f \cdot s' \cdot e \cdot v \cdot f \cdot t \quad \rightsquigarrow \quad s \cdot e \cdot v \cdot f \cdot s' \cdot e \cdot u \cdot f \cdot t$$
3. for any $s, u \in T^1$ and any $t \in T$
$$s \cdot u^l \cdot t \quad \leftrightsquigarrow \quad s \cdot u^{l+1} \cdot t$$

By (†), these transformations preserve membership in L. We denote by $t \approx t'$ the fact that a tree t can be transformed into the tree t' using operations of (†).

The proof is divided into two parts. In the first part, we give a *canonical decomposition* of a tree t, into *skeletal normal form* $\bar{t}$, which consists of a fixed set of *blocks* connected by idempotents. We will then use this decomposition together with the finite rank of trees to show that if two trees t and t' are such that $t \equiv^K t'$ for some sufficiently big K then $\bar{t}$ and $\bar{t}'$ have the same blocks with the same number of occurrences up to some threshold which increases with K. Note that this part works for any regular (ranked) tree language L.

In the second part we will show that if the threshold (and therefore K) is big enough then $\bar{t}'$ can be transformed into $\bar{t}$ using only the transformation rules above. Therefore if L satisfies (†), $\bar{t} \in L$ iff $\bar{t}' \in L$. By construction of $\bar{t}$ and $\bar{t}'$ the latter implies $t \in L$ iff $t \in L'$.

We now move to part one. Let x be the number of states of A_L. Fix m_L to be an integer bigger than x^x. The transformation is based on the following key lemma (generalizing Lemma B.1.2 of [Str94]), which says that we can insert idempotents throughout a tree.

Lemma 1. *For any tree $t \in T$ and any node n in t such that the length of the path from the root of t to n is equal to m_L there exists an idempotent e, a pointed tree $t' \in T^1$ and a tree $t'' \in T$ such that $t = t' \cdot t''$, n is in t'', and $t' \simeq t' \cdot e$. In particular, $t \simeq t' \cdot e \cdot t''$*

In particular Lemma 1 shows that, in any tree, if we take a path long enough, we can break it and insert an idempotent, without affecting membership in L . In order to obtain a canonical insertion we fix, once and for all, for each idempotent (viewed as a function) a representative in T^1 and an order among idempotents. When applying Lemma 1 we will always choose t' minimal and e minimal. In other words we will always insert an idempotent as soon as possible and insert the *smallest* possible one whenever there is a choice.

The construction of $\bar{t}$ from t is done as follows. For any subtree t' of t and any leaf n of t' which is at distance bigger than m_L from the root, apply Lemma 1 with t' and n and choose the highest position $p_{t',n}$ (positions are viewed as marked nodes in t) in the path from the root of t' to n and the minimal idempotent $e_{t',n}$ that can be inserted at that node. We now define $\bar{t}$ as t with all $e_{t',n}$ inserted at the corresponding positions $p_{t',n}$. Notice that different n may lead to the same $p_{t',n}$ but in this case the corresponding idempotents are equal. We next claim that if e is inserted at node n in tree t, and t' is any other tree containing n, then we cannot insert an idempotent $f \neq e$ in n for t'. This is because the insertion of e for t depended only on some path p leading from the root of t to n. For some other t', if there is some path p' to n in t', then p' must either be contained in p or contain p. In the former case, by the choice of the highest place, we must have inserted f for p, while in the latter case we must have inserted e for p'.

By induction on the idempotent inserted and using the crucial fact that $t' \simeq t' \cdot e$ for all idempotents inserted (and hence the same holds for all supertrees of t'), it is straightforward to prove the following lemma.

Lemma 2. *$t \in L$ iff $\bar{t} \in L$*

Note also that the construction above guarantees that an idempotent is inserted at least every m_L nodes in a given path. Therefore $\bar{t}$ can be viewed as a tree divided into *blocks* where a block is a finite tree of depth $\leq m_L$ with an idempotent attached to its root and idempotents attached to some of its leaves. We call *ports* the holes of the later idempotents. Thus in $\bar{t}$, blocks are connected via their ports and the corresponding idempotents match: a block b_1 is connected to the i^{th} port of block b_2 if the idempotent of the root of b_1 is equal to the idempotent of the i^{th} port of b_2. As the trees are ranked and we have fixed a representative for each idempotent the number of possible distinct blocks is finite.

We illustrate the construction of $\bar{t}$ from t with the following example:

Example 1. Let L be the regular language consisting of all trees containing arbitrarily many occurrences of the patterns  and  separated by arbitrarily long paths of nodes labeled with e. An obvious idempotent of L is  .

Figure 2 shows a tree t and its corresponding block decomposition $\bar{t}$. Note that b_2 and b_4 are two different blocks even though their underlying subtrees are the same () because b_2 and b_4 differ in their ports.

Finally note that the construction is local-canonical. By this we mean that the neighborhoods of depth m_L of t determine the blocks of $\bar{t}$. Indeed we can prove the following key lemma:

Lemma 3. *For each integer d, let K_d be $d + k^{m_L+1}$ then $t \equiv^{K_d} t'$ implies $\bar{t}$ and $\bar{t}'$ have exactly the same blocks with the same number of occurrences, up to threshold d.*

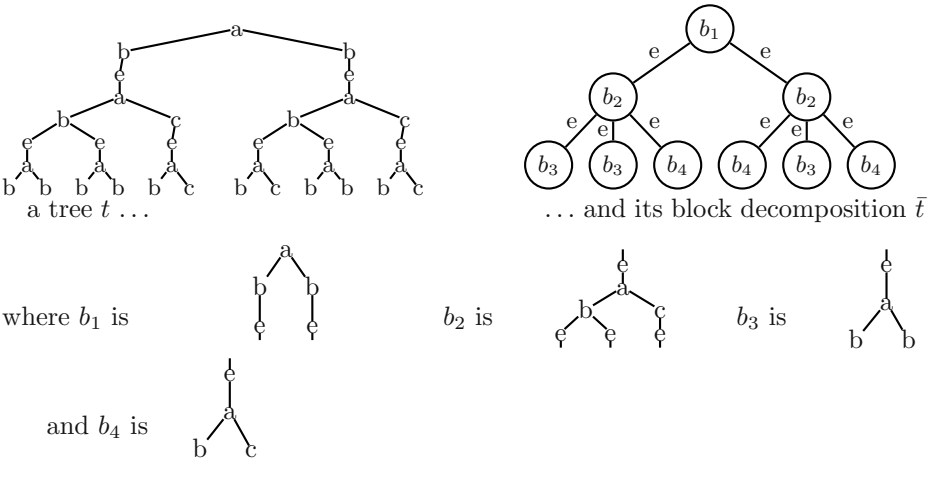


Fig. 2. Illustration of the block decomposition

This ends the first part. By Lemma 2 we now have to prove that $t \equiv^K t'$ for some K implies $\bar{t} \in L$ iff $\bar{t}' \in L$.

Let BT (*block-trees*) be the set of trees which are formed with blocks. More formally a block tree is a tree with nodes labeled by blocks (b_n is the block associated to node n) and edges labeled by numbers (we assume a canonical ordering for the ports of each block) so that the whole structure is consistent with the block definition and the idempotent attached to each port: if node m is the i^{th} child of a node n then the idempotent of the root of b_m matches the idempotent of the i^{th} port of b_n. The pointed block trees PBT are pointed trees formed of blocks.

For the second part we make use of the transformations obtained from (†). These operations, and the equivalence relation $t \approx t'$ can also be viewed as having domain block trees (since a block tree is a tree with additional structure). In what follows, we will always be re-arranging the blocks given by the original decomposition that constructs $\bar{t}$ and $\bar{t}'$.

If t is in PBT and b is a block, $|t|_b$ denotes the number of occurrences of b in t. As we will count the number of blocks up to some threshold we write $t =^b_d t'$ if for each block b we have $|t|_b = |t'|_b$ or both $|t|_b$ and $|t'|_b$ are bigger than d. We use $t =^b_\infty t'$ if for each block b we have $|t|_b = |t'|_b$.

Our plan for the remainder of the proof is as follows. Assuming that $t \equiv^K t'$ for large enough K, we want to show that $t \in L$ iff $t' \in L$. By Lemma 3 and Lemma 2 we have to show that if $\bar{t} =^b_d \bar{t}'$ for some large enough d then $\bar{t} \in L$ iff $\bar{t}' \in L$. In order to show this we will first transform $\bar{t}'$ into $\bar{t}_1$ – without affecting membership in L – so that for each block b, $|\bar{t}_1|_b \geq |\bar{t}|_b$ and $\bar{t} =^b_{d'} \bar{t}_1$ for some $d' < d$ but still big enough. In a second step we will construct $\bar{t}_2$ from $\bar{t}_1$ without affecting membership in L so that $\bar{t}_2$ is $\bar{t}$ with extra sections inserted between successive nodes. We will say that $\bar{t}$ is *pseudo-included* in $\bar{t}_2$. We can do that without changing the number of occurrences of each block: $\bar{t}_2 =^b_\infty \bar{t}_1$. In a third step we will show how to remove those extra sections without affecting membership in L.

The following lemma takes care of the first step. Its proof, is done using a careful succession of swapping moves followed by a pumping argument.

Lemma 4. *For each integer d' there exists an integer d such that if t and t' are block trees verifying $t =^b_d t'$ then there exists t'' such that, for each block b, $|t''|_b \geq |t|_b$, $t =^b_{d'} t''$, and $t'' \in L$ iff $t' \in L$.*

A block tree t is *pseudo-included* in a block tree t' if there is an injective mapping h from nodes of t to nodes of t' such that: (i) $b_n = b_{h(n)}$ and (ii) if n is the i^{th} child of m in t then $h(n)$ is a descendant of i^{th} child of $h(m)$ in t'. In this case the *h-pseudo-tree* is the minimum subtree of t' which contains $h(t)$. The second step of the proof, showing pseudo-inclusion, is contained in the following lemma which is proved by induction on t using only the swapping moves of (†).

Lemma 5. *Let $t \in BT$ and $t' \in BT$. If for all blocks b occurring in t we have $|t|_b \leq |t'|_b$ then there exists $t'' \approx t'$ such that t is pseudo-included in t'' and $t'' =^b_\infty t'$.*

In order to complete the last step of Theorem 2 we need to be able to remove extra sections. A *loop* is a pointed block tree which has the same idempotent in its root and in its hole. To prove the next step we will need the following lemma which shows that it is possible to remove *loops* containing only blocks that occur many times in the rest of the tree. The proof works by re-arranging the loop into the form u^l, then applying (†)3;it is inspired by Lemma B.3.7 of [Str94].

Lemma 6. *Let $t = t_1 \cdot u \cdot t_2 \in BT$ such that u is a loop. If for each block b occurring in u we have $|t_1 \cdot t_2|_b \geq (|u|_b * l) + 1$ then $t \approx t_1 \cdot t_2$.*

Proof of Theorem 2: Let x be the number of states of A and e_L be the number of transition functions of L. Let $d' = k^{e_L * x} * l + 1$. Let d be the number required in Lemma 4 for d'. Let K be the number K_d required in Lemma 3 for d. We show that $t \equiv^K t'$ implies $t \in L$ iff $t' \in L$. Assume $t \equiv^K t'$. From Lemma 3 we know that $\bar{t}$ and $\bar{t'}$ have the same number of occurrences of each block up to d. Moreover from Lemma 2 $t \in L$ iff $\bar{t} \in L$. Therefore it suffices to show that $\bar{t} \in L$ iff $\bar{t'} \in L$. By the choice of K we can apply Lemma 4 and construct t'' such that $t'' \in L$ iff $\bar{t'} \in L$ and $t'' =^b_{d'} t$. We can now apply Lemma 5 and obtain t''' such that $t''' \in L$ iff $t'' \in L$, $\bar{t}$ is pseudo-included in t''', and $t''' =^b_\infty t'' =^b_{d'} t$. Therefore it suffices to prove that $\bar{t} \in L$ iff $t''' \in L$.

We construct by induction $t_1 \cdots t_n$ such that: (i) t_1 is t''', (ii) for all i $t_i \in L$ iff $t_{i+1} \in L$, (iii) for all i $t_i = v_i[u_1^i \cdots u_{k_i}^i]$ where v_i is a prefix of $\bar{t}$, (iv) for all i v_i is a strict prefix of v_{i+1}, (v) for all i $\bar{t}$ is pseudo-included in t_i by a mapping sending v_i to its copy in t_i, and, (vi) $t_n = \bar{t}$. From (i), (ii) and (vi), the desired equivalence of $\bar{t} \in L$ and $t''' \in L$ follows.

Assuming we have constructed t_i, we show how to construct t_{i+1} as long as v_i is a strict prefix of $\bar{t}$. This suffices, since if $v_i = \bar{t}$ then (v) implies that we have achieved (vi). By the induction hypothesis we know that $\bar{t} = v_i[w_1 \cdots w_{k_i}]$, $t_i = v_i[u_1 \cdots u_{k_i}]$, and that each w_j is pseudo-included in u_j. Let $(h_j)_{1 \leq j \leq k_i}$ be the corresponding pseudo-inclusion mappings and h_i be the mapping pseudo-including $\bar{t}$ in t_i. Let $(n_j)_{1 \leq j \leq k_i}$ be the roots of w_j. If for some j one of the h_j maps n_j to the root of u_j then we are done as we can immediately extend all the required properties to $v_{i+1} = v_i \cup \{n_j\}$. Otherwise each u_j is of the form $u'_j \cdot n'_j[\Delta_j]$ where u'_j is a nonempty block tree, $n'_j = h_j(n_j)$ and Δ_j is a forest of block trees. Note that u'_j is a loop.

The crucial fact is that all blocks which occur in t_i outside of $h_i(\bar{t})$ have strictly more occurrences in t_i than in $\bar{t}$. Therefore they occur more than d' times in $\bar{t}$. In particular this is the case for all blocks occurring in u'_j.

From t_i we construct t'_i which decreases the size of u'_j without affecting the rest. To do this we repeatedly apply the following operation which does not affect membership in L and does not affect the block structure. Let u be a block tree. Label each node with the pair (q, e) where e is the idempotent of the root of the block associated to the node and q is the state reached by the automaton A_L recognizing L at the root of that block. Whenever there is a branch p in u which contains the same label twice, we prune the section between the top idempotent and the bottom one (using the standard pumping lemma). This yields a u''_j whose depth is bounded by $e_L * x$. When we have done this we have $t'_i \in L$ iff $t_i \in L$ and

for each block b occurring in u''_j we have $|u''_j|_b * l + 1 \leq d' \leq |\bar{t}|_b \leq |t_i - u''_j|_b$. The first inequality is by the choice of d' and the size of u''_j. The second one is from the remark in the previous paragraph and the last one is by pseudo-inclusion of $\bar{t}$ into t_i.

We can therefore apply Lemma 6 with $t_1 = v_i$ and $u = u''_j$ to remove u''_j obtaining the desired t_{i+1}. This concludes the proof of the theorem.

5 Decidability

By the minimal deterministic automaton for a language, we mean an automaton A in which (i) for each state $q \in A$ there is some tree such that the automaton run on the tree reaches state q at the root and (ii) if t and t' are two pointed trees such that whenever we have that $u \cdot t \cdot v \in L$ iff $u \cdot t' \cdot v \in L$ for all u, v, then $t \simeq t'$ (that is, they define the same function on states of A). We seek algorithms for deciding membership in FO that are polynomial in the size of such an A.

In the string case deciding whether a regular language L can be defined in FO is PTime in the size of the minimal deterministic automaton recognizing L [Pin96]. That is, it is possible to check in PTime that (†)2 and (†)3 holds. Note that this is not immediate, as checking (†)3 alone is PSpace-complete [CH91]. It turns out that ideas similar to [Pin96] show that (†) can actually be checked in PTime.

We first sketch how to check (†)2 and (†)3 together and then we show how to check for (†)1. These are easy extension to trees of the proof of [Pin96]. The first idea is to replace (†)3 by the condition (†)3′: there exists l such that for any $s, x, y \in T^1$, $e \in T^1$ idempotents, and $s' \in T$ $s \cdot (e \cdot x \cdot e \cdot y \cdot e)^l \cdot s' \in L$ iff $s \cdot (e \cdot x \cdot e \cdot y \cdot e)^l \cdot e \cdot x \cdot e \cdot s' \in L$.

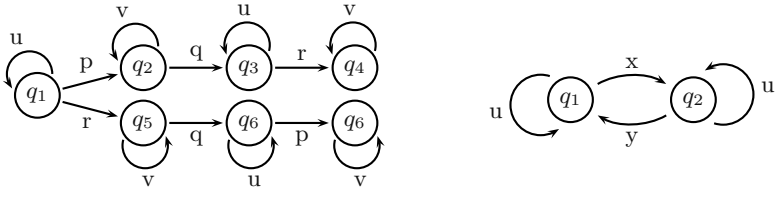
In [Pin96] it is shown that (†)2 and (†)3 are equivalent to (†)2 and (†)3′. In fact, [Pin96] shows that in any monoid satisfying $e \cdot u \cdot f \cdot s' \cdot e \cdot v \cdot f = e \cdot v \cdot f \cdot s' \cdot e \cdot u \cdot f$, we have the following are equivalent: a) there is l such that $u^l = u^{l+1}$ holds and b) there is l such that $(e \cdot x \cdot e \cdot y \cdot e)^l = (e \cdot x \cdot e \cdot y \cdot e)^l \cdot e \cdot x \cdot e$ holds (where e, f range over idempotents and x, y, u, v, s' range over monoid elements). Since the set of $\simeq$ classes of pointed trees associated with a regular tree language forms a monoid, this result can be applied to pointed trees. So we have:

Lemma 7. *[Pin96] For any regular tree language L, L verifies (†)2 and (†)3 iff L verifies (†)2 and (†)3′.*

We now fix A to be a minimal bottom-up tree automaton recognizing L. The set of states of A is denoted by Q and δ is the transition function of A. It turns out that each of (†)1, (†)2, and (†)3′ can be checked in PTime by means of forbidden patterns in a graph constructed from A.

For the purposes of this section, a *pattern* is a graph whose edges are labeled by variables which range over elements of Γ^* for some finite alphabet Γ. In addition a pattern comes with side conditions stating which nodes of the pattern should be interpreted as distinct nodes. Let G be a graph whose edges are labeled in Γ. Such a graph G *matches* a pattern if there is a mapping m taking each

variable in the pattern to strings in Γ^* and each node of the pattern to a node of G such that for each side constraint $p_1 \neq p_2$, $m(p_1) \neq m(p_2)$, and such that whenever there is an edge from p_1 to p_2 in the pattern labeled with v, there is a path from $m(p_1)$ to $m(p_2)$ in G whose labels yield the string $m(v)$. In [CPP93] it is noted that for every fixed pattern, the problem of determining given a graph, whether the graph matches the pattern, is in PTIME. This result was used to show that FO-definability is in PTIME in the string case. From a minimal automaton A recognizing L one constructs an edge-labeled graph $G_A = (V_A, E_A)$ as follows. The vertex set V_A of G_A is the set of states of A. The transitions $E_A \subseteq V_A \times \Sigma \times V_A$ are labeled with letters of the alphabet Σ of L and correspond to the transitions of A. Let P_2 be the pattern depicted in the left below together with the condition $q_4 \neq q_6$.



Let $P_{3'}$ be the pattern depicted in the right above together with the condition $q_1 \neq q_2$.

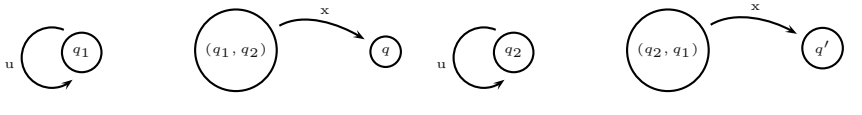
It has been shown that (i) [Pin96] L verifies (†)2 iff G_A does not match P_2 and (ii) [CPP93] L verifies (†)3′ iff G_A does not match $P_{3'}$. Minimality of A is used in the left to right direction.

This result extends to trees as follows (we show only the case of binary ordered trees, the extension to arbitrary arity and to unordered trees is immediate). From an automaton A define $G^1_A = (V^1_A, E^1_A)$ as follows. The set of vertices V^1_A is Q. The set of edges $E^1_A \subseteq V^1_A \times \Gamma \times V^1_A$ where $\Gamma = \Sigma \times Q \times \{0,1\}$ connects a node p to a node p' via an edge $(a,q,0)$ if $\delta(\langle p,q\rangle, a) = q'$ and via an edge $(a,q,1)$ if $\delta(\langle q,p\rangle, a) = q'$. That is, an edge represents the inverse of a transition of the automaton. The same proofs as in [Pin96, CPP93] show that:

Lemma 8. *L verifies (†)2 and (†)3′ iff G^1_A does not match P_2 nor $P_{3'}$.*

For (†)1 we need a new graph $G^2_A = (V^2_A, E^2_A)$. V^2_A is now $Q \cup (Q \times Q)$. E^2_A is E^1_A together with arrows labeled in $(\Sigma^2 \times Q^2 \times \{0,1\}^2) \cup \Sigma$ connecting: a node (p_1,p_2) to a node q via an edge a if $\delta(\langle p_1,p_2\rangle, a) = q$, a node (p_1,p_2) to a node (q_1,q_2) via an edge $(a,b,p,q,1,0)$ if $\delta(\langle p,p_1\rangle, a) = q_1$ and $\delta(\langle p_2,q\rangle, b) = q_2$, and similarly for edges labeled $(a,b,p,q,0,1)$,$(a,b,p,q,0,0)$, $(a,b,p,q,1,1)$.

We extend the notion of pattern to allow nodes in a pattern to be labeled with either variables or pairs of variables ranging over Q. The notion of matching is extended in the obvious way. Let P_1 be the following pattern together with $q \neq q'$.



Lemma 9. *L verifies (†)1 iff G^2_A does not match P_1*

A simple modification of the argument in [CPP93] shows that matching this extended notion of pattern is in PTIME. From this, using Lemmas 7, 8, and 9, we have the desired result:

Proposition 1. *There is a* PTIME *algorithm that, given a minimal deterministic bottom-up tree automaton, checks whether the corresponding language is definable in FO.*

6 Conclusion and Open Issues

The main result presented here is the decidability of FO-definability in ranked trees. What about unranked trees? We believe that Theorem 1 remains true in this general case. We also believe that our characterization (and the decidability results that follow) extends to ω-trees. In addition to giving a decision procedure, the characterization here has been useful for demonstrating that certain queries are first-order; for example, it is has been used to prove that order-invariant first-order queries over trees are first-order expressible.

In the case of strings, replacing the aperiodicity condition in part 3 of (†) with a periodicity mod q condition (and ignoring the condition in part 1 of (†)) one obtains a characterization of FO extended with counting quantifiers allowing to count modulo q [Str94]. We believe that modifying (†) in the same way should yield a characterization of FO with counting mod q in the case of ranked trees. The class LT of languages is defined as for LTT but without the threshold. That is, one can check the occurrence or absence of a pattern in a string but can no longer count the number of occurrences. We are considering how to modify our axioms to characterize LT. In the string case a characterization of this class is obtained by adding an "idempotent condition" on top of the commutative condition for blocks separated by identical idempotents. That is, the equation $(e \cdot s \cdot e)^2 = e \cdot s \cdot e$ needs to be added to (†). This does not extend directly to trees, since additional constraints are necessary to take care of the horizontal behavior of trees.

References

[BP89] D. Beauquier and J-E. Pin. Factors of words. In *Proc. of Intl. Coll. on Automata, Languages and Programming*, pages 63–79, 1989.

[BW04] M. Bojanczyk and I. Walukiewicz Characterizing EF and EX tree logics. In *CONCUR*, pages 131-145, 2004.

[Bc60] J. Bchi. Weak second-order logic and finite automata. *S. Math. Logik Grunlagen Math.*, 6:66–92, 1960.

[CH91] S. Cho and D T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991.

[CPP93] J. Cohen, D. Perrin, and J-E. Pin. On the expressive power of temporal logic for finite words. *Journal of Computer and Science Systems*, 46(1993):271–294, 1993.

[Don70] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[EW03] Z. Ésik and P. Weil. On logically defined recognizable tree languages. In *FSTTCS*, pages 195-207, 2003.

[Heu91] U. Heuter. First-order properties of trees, star-free expressions, and aperiodicity. *Informatique Thorique et Applications*, 25:125–146, 1991.

[Lib04] L. Libkin. *Elements of finite model theory*. Springer, 2004.

[MP71] R. McNaughton and S. Papert. *Counter-free Automata*. MIT Press, 1971.

[Pin96] J-E. Pin. The expressive power of existential first order sentences of Bchi's sequential calculus. In *Proc. of Intl. Coll. on Automata, Languages and Programming*, pages 300–311, 1996.

[Pot95] A. Potthoff. First-order logic on finite trees. In *Theory and Practice of Software Development (TAPSOFT)*, pages 125–139, 1995.

[PT93] A. Potthoff and W. Thomas. Regular tree languages without unary symbols are star-free. In *Fundamentals of Computation Theory (FCT)*, pages 396–405, 1993.

[Sch65] M. P. Schtzenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[Str94] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhuser, 1994.

[Tho97] W. Thomas. *Handbook of Formal Languages*, volume 3, chapter 7. Springer, 1997.

[TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata with an application to a decision problem of second order logic. *Math. Syst. Theory*, 2:57–82, 1968.

[TW85] D. Thérien and A. Weiss. Graph congruences and wreath products *J. Pure and Applied Algebra*, 36:205–215, 1985.

[Wil96] T. Wilke. An algebraic characterization of frontier testable tree languages. *Theoretical Computer Science*, 154(1):85–106, 1996.

Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations

Kousha Etessami[1] and Mihalis Yannakakis[2]

[1] School of Informatics, University of Edinburgh
[2] Department of Computer Science, Columbia University

Abstract. We introduce and study Recursive Markov Chains (RMCs), which extend ordinary finite state Markov chains with the ability to invoke other Markov chains in a potentially recursive manner. They offer a natural abstract model for probabilistic programs with procedures, and are a probabilistic version of Recursive State Machines. RMCs generalize Stochastic Context-Free Grammars (SCFG) and multi-type Branching Processes, and are intimately related to Probabilistic Pushdown Systems. We focus here on termination and reachability analysis for RMCs. We present both positive and negative results for the general class of RMCs, as well as for important subclasses including SCFGs.

1 Introduction

We introduce and study *Recursive Markov Chains* (RMCs), a natural model for systems involving probability and recursion. Informally, a Recursive Markov Chain consists of a collection of finite state component Markov chains (MC) which can call each other in a potentially recursive manner. RMCs are a natural probabilistic version of Recursive State Machines (RSMs) ([1, 4]), with transitions labeled by probabilities. RSMs and closely related models like Pushdown Systems (PDSs) have been studied extensively in recent research on model checking and program analysis, because of their applications to verification of sequential programs with procedures. RMCs offer a natural abstract model for probabilistic procedural programs. Probabilistic models of programs are of interest for at least two reasons. First, the program may use randomization, in which case the transition probabilities reflect the random choices of the algorithm. Second, we may want to model and analyse a program under statistical conditions on its behaviour (e.g., based on profiling statistics or on statistical assumptions), and to determine the probability of properties of interest, e.g., that the program terminates, and/or that it terminates in a certain state.

Recursive Markov chains are an interesting and important model in their own right. RMCs provide a succinct finite representation for a natural class of denumerable Markov Chains that generalize other well-studied models, such as Stochastic Context-Free Grammars (SCFGs) and Multitype Branching Processes, and they are intimately related to Probabilistic Pushdown Systems (pPDSs).

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 340–352, 2005.

Single-exit RMCs, the special case where each component MC has exactly 1 exit (terminating state), are in a precise sense "equivalent" to SCFGs. SCFGs have been studied extensively, especially in the Natural Language Processing (NLP) community, since the 1970s (see, e.g., [23]). Their theory is directly connected with the theory of *multi-type Branching Processes* (MT-BPs) initiated by Kolmogorov and others (see, e.g., [18, 20]). BPs are an important class of stochastic processes. Their theory dates back to the 19th century and the work of Galton and Watson on population dynamics. Multi-type BPs and SCFGs have been applied in a wide variety of stochastic contexts besides NLP, including population genetics ([19]), models in molecular biology including for RNA ([27]), and the study of nuclear chain reactions ([13]). Many variants and extensions of MT-BPs have also been studied. Despite this extensive study, some basic algorithmic questions about SCFGs and MT-BPs have not been satisfactorily answered. For example, is the probability of termination of a given SCFG (i.e., the probability of its language) or MT-BP (i.e. the so-called probability of extinction) $\geq p$? Is it $= 1$? Can these questions be decided in polynomial time in general? What if there are only a constant number of types in the branching process (non-terminals in the grammar)? RMCs form a natural generalization of SCFGs and MT-BPs, however their underlying stochastic processes appear not to have been studied in their own right in the rich branching process literature.

Our goal in this paper is to provide efficient algorithms and to determine the computational complexity of reachability analysis for RMCs. Namely, we are interested in finding the probability of eventually reaching a given terminating vertex of the RMC starting from a given initial vertex. As with ordinary Markov chains, such algorithms are a core building block for model checking and other analyses of these probabilistic systems. For SCFGs (MT-BPs), this amounts to an algorithm for determining the probability of termination (extinction).

It turns out reachability probabilities for RMCs are captured by the Least Fixed Point (LFP) solution of certain monotone systems of nonlinear polynomial equations. We observe that these solutions can be irrational even for SCFGs, and not solvable by radicals. Thus we can't hope to compute them exactly.

By appealing to the deep results on the complexity of decision procedures for the Existential Theory of Reals (see, e.g., [6]), we show that for general RMCs we can decide in PSPACE whether the probability is $\leq p$, or $= p$, for some rational $p \in [0, 1]$, and we can approximate the probabilities to within any given number of bits of precision. For an SCFG where the number of non-terminals is bounded by a constant (or a MT-BP with a bounded number of types), we can answer these questions in polynomial time. We show that this holds more generally, for RMCs where the total number of entries and exits of all components is bounded by a constant. Furthermore, we show that for single-exit RMCs with an arbitrary number of components (i.e., for general SCFGs), we can decide if the probability is *exactly* 1 in P-time.

The monotone nonlinear systems for RMCs give rise to a natural iterative numerical algorithm with which to approximate the LFP. Namely, the system

of equations has the form $\mathbf{x} = P(\mathbf{x})$, where $\mathbf{x}$ is a vector, such that the vector of probabilities we are seeking is given by $\lim_{k\to\infty} P^k(\mathbf{0})$, where $P^1(\mathbf{0}) = P(\mathbf{0})$ and $P^{k+1}(\mathbf{0}) = P(P^k(\mathbf{0}))$. We show that this iteration can be very slow to converge. Remarkably however, we show that a multi-dimensional Newton's method converges monotonically to the LFP on a decomposed version of the system, and constitutes a rapid "acceleration" of the standard iteration. Note that in other contexts, in general Newton's method is not guaranteed to converge; but when it does converge, typically it converges very fast. We thus believe that in our context Newton provides an efficient practical method for numerically estimating these probabilities for all RMCs.

Lastly, for "lower bounds", we show that one can not hope to improve our PSPACE upper bounds for RMCs without a major breakthrough, by establishing a connection to a fundamental open problem in the complexity of numerical computation: the square-root sum problem. This problem is known to be in PSPACE, but its containment even in NP is a longstanding open problem first posed in 1976 ([16]), with applications to a number of areas including computational geometry. We show the square-root sum problem is polynomial-time reducible to the problem of determining for an SCFG whether the termination probability is $\leq p$ for some $p \in [0, 1]$, and also to the problem of determining whether a 2-exit RMC terminates with probability 1.

Due to space limitations, all proofs are omitted. Please see the full paper [15].

Related Work. The work in the verification literature on algorithmic analysis of pushdown systems is extensive (see, e.g., [3, 10]). Recursive state machines were introduced in [1, 4] as a more direct graphical model of procedural programs, and their algorithmic verification questions were thoroughly investigated. A recent work directly related to ours is by Esparza, Kucera, and Mayr [12]. They consider model checking for probabilistic pushdown systems (pPDSs). pPDSs and RMCs are intimately related models, and there are efficient P-time translations from one to the other. As part of their results [12] show decidability of reachability questions for pPDSs by appealing to results on the theory of reals. In particular, they derive EXPTIME upper bounds for reachability. Our work was done independently and concurrently with theirs (a preliminary draft of our work was made available to the authors of [12] after we learned of their work). In any case, although their work overlaps briefly with ours, their primary focus is on decidability (rather than precise complexity) of model checking problems for a probabilistic branching-time temporal logic, PCTL. Our work also answers several complexity questions raised in their work. As mentioned earlier, SCFGs have been studied extensively in the NLP literature (see, e.g., [23]). In particular, the problem of *consistency* of a SCFG (whether it terminates with probability 1) has been studied, and its connection to the extinction problem for MT-BPs is well known [18, 7, 17, 9]. However, none of the relevant references provide a complete algorithm and characterization for consistency. Another work on pPDSs is [2]. They do not address algorithmic questions for reachability.

2 Basics

A *Recursive Markov Chain (RMC)*, A, is a tuple $A = (A_1, \ldots, A_k)$, where each *component graph* $A_i = (N_i, B_i, Y_i, En_i, Ex_i, \delta_i)$ consists of:

- A set N_i of *nodes*.
- A subset of *entry* nodes $En_i \subseteq N_i$, and a subset of *exit* nodes $Ex_i \subseteq N_i$.
- A set B_i of *boxes*. Let $B = \cup_{i=1}^{k} B_i$ be the (disjoint) union of all boxes of A.
- A mapping $Y_i : B_i \mapsto \{1, \ldots, k\}$ that assigns to every box (the index of) one of the components, $A_1, \ldots, A_k$. Let $Y = \cup_{i=1}^{k} Y_i$ denote the map $Y : B \mapsto \{1, \ldots, k\}$ which is consistent with each Y_i, i.e., $Y|_{B_i} = Y_i$, for $1 \leq i \leq k$.
- 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*, $Return_b = \{(b, ex) \mid ex \in Ex_{Y(b)}\}$.
- A transition relation δ_i, where transitions are of the form $(u, p_{u,v}, v)$ where:
  1. the source u is either a non-exit node $u \in N_i \setminus Ex_i$, or a return port $u = (b, ex) \in Return_b$, where $b \in B_i$.
  2. The destination v is either a non-entry node $v \in N_i \setminus En_i$, or a call port $v = (b, en) \in Call_b$, where $b \in B_i$.
  3. $p_{u,v} \in \mathbb{R}_{>0}$ is the transition probability from u to v. (We assume $p_{u,v}$ is rational.)
  4. *Consistency of probabilities*: for each u, $\sum_{\{v' | (u, p_{u,v'}, v') \in \delta_i\}} p_{u,v'} = 1$, unless u is a call port or exit node, neither of which have outgoing transitions, in which case by default $\sum_{v'} p_{u,v'} = 0$.

We will use the term *vertex* of A_i to refer collectively to its set of nodes, call ports, and return ports, and we denote this set by Q_i, and we let $Q = \bigcup_{i=1}^{k} Q_i$ be the set of all vertices of the RMC A. That is, the transition relation δ_i is a set of probability-weighted directed edges on the set Q_i of vertices of A_i. Let $\delta = \cup_i \delta_i$ be the set of all transitions of A.

An RMC A defines a global denumerable Markov chain $M_A = (V, \Delta)$ 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. More precisely, the states V 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 $\langle \beta, u \rangle \in V$ and $(u, p_{u,v}, v) \in \delta$, then $\langle \beta, v \rangle \in V$ and $(\langle \beta, u \rangle, p_{u,v}, \langle \beta, v \rangle) \in \Delta$
3. if $\langle \beta, (b, en) \rangle \in V$, $(b, en) \in Call_b$, then $\langle \beta b, en \rangle \in V$ and $(\langle \beta, (b, en) \rangle, 1, \langle \beta b, en \rangle) \in \Delta$
4. if $\langle \beta b, ex \rangle \in V$, $(b, ex) \in Return_b$, then $\langle \beta, (b, ex) \rangle \in V$ and $(\langle \beta b, ex \rangle, 1, \langle \beta, (b, ex) \rangle) \in \Delta$

Item 1 corresponds to the possible initial states, 2 corresponds to a transition within a component, 3 is when a new component is entered via a box, 4 is when the process exits a component and control returns to the calling component.

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. If we wish to view M_A as a proper Markov chain, we can consider the terminating states as absorbing states of M_A, with a self-loop of probability 1.

RMCs where the *call graph* between components forms an acyclic graph are called *Hierarchical Markov Chains* (HMCs). In this special case M_A is finite, but can be exponentially larger than the HMC which specifies it.

The Central Reachability Questions. Our goal is to answer termination and reachability questions for RMCs. Given a vertex $u \in Q_i$ and an exit $ex \in Ex_i$, both in the same component A_i, let $q^*_{(u,ex)}$ denote the probability of eventually reaching the terminating state $\langle \epsilon, ex \rangle$, starting at the initial state $\langle \epsilon, u \rangle$. Computing probabilities $q^*_{(u,ex)}$ will allow us to efficiently obtain other probabilities.

For a given pair of vertices $u, v \in Q$ of the RMC, let $[u, v]$ denote the probability that starting at state $\langle \epsilon, u \rangle$ we will eventually reach a state $\langle \beta, v \rangle$ for some $\beta \in B^*$. We can obtain the probabilities $[u, v]$ based on the probabilities $q^*_{(u,ex)}$. One way to do this is as follows: add a new special exit $ex^\star_i$ to every component A_i of the RMC, remove the out-edges from $v \in Q_j$ and instead add a transition $v \xrightarrow{1} ex^\star_j$, and add transitions $w \xrightarrow{1} ex^\star_h$, for every return port $w = (b, ex^\star_k)$, where $b \in B_h$. Now, for $u \in Q_i$, $[u, v]$ in the original RMC is equal to $q^*_{(u,ex^\star_i)}$ in the revised RMC. (Intuitively, when we encounter v we "raise an exception", pop the entire call stack, and exit the system.) There is also a more involved way to obtain the probability $[u, v]$ from the probabilities $q^*_{(u,ex)}$ without increasing the number of exits in any component. We can thus focus on finding efficient algorithms for the following central questions:

(1) *Qualitative* reachability problem: Is $\mathbf{q}^*_{(u,ex)} = 1$?
(2) *Quantitative* reachability problems: Given $r \in [0, 1]$, is $q^*_{(u,ex)} \geq r$? Is $q^*_{(u,ex)} = r$? Compute or approximate the exact probabilities $q^*_{(u,ex)}$.

Single-Exit RMCs and Stochastic Context-Free Grammars. A *Stochastic Context-Free Grammars* (SCFG) is a tuple $G = (T, V, R, S_1)$, where T is a set of *terminal* symbols, $V = \{S_1, \ldots, S_k\}$ is a set of *non-terminals*, and R is a set of rules $S_i \xrightarrow{p} \alpha$, where $S_i \in V$, $p \in [0, 1]$, and $\alpha \in (V \cup T)^*$, such that for every non-terminal S_i, $\sum_{\langle p_j | (S_i \xrightarrow{p_j} \alpha_j) \in R \rangle} p_j = 1$. Let $p(S_j)$ denote the probability that the grammar, started at S_j, will terminate and produce a finite string. SCFGs are "equivalent" to single-exit RMCs in the following sense.

Proposition 1. *Every SCFG G can be transformed to a 1-exit RMC A, such that $|A| \in O(|G|)$, and there is a bijection from non-terminals S_j in G to components A_j of A, each with a single entry en_j and exit ex_j, such that $p(S_j) = \mathbf{q}^*_{(en_j, ex_j)}$, for all j. Conversely, every 1-exit RMC A can be transformed to a SCFG G of size $O(|A|)$, such that there is a map from vertices u to non-terminals S_u of G, such that if ex is u's component exit, then $\mathbf{q}^*_{(u,ex)} = p(S_u)$.*

The System of Nonlinear Equations Associated with an RMC. Consider the probabilities $q^*_{(u,ex)}$ as unknowns. We can set up a system of (nonlinear) polynomial equations, such that these probabilities must be a solution of the system, and in fact precisely the *Least Fixed Point* solution (which we define). Let us use a variable $x_{(u,ex)}$ for each unknown probability $q^*_{(u,ex)}$. We will often find it convenient to index the variables $x_{(u,ex)}$ according to a fixed order, so we

can refer to them also as $x_1, \ldots, x_n$, with each $x_{(u,ex)}$ identified with x_j for some j. We thus obtain a vector of variables: $\mathbf{x} = (x_1\ x_2 \ldots x_n)^T$.

Definition 1. *Given RMC $A = (A_1, \ldots, A_k)$, we define a system of polynomial equations, S_A, over the variables $x_{(u,ex)}$, where $u \in Q_i$ and $ex \in Ex_i$, for $1 \leq i \leq k$. The system contains one equation of the form $x_{(u,ex)} = P_{(u,ex)}(\mathbf{x})$, for each variable $x_{(u,ex)}$. Here $P_{(u,ex)}(\mathbf{x})$ denotes a multivariate polynomial with positive rational coefficients. There are 3 cases to distinguish, based on the "type" of vertex u:*

1. *Type I: $u = ex$. In this case: $x_{(ex,ex)} = 1$.*
2. *Type II: either $u \in N_i \setminus \{ex\}$ or $u = (b, ex')$ is a return port. In these cases:*
$$x_{(u,ex)} = \sum_{\{v|(u,p_{u,v},v)\in\delta\}} p_{u,v} \cdot x_{(v,ex)}.$$
(If u has no outgoing transitions, this equation is by definition $x_{(u,ex)} = 0$.)
3. *Type III: $u = (b, en)$ is a call port. In this case:*
$$x_{((b,en),ex)} = \sum_{ex' \in Ex_{Y(b)}} x_{(en,ex')} \cdot x_{((b,ex'),ex)}$$

In vector notation, we denote $S_A = (x_j = P_j(\mathbf{x}) \mid j = 1, \ldots, n)$ by: $\mathbf{x} = P(\mathbf{x})$.

Note we can easily construct the system $\mathbf{x} = P(\mathbf{x})$ from A in polynomial time: $P(\mathbf{x})$ has size $O(|A||Ex|^2)$, where $|Ex|$ denotes the maximum number of exits of any component of A. We will now identify a particular solution to the systems $\mathbf{x} = P(\mathbf{x})$, called the *Least Fixed Point* (LFP) solution, which gives us precisely the probabilities we are after. For vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, define the partial-order $\mathbf{x} \preceq \mathbf{y}$ to mean that $x_j \leq y_j$ for every coordinate j. For $D \subseteq \mathbb{R}^n$, we call a mapping $H : \mathbb{R}^n \mapsto \mathbb{R}^n$ *monotone* on D, if: for all $\mathbf{x}, \mathbf{y} \in D$, if $\mathbf{x} \preceq \mathbf{y}$ then $H(\mathbf{x}) \preceq H(\mathbf{y})$. Define $P^1(\mathbf{x}) = P(\mathbf{x})$, and define $P^k(\mathbf{x}) = P(P^{k-1}(\mathbf{x}))$, for $k > 1$.

Recall that $q^*_{(u,ex)}$ denotes the probability of eventually reaching $\langle \epsilon, ex \rangle$ starting at $\langle \epsilon, u \rangle$ in M_A. Let $\mathbf{q}^* \in \mathbb{R}^n$ denote the corresponding n-vector of probabilities (using the same indexing as used for $\mathbf{x}$). For $k \geq 0$, let $\mathbf{q}^k$ denote the n-vector of probabilities where $q^k_{(u,ex)}$ is the probability of reaching $\langle \epsilon, ex \rangle$ starting at $\langle \epsilon, u \rangle$ *in at most k steps* of M_A, meaning via a path in M_A of length at most k. Let $\mathbf{0}$ ($\mathbf{1}$) denote the n-vector consisting of 0 (respectively, 1) in every coordinate. Define $\mathbf{x}^0 = \mathbf{0}$, and for $k \geq 1$, define $\mathbf{x}^k = P(\mathbf{x}^{k-1}) = P^k(\mathbf{0})$.

Theorem 1. *Let $\mathbf{x} = P(\mathbf{x})$ be the system S_A associated with RMC A.*

1. *$P : \mathbb{R}^n \mapsto \mathbb{R}^n$ is monotone on $\mathbb{R}^n_{\geq 0}$. Hence, for $k \geq 0$, $\mathbf{0} \preceq \mathbf{x}^k \preceq \mathbf{x}^{k+1}$.*
2. *For all $k \geq 0$, $\mathbf{q}^k \preceq \mathbf{x}^{k+1}$.*
3. *$\mathbf{q}^* = P(\mathbf{q}^*)$. In other words, $\mathbf{q}^*$ is a* fixed point *of the map P.*
4. *For all $k \geq 0$, $\mathbf{x}^k \preceq \mathbf{q}^*$.*
5. *$\mathbf{q}^* = \lim_{k\to\infty} \mathbf{x}^k$.*
6. *For all $\mathbf{q}' \in \mathbb{R}^n_{\geq 0}$, if $\mathbf{q}' = P(\mathbf{q}')$, then $\mathbf{q}^* \preceq \mathbf{q}'$.*
In other words, $\mathbf{q}^*$ is the Least Fixed Point, LFP(P), *of $P : \mathbb{R}^n_{\geq 0} \mapsto \mathbb{R}^n_{\geq 0}$.*

We have thus identified $\mathbf{q}^*$ as $\mathrm{LFP}(P) = \lim_{k\to\infty} \mathbf{x}^k$. We can (naively) view Theorem 1 as giving an iterative algorithm to compute LFP(P), by computing the iterates $\mathbf{x}^k = P^k(\mathbf{0})$, $k \to \infty$, until we think we are "close enough". (But how do we know? Please read on.) We now observe several unfortunate properties of S_A that present obstacles for efficiently computing LFP(P).

Theorem 2. *All following RMCs, except the HMCs in (4), have one component, one entry en, and one exit ex.*

1. Irrational probabilities*: there is a RMC, A, such that the probability* $\mathbf{q}^*_{(en,ex)}$ *is an irrational number, and is in fact not "solvable by radicals". Thus, computing* LFP(P) exactly *is not possible in general.*
2. Slow convergence*: there is a RMC such that it requires an exponential number of iterations,* 2^{k-3}*, to gain k bits of precision.*
3. Qualitative questions not purely structure-dependent*: there are 2 "structurally" identical RMCs,* A' *and* A''*, that only differ in values of non-zero transition probabilities, but* $\mathbf{q}^*_{(en,ex)} = 1$ *in* A'*, while* $\mathbf{q}^*_{(en,ex)} < 1$ *in* A''*.*
4. Very small & very large probabilities*: There is a HMC, with* $m+1$ *components, and of total size* $O(m)$*, where component* A_m *has entry* en_m *and two exits* ex'_m *and* ex''_m*, such that* $\mathbf{q}^*_{(en_m,ex'_m)} = \frac{1}{2^{2^m}}$ *and* $\mathbf{q}^*_{(en_m,ex''_m)} = 1 - \frac{1}{2^{2^m}}$*.*

These facts illustrate some of contrasts between RMCs and finite Markov chains (MCs). For example, for finite MCs reachability probabilities are rational values that are efficiently representable and computable; moreover, qualitative questions, such as whether a state is reached with probability 1, only depend on the structure (edges) of the MC, and not on values of transition probabilities.

For RMC $A = (A_1, \ldots, A_k)$, let $\theta = \max_{i \in \{1,\ldots,k\}} \min\{|En_i|, |Ex_i|\}$. Note that $q^*_{(u,ex)} = 0$ iff there is no path in the *graph* of M_A from $\langle \epsilon, u \rangle$ to $\langle \epsilon, ex \rangle$. Reachability in RSMs was studied in [1, 4], where it was shown that the problem can be decided in $O(|A|\theta^2)$ time, thus:

Theorem 3. *(see [1, 4]) Given RMC A, we can determine in time* $O(|A|\theta^2)$*, for all vertices u and exits ex, whether or not* $\mathbf{q}^*_{(u,ex)} = 0$*.*

3 RMCs and the Existential Theory of Reals

We now show that the central reachability questions for RMCs can be answered by appealing to algorithms for deciding the *Existential Theory of the Reals*, $\boldsymbol{ExTh}(\mathbb{R})$. This consists of sentences in prenex form: $\exists x_1, \ldots, x_n R(x_1, \ldots, x_n)$, where R is a boolean combination of "atomic predicates" of the form $f_i(\mathbf{x}) \Delta 0$, where f_i is a multivariate polynomial with rational coefficients over the variables $\mathbf{x} = x_1, \ldots, x_n$, and Δ is a comparison operator $(=, \neq, \geq, \leq, <, >)$.

Beginning with Tarski, algorithms for deciding the First-Order Theory of Reals, $\boldsymbol{Th}(\mathbb{R})$, and its existential fragment $\boldsymbol{ExTh}(\mathbb{R})$, have been deeply investigated. In the current state of the art, it is known that $\boldsymbol{ExTh}(\mathbb{R})$ can be decided in PSPACE [8, 25, 5]. Furthermore it can be decided in exponential time, where the exponent depends (linearly) only on the number of variables; thus for a fixed number of variables the algorithm runs in polynomial time.

Suppose we want to decide whether a rational vector $\mathbf{c} = [c_1, \ldots, c_n]^T$ is $LFP(P)$. Consider the sentence: $\varphi \equiv \exists x_1, \ldots, x_n \bigwedge_{i=1}^n P_i(x_1, \ldots, x_n) = x_i \wedge \bigwedge_{i=1}^n x_i = c_i$. φ is true iff $\mathbf{c} = P(\mathbf{c})$. To guarantee that $\mathbf{c} = \text{LFP}(P)$, we additionally need: $\psi \equiv \exists x_1, \ldots, x_n \bigwedge_{i=1}^n P_i(x_1, \ldots, x_n) = x_i \wedge \bigwedge_{i=1}^n 0 \leq x_i \wedge \bigvee_{i=1}^n x_i < c_i$.

ψ is false iff there is no solution $\mathbf{z} \in \mathbb{R}^n_{\geq 0}$ to $\mathbf{x} = P(\mathbf{x})$ such that $\mathbf{c} \not\preceq \mathbf{z}$. Hence, to decide whether $\mathbf{c} = \mathrm{LFP}(P)$, we only need two queries to a decision procedure for $\boldsymbol{ExTh}(\mathbb{R})$. Namely, we check that φ is true, and hence $\mathbf{c} = P(\mathbf{c})$, and we then check that ψ is false, and hence $\mathbf{c} = \mathrm{LFP}(P)$. Note that all multi-variate polynomials in our systems $\mathbf{x} = P(\mathbf{x})$ have (multivariate) degree $d \leq 2$.

Theorem 4. *Given a RMC A and given a vector of rational probabilities $\mathbf{c}$, there is a PSPACE algorithm to decide whether $\mathrm{LFP}(P) = \mathbf{c}$, as well as to decide whether $\mathbf{q}^*_j \ \Delta c_j$, for any comparison operator Δ. Moreover, the running time of the algorithm is $O(|A|^{O(1)} \cdot 2^{O(n)})$ where n is the number of variables in the system $\mathbf{x} = P(\mathbf{x})$. Hence the running time is polynomial if n is bounded.*

$\boldsymbol{ExTh}(\mathbb{R})$ gives us a way to ask questions like: "Is there a solution to $\mathbf{x} = P(\mathbf{x})$ where $a \leq x_i \leq b$?" for any rational numbers a and b, and if we wish, with either inequality replaced by strict inequality. Since $\mathbf{0} \preceq \mathrm{LFP}(P) \preceq \mathbf{1}$, we can use such queries in a "binary search" to "narrow in" on the value of each coordinate of $\mathrm{LFP}(P)$. Via simple modifications of sentences like ψ, we can gain one extra bit of precision on the exact value of c_i with each extra query to $\boldsymbol{ExTh}(\mathbb{R})$. So, if we want j bits of precision for each c_i, $i = 1, ...n$, we need to make $j \cdot n$ queries. The sizes of the queries do not vary by much: only with an additive factor of at most j-bits, to account for the constants a and b. This discussion yields:

Theorem 5. *Given RMC A, and a number j in unary, there is an algorithm that approximates the coordinates of $\mathrm{LFP}(P)$ to within j bits of precision in PSPACE. The running time is $O(j \cdot |A|^{O(1)} \cdot 2^{O(n)})$, where n is the number of variables in $\mathbf{x}$.*

With a more involved construction we can handle in polynomial time all RMCs that have a constant number of components, each with a constant number of entries and exits; the components themselves can be arbitrarily large.

Theorem 6. *Given an RMC with a bounded total number of entries and exits, we can decide in polynomial time whether $\mathrm{LFP}(P) = \mathbf{c}$, or whether $\mathbf{q}^*_j \ \Delta c_j$, for any comparison operator Δ, and we can approximate each probability to within any given number of bits of precision. In particular, this applies to SCFGs with a bounded number of terminals and MT-BPs with a bounded number of types.*

4 RMCs and Newton's Method

This section approaches efficient numerical computation of $\mathrm{LFP}(P)$, by studying how a classical numerical solution-finding method performs on the systems $\mathbf{x} = P(\mathbf{x})$. Newton's method is an iterative method that begins with an initial guess of a solution, and repeatedly "revises" it in an attempt to approach an actual solution. In general, the method may not converge to a solution, but when it does, it is typically fast. For example, for the bad RMC of Theorem 2.2, where the Iterative algorithm converges exponentially slowly, one can show that Newton's method converges exponentially faster, gaining one bit of precision per iteration.

- Preprocess the system $\mathbf{x} = P(\mathbf{x})$, eliminating all variables $x_{(u,ex)}$ where $\mathbf{q}^*_{(u,ex)} = 0$.
- Construct the DAG of SCCs, H, based on the remaining system of equations.
- While (there is a sink SCC, C, remaining in the DAG H)
  - If C is the trivial SCC, $C = \{1\}$, then associate the value 1 with this node.
    Else, run Newton's method, starting at $\mathbf{0}$, on the equations for the set of variables in C, where these equations are augmented by the values of previously computed variables.
    Stop if a fixed point is reached, or when approximate solutions for C are considered "good enough".
    Store these final values for the variables in C and substitute these values for those variables in all remaining equations.
  - remove C from the DAG.

Fig. 1. Decomposed Newton's method

Recall that, given a univariate polynomial $f(x)$ (or more generally, a univariate differentiable function), and an initial guess x_0 for a root of $f(x)$, Newton's method computes the sequence $x_0, x_1, \ldots, x_k$, where $x_{k+1} := x_k - \frac{f(x_k)}{f'(x_k)}$. There is a natural n-dimensional version of Newton's method (see, e.g, [26] and [24]). Given a suitably differentiable map $F : \mathbb{R}^n \mapsto \mathbb{R}^n$, we wish to find a solution to the system $F(\mathbf{x}) = \mathbf{0}$. Starting at some $\mathbf{x_0} \in \mathbb{R}^n$, the method works by iterating $\mathbf{x}_{k+1} := \mathbf{x}_k - (F'(\mathbf{x}_k))^{-1}F(\mathbf{x}_k)$, where $F'(\mathbf{x})$ is the *Jacobian matrix* of partial derivatives. For each $\mathbf{c} \in \mathbb{R}^n$, $F'(\mathbf{c})$ is a real-valued matrix whose (i, j) entry is the polynomial $\frac{\partial F_i}{\partial x_j}$ evaluated at $\mathbf{c}$. For the method to be defined, $F'(\mathbf{x}_k)$ must be invertible at each point $\mathbf{x}_k$ in the sequence. Even when the $\mathbf{x}_k$'s are defined and a solution exists, Newton's method need not converge, and diverges even for some univariate polynomials of degree 3. We already know one convergent iterative algorithm for computing LFP(P). Namely, computing the sequence $\mathbf{x}^j = P^j(\mathbf{0})$, $j \to \infty$. Unfortunately, we saw in Thm. 2 that this algorithm can be very slow. The question arises whether Newton's method, applied to $F(\mathbf{x}) = P(\mathbf{x}) - \mathbf{x}$, can guarantee convergence to LFP(P), and do so faster. That is essentially what we establish in this section.

We cannot in general obtain convergence of Newton's method for the entire system $\mathbf{x} = P(\mathbf{x})$ at once, because for instance the condition on invertibility of the Jacobian may not hold in general. But it turns out that such anomalous cases can be avoided: we first preprocess the system (in linear time in its size, by Theorem 3) to remove all variables $x_{(u,ex)}$ where $\mathbf{q}^*_{(u,ex)} = 0$. Then we form a graph G whose nodes are the remaining variables x_i and the constant 1, and whose edges are (x_i, x_j) if x_j appears in $P_i(\mathbf{x})$, and edge $(x_i, 1)$ if $P_i(\mathbf{x}) \equiv 1$. We *decompose* the graph (and the system) into strongly connected components (SCCs) and apply Newton's method separately on each SCC bottom-up, as shown in Fig.1. In Fig.1 we have not specified explicitly how many iterations are performed. For concreteness in the following theorem, suppose that we perform k iterations for every SCC. Let $\mathbf{x}_k$ be the resulting tuple of values.

Theorem 7. *In the Decomposed Newton's Method of Fig. 1, the sequence* $\mathbf{x}_k, k \to \infty$, *monotonically converges to* $\mathbf{q}^*$. *Moreover, for all* $k \geq 0$, $\mathbf{x}_k \succeq P^k(\mathbf{0})$.

From our proof it actually follows that Newton's method in general constitutes a rapid "acceleration" of the standard iteration, $P^k(\mathbf{0})$, $k \to \infty$. In particular, for finite MCs, which generate linear systems, the decomposed Newton's method converges in one iteration to LFP(P).

5 1-Exit RMCs and Consistency of SCFGs

An SCFG is called *consistent* if it generates a terminal string with probability 1. We provide a simple, concrete efficient algorithm to check consistency using the connection of SCFG's to 1-exit RMC's and to multi-type Branching Processes. MT-BPs model the growth of a population of objects of a number of distinct types. The probability of extinction of a type in a MT-BP is related to the probability of the language generated by a SCFG. Using this connection and classical results on branching processes, one can "characterize" the question of termination of a SCFG as a question related to eigenvalues of certain matrices associated with the SCFG (see, e.g., [18] and [7, 17]). These "characterizations" unfortunately often omit special uncovered cases (and sometimes contain errors, eg. [7]) and do not give a complete algorithm.

Our algorithm for checking SCFG consistency is outlined in Fig. 2. The matrix $B(1)$ in the algorithm is precisely the Jacobian matrix of the system of polynomials $P(\mathbf{x})$, from section 4, where we substitute 1 for every variable x_i. To finish the algorithm, we only need to show that we can test in polynomial time whether the spectral radius of a non-negative rational matrix $B(1)$ is > 1. There are a number of ways to show this. One is by appealing to the existential theory of the reals. By the Perron-Frobenius theorem (see [21]), the maximum magnitude eigenvalue of a non-negative matrix is always real. Recall that the eigenvalues of a matrix M are the roots of the characteristic polynomial $h(x) = Det(M - xI)$. This univariate polynomial can be computed in polynomial time, and we can test whether $\rho(B(1)) > 1$ by testing the 1-variable sentence in $\boldsymbol{ExTh}(\mathbb{R})$: $\exists x(x > 1 \ \wedge \ h(x) = 0)$. More efficiently, for the non-negative matrices $B(1)$ we can also use Linear Programming to decide whether $\rho(B(1)) > 1$. Furthermore, with a more involved algorithm (see [15]) we can classify in one pass the termination probability of all the nonterminals (and all vertices of a 1-exit RMC).

Input: A SCFG G, with start non-terminal S_1.
1. Remove all nonterminals unreachable from S_1.
2. If there is any "useless" nonterminal left (i.e., a nonterminal that does not derive any terminal string), return NO.
3. For the remaining SCFG, let ρ be the maximum eigenvalue of the matrix $B(1)$ (the Jacobian matrix of $P(\mathbf{x})$, evaluated at the all 1-vector).
   If $\rho > 1$ then return NO; otherwise (i.e., if $\rho \leq 1$) return YES.

Fig. 2. SCFG consistency algorithm

Theorem 8. *Given a 1-exit RMC, A, there is a polynomial time algorithm to determine, for each vertex u and exit ex, which of the following three cases holds: (1)* $\mathbf{q}^*_{(u,ex)} = 0$*, or (2)* $\mathbf{q}^*_{(u,ex)} = 1$*, or (3)* $0 < \mathbf{q}^*_{(u,ex)} < 1$*. In particular, we can test SCFG consistency in polynomial time.*

6 RMCs and the Square-Root Sum Problem

We show that the square-root sum problem is reducible to the SCFG quantitative reachability problem, and to the general RMC qualitative reachability problem. Let SQUARE-ROOT-SUM be the following problem: given $(d_1, \ldots, d_n) \in \mathbb{N}^n$ and $k \in \mathbb{N}$, decide whether $\sum_{i=1}^{n} \sqrt{d_i} \leq k$. The complexity of this problem is open since 1976. It is known to be contained in PSPACE (e.g., by appeal to $\boldsymbol{ExTh}(\mathbb{R})$), however, it is not even known to be contained in NP. It is a major open problem in the complexity of exact numerical algorithms, with applications in computational geometry and elsewhere. (See, e.g., [16, 29, 22].)

Let SCFG-PROB be the following problem: given a SCFG (with rational edge probabilities) and given a rational number $p \in [0, 1]$, decide whether the SCFG terminates (i.e., produces a finite string) with probability $\geq p$.

Theorem 9. *SQUARE-ROOT-SUM is P-time reducible to SCFG-PROB.*

Let 2-EXIT-SURE be the following problem: given a 2-exit RMC with rational edge probabilities, and an entry-exit pair *en* and *ex* of some component, decide whether $q^*_{(u,ex)} = 1$. We can modify the above construction to show:[1]

Theorem 10. *SQUARE-ROOT SUM is P-time reducible to 2-EXIT-SURE.*

7 Conclusions

We introduced Recursive Markov Chains, and studied basic algorithmic problems in their analysis involving termination and reachability. A wide variety of techniques came into play, from the existential theory of the reals, theory of branching processes, numerical computing, etc. A number of questions remain open, both for the problems we have investigated and for further directions. For example, we proved that Newton's method converges monotonically, and dominates the iterative algorithm. We expect that this is the practical way to approximate the probabilities, and we believe that in fact Newton gains i bits of precision in a polynomial number of iterations in the unit-cost real RAM model. Moreover, the reductions from the square-root sum problem to deciding whether a probability is $\leq p$ does not preclude the possibility that these probabilities can be *approximated* to i bits of precision in P-time without yielding a P-time solution to the square-root sum problem. Indeed, sum of square-roots itself can be so approximated using Newton's method.

[1] Theorem 10 has also been observed independently by J. Esparza & A. Kucera ([11]) based on a preliminary draft of this paper which included Theorem 9.

A number of further directions are worth pursuing, building upon this work. We have extended our methods to algorithms for the verification of linear time properties of RMC's ([14]). Another direction we are pursuing is the analysis of Recursive Markov Decision Processes and Recursive Stochastic Games.

References

1. R. Alur, K. Etessami, and M. Yannakakis. Analysis of recursive state machines. In *Proc. of 13th Int. Conf. on Computer-Aided Verification*, pp. 304–313, 2001.
2. S. Abney, D. McAllester, and F. Pereira. Relating probabilistic grammars and automata. *Proc. 37th Ann. Ass. for Comp. Linguistics*, pp. 542–549, 1999.
3. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: App's to model checking. In *CONCUR'97*, pp. 135–150, 1997.
4. M. Benedikt, P. Godefroid, and T. Reps. Model checking of unrestricted hierarchical state machines. In *Proc. of ICALP'01*, LNCS 2076, pp. 652–666, 2001.
5. S. Basu, R. Pollack, and M. F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. *J. of the ACM*, 43(6):1002–1045, 1996.
6. S. Basu, F. Pollack, and M. F. Roy. *Algorithms in Real Algebraic Geometry.* Springer, 2003.
7. T. L. Booth and R. A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, 22(5):442–450, 1973.
8. J. Canny. Some algebraic and geometric computations in PSPACE. In *Prof. of 20th ACM STOC*, pp. 460–467, 1988.
9. Z. Chi and S. Geman. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 1997.
10. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *12th CAV*, LNCS 1855, pp. 232–247, 2000.
11. J. Esparza and A. Kučera. *personal communication*, 2004.
12. Javier Esparza, Antonín Kučera, and Richard Mayr. Model checking probabilistic pushdown automata. In *LICS 2004*, 2004.
13. C. J. Everett and S. Ulam. Multiplicative systems, part i., ii, and iii. Technical Report 683,690,707, Los Alamos Scientific Laboratory, 1948.
14. K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic systems. Technical report, School of Informatics, U. of Edinburgh, 2004. *Submitted.*
15. K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations. Technical report, School of Informatics, University of Edinburgh, 2004. (Full paper).
16. M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *8th ACM STOC*, pp. 10–22, 1976.
17. U. Grenander. *Lectures on Pattern Theory, Vol. 1.* Springer-Verlag, 1976.
18. T. E. Harris. *The Theory of Branching Processes.* Springer-Verlag, 1963.
19. P. Jagers. *Branching Processes with Biological Applications.* Wiley, 1975.
20. A. N. Kolmogorov and B. A. Sevastyanov. The calculation of final probabilities for branching random processes. *Dokaldy*, 56:783–786, 1947. (Russian).
21. P. Lancaster and M. Tismenetsky. *The Theory of Matrices.* Academic Press, 2nd edition, 1985.
22. G. Malajovich. An effective version of Kronecker's theorem on simultaneous diophantine equations. Technical report, City U. of Hong Kong, 1996.

23. C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
24. J. M. Ortega and W.C. Rheinbolt. *Iterative solution of nonlinear equations in several variables*. Academic Press, 1970.
25. J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. parts i,ii, iii. *J. of Symbolic Computation*, pp. 255–352, 1992.
26. J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer, 1993.
27. Y. Sakakibara, M. Brown, R Hughey, I.S. Mian, K. Sjolander, R. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22(23):5112–5120, 1994.
28. B. A. Sevastyanov. The theory of branching processes. *Uspehi Mathemat. Nauk*, 6:47–99, 1951. (In Russian).
29. P. Tiwari. A problem that is easier to solve on the unit-cost algebraic ram. *Journal of Complexity*, pages 393–397, 1992.

Connectivity for Wireless Agents Moving on a Cycle or Grid*

Josep Díaz[1], Xavier Pérez[1], Maria J. Serna[1], and Nicholas C. Wormald[2]

[1] Dept. Llenguatges i Sistemes, Universitat Politecnica de Catalunya
{diaz, xperez, mjserna}@lsi.upc.edu
[2] Dept. Combinatorics and Optimization, University of Waterloo
nwormald@uwaterloo.ca

Abstract. We present a mathematical model to analyse the establishment and maintenance of communication between mobile agents. We assume that the agents move through a fixed environment modelled by a motion graph, and are able to communicate if they are at distance at most d. As the agents move randomly, we analyse the evolution in time of the connectivity between a set of w agents, asymptotically for a large number N of nodes, when w also grows large, and for different values of d. The particular topologies of the environment we study in this paper are the cycle and the toroidal grid.

1 Introduction

Consider a setting in which a large number of mobile agents can perform concurrent basic movements: ahead/behind/left/right, determining a grid pattern, or left/right, describing a line. Each agent can communicate directly with any other agents which are within a given distance d. This enables communications with agents at a further distance using several intermediate agents. At each step in time there is an ad-hoc network defined by the dynamic graph whose vertex set consists of the agents, with an edge between any two agents iff they are within the distance d of each other.

In this paper, we study the static and dynamic connectivity characteristics of communicating agents, in a framework called the *walkers model*, which we define as follows. A connected graph $G = (V, E)$ with $|V| = N$ is given, and a number w of *walkers* (agents). Also given is a "distance" d. A set W of walkers, with $|W| = w$, are placed randomly and independently on the vertices of G (a vertex may contain more than one walker). Each walker has a range d for communication; that is, two walkers w_1 and w_2 can communicate in one hop if the distance, in G, between the position of the walkers is at most d. Two walkers can communicate if they can reach each other by a sequence of such hops. In addition, each

* The work of the first three authors was partially supported by the EU under contract 001907 (DELIS). The fourth author was supported by the Canada Research Chairs program. The first author was partially supported by the Distinció per a la Recerca de la GC, 2002.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 353–364, 2005.

walker takes an independent standard random walk on G, i.e. moves at each time step to a eighbouring vertex, each neighbour chosen with equal probability.

The interesting features of the walkers model are encapsulated by the *graph of walkers*, $G_f[W]$. Here f is an random assignment $f : W \to V$ of walkers into the vertices of G. The vertices of $G_f[W]$ are the vertices in G that contain at least one walker, two vertices in $G_f[W]$ being joined by an edge iff they are at distance at most d in G. We refer to *components* of $G_f[W]$ in the usual sense, and call a component *simple* if it is formed by only one isolated vertex. We are interested in the probability of $G_f[W]$ being connected, or in the number of components and their sizes, with mild asymptotic restrictions on w and d.

Our primary goal with the walkers model is to characterise the dynamics of the connectivity of the network. To obtain enough information to do this, we first examine a variation of the model called the *static* model. This is a snapshot of the model at one point in time: there is merely the random function f, and we are interested in the distribution of the number of components, as well as other information which helps to answer the dynamic questions.

In the dynamic situation, there is an initial placement of walkers as in the static case, and at each time step, every walker simultaneously moves one step to a randomly selected neighbour vertex in G. This gives rise to a random graph process, where $G_{f_t}[W]$ denotes the graph of walkers at time $t = 0, 1, \ldots$. We are interested in studying the birth and death of components, and the sudden connection and disconnection of $G_{f_t}[W]$ in a dynamic setting.

We consider a sequence of graphs G with increasing numbers of vertices N, for N tending to infinity. The parameters w and d are functions of N. We restrict to the case $w \to \infty$ in order to avoid considering small-case effects. Of course we take $d \geq 1$. We make further restrictions on w and d in order to rule out non interesting cases, such as values of the parameters in which the network is asymptotically almost surely (a.a.s.) disconnected or a.a.s. connected. In this paper, we study the walkers model for two particular sequences of graphs G: the cycle of length N and the $n \times n$ toroidal grid. (In the case of the grid, we use the ℓ^1 distance, modelling the distance along roads in a city grid, but our approach is useful for other metrics.) Amongst other things, we determine the limiting probability of connectedness of the graph for the appropriate range of w and d, and also the expected time the graph spends in the connected state after it undergoes a transition from disconnected to connected (and similarly, for the disconnected state).

Nowadays, the *random geometric graph* has became the basic network model to study communication in wireless networks. In this model, the broadcasting stations (centre of the disk) could be distributed according to a Poisson process or uniformly at random on a bounded subset of $\mathbb{R}^2$, see for example [8]. For instance, it is known that for a random geometric graphs with n vertices and radius d (where d is a function of n), a.a.s. the graph is connected if $d \geq \sqrt{\log n/n}$ [7]. The theoretical results obtained not only on connectivity but also on other graph parameters like chromatic number, have help in dealing with more technological issues as efficient broadcasting algorithms for wireless ad-hoc networks

or message congestion, using as a basis the static situation. In the present paper, we obtain much sharper results on the static properties than previously obtained (albeit with a slightly different model). We give precise characterisations of connectivity for two graphs: the cycle and the toroidal grid with the Manhattan distance. In particular, given a grid with n^2 nodes, where we sprinkle uniformly at random (u.a.r.) w walkers on the nodes of grid, and given $d = o(n)$, we give a specific equation for the expected number μ of simple components in the grid, as the ratio w/n^2 tends to $0, c$ or ∞. From these expressions, we deduce the connectivity threshold for $G[W]$, when $\mu \to \infty$ (disconnected a.a.s.), when $\mu \to \Theta(1)$ (simple components except one isolated component) and when $\mu \to 0$ (connected a.a.s.).

In recent times, the big issue has been the mobility of the agents, where connections in the network are created and destroyed as the agents move further apart or closer together. There has been quite a bit of work designing efficient communication algorithms for motion agents, see [1, 6] for nice surveys. Most of the work is experimental [9]. Other interesting work deals with a data structure which is able at time t to decide quickly if two given stations are connected [5]. However, no theoretical work has been done with the global connectivity properties of dynamic wireless networks. Again we consider first the case of the cycle and the toroidal grid. In particular for the toroidal grid, we give firstly a precise estimation of the probability that, if the walkers are connected, they become disconnected in the next step (Theorem 8). Then using that result, we give precise asymptotic estimations on the expected number of steps that the grid will maintain connected (once it becomes connected) or disconnected, as the agents perform random movements on the nodes of the grid (Theorem 10). We believe that the study of the behaviour of multiple, simultaneous random walks is an important open problem which could have further applications in other fields of computer science. By lack of space, the proofs as well as the results on the grid for the l_2 norm, are left for the long version.

2 General Definitions and Basic Results

The reader is referred to [3] for the basic definitions and theorems on probability. As usual, for any integer n, we use $[n] = \{1, 2, \ldots, n\}$ and for any integers n and m, $[n]_m = n!/(n-m+1)!$.

For our specific work, we begin with some definitions and results which are common for all G. Define K to be the random variable counting the number of connected components, in $G_f[W]$, under random assignment f of walkers. Let ϱ denote the expected number of walkers at a vertex. Then $\varrho = w/N$. For $v \in V$, define h_v to be the number of vertices in G at distance at most d from v, and define $h = \min_{v \in V} h_v$. Notice that h is the minimum number of empty vertices in G around a simple component. (We say that a vertex v is *empty* if it contains no walkers, and *occupied* if it contains at least one.)

By considering the well known coupon-collector's problem, we observe that if $w = N \log N + \omega(N)$ then $G_f[W]$ is trivially a.a.s. connected due to every vertex

being occupied. Moreover, for the graphs G which we consider in this paper, if $h \in \Omega(N/\sqrt{w})$ then $G_f[W]$ is a.a.s. connected as well. This last claim will be seen in Observations 1 and 3. Thus, we consider throughout the paper $w < N \log N + O(N)$ and $h = o(N/\sqrt{w})$. In fact, our proofs will just require h to be $o(N)$.

As a key step in most of our proofs, we often need to compute the probability of having a certain configuration of walkers at a given time t. For this we apply Lemma 1 below.

Sometimes we also need the probability of certain configurations of walkers involving two consecutive time steps, in order to record the event that walkers jump to the appropriate place at time t. There is a convenient way to view this by partitioning every vertex v of G into as many *sub-vertices* as its degree, where each sub-vertex of v is associated with a different neighbour of v. Any given walker on one vertex will occupy the sub-vertex corresponding to the neighbour to which it will move at the next time step. Thus, a walker at v moving to a neighbour will occupy each of the sub-vertices of v with the same probability. In this case, we can also apply Lemma 1 with sub-vertices, since these form a static configuration (even though it encodes a dynamic transition).

Assign *size* 1 to all vertices in G. For a given sub-vertex in a vertex v with degree δ_v, its *size* will be $1/\delta_v$. Given a set A of vertices or sub-vertices, we define the *size* of A to be the sum of the sizes of its elements.

The following lemma comes by inclusion-exclusion.

Lemma 1. *Let $A_0, \ldots, A_m$ be pairwise disjoint sets of vertices (or sub-vertices) in G, with sizes $S_0, \ldots, S_m$ respectively. Let $N = |V(G)|$. If $\sum_{i=0}^{m} S_i = o(N)$, then*

$$\mathbf{P}\left(A_0 \text{ empty } \wedge \bigwedge_{i=1}^{m} A_i \text{ not empty}\right) \sim \left(1 - \frac{S_0}{N}\right)^w \prod_{i=1}^{m} \left(1 - e^{-S_i \varrho}\right).$$

To cover large sizes S (not necessarily $o(N)$) we need the following variation on the previous lemma.

Lemma 2. *Let A be a set of vertices in G of size S, and $v_1, \ldots, v_m$ vertices not in A, with $m \geq 1$. Assume $|V(G)| = N$ and $N - S \to \infty$. The probability that no vertex in A is occupied and $v_1, \ldots, v_m$ are all occupied is at most $p_0 p^{m-1} \alpha^w$ where $p_0 = 1 - e^{-\varrho/\alpha}$, $\alpha = 1 - S/N$ and*

$$p = \begin{cases} 1 & \text{if } \rho/\alpha \to \infty, \\ \rho/\alpha & \text{if } \rho/\alpha = O(1). \end{cases}$$

3 The Cycle

Let $G = C_N$ be the cycle with N nodes.

Observation 1. *Notice that for C_N, $h = 2d$. Cover the cycle with $\lceil \frac{N}{\lceil d/2 \rceil} \rceil$ paths of $\lceil d/2 \rceil$ vertices. If $h = \Omega(N/\sqrt{w})$, then the probability that some path is empty of walkers is at most*

$$\left\lceil \frac{N}{\lceil d/2 \rceil} \right\rceil \left(1 - \frac{\lceil d/2 \rceil}{N}\right)^w \leq O(\sqrt{w}) e^{-\Omega(\sqrt{w})} \to 0$$

Thus, a.a.s. each of these paths is occupied by at least one walker, and $G_f[W]$ is connected. So we assume for the rest of the section that $h = o(N/\sqrt{w})$, in fact the assumption $d = o(N)$ is all we require.

To study the connectivity of C_N, we introduce the concept of *hole*. Let us say there is a *hole* between two vertices u and v if u and v each contain at least one walker, but no vertex in the clockwise path from u to v contains a walker. We say that such a hole *follows* u, or that u is the *start vertex* of the hole. The number of internal vertices in a hole is its *size*. An *s-hole* is a hole whose size is at least s. Notice that at least two d-holes are needed to disconnect the walkers on C_N.

Let H be the random variable counting the number of d-holes, when w walkers are placed u.a.r. on C_N, and let $\mu_H = \mathbf{E}[H]$ be its expectation (just μ for short throughout this section).

Holes are closely related to components: trivially,

$$K = \begin{cases} 1 & \text{if } H = 0, 1, \\ H & \text{if } H > 1. \end{cases} \qquad \text{and thus} \quad \mathbf{E}[K] = \mathbf{P}(H = 0) + \mathbf{E}[H]. \tag{1}$$

Static Properties. Here, we study the connectivity of the graph of walkers $G_f[W]$ in the static situation, by analysing the behaviour of H. In view of (1), notice that if $\mu \to 0$ then $\mathbf{P}(K = 1) \to 1$, i.e. $G_f[W]$ is a.a.s. connected.

Theorem 1. *The expected number of holes satisfies*

$$\mu \sim N\left(1 - e^{-\varrho}\right)\left(1 - \frac{d}{N}\right)^w \sim \begin{cases} w\left(1 - \frac{d}{N}\right)^w & \text{if } \varrho \to 0, \\ N\left(1 - e^{-\varrho}\right)\left(1 - \frac{d}{N}\right)^w & \text{if } \varrho \to c, \\ N\left(1 - \frac{d}{N}\right)^w & \text{if } \varrho \to \infty. \end{cases}$$

Furthermore, if μ is bounded then H is asymptotically Poisson with mean μ, and if μ is bounded away from 0 then $\mu \sim N\left(1 - e^{-\varrho}\right) e^{-d\varrho}$.

The proof is done by estimating the factorial moments of H, using indicator variables.

From the second part of this theorem we can immediately obtain the probability that $G_f[W]$ is connected, when w walkers are scattered u.a.r. through the vertices of C_N.

Corollary 1. *If w walkers are placed u.a.r. on C_N, then $\mathbf{P}(K = 1) = e^{-\mu}(1 + \mu) + o(1)$.*

This implies that $G_f[W]$ is a.a.s. disconnected if $\mu \to \infty$, and a.a.s. connected if $\mu \to 0$. So we may restrict attention to $\mu = \Theta(1)$.

It is a simple matter determine from this and the first part of Theorem 1 the threshold value of ρ (or of d) at which the walkers graph becomes connected, and the probability of connectedness when around the threshold.

Dynamic Properties. Assume that from an initial random placement f of the walkers, at each step, every walker moves from its current position to one of its neighbours, with probability 1/2 of going either way. This is a standard random walk on the cycle for each walker. To study the connectivity properties of the dynamic graph of walkers we need to introduce some notation.

A *configuration* is an arrangement of the w walkers on the vertices of C_N. Consider the graph of configurations, where the vertices are the N^w different configurations. Any configuration has 2^w neighbours, and the dynamic process can be viewed as a random walk on the graph of configurations, in particular, a Markov chain $\mathcal{M}(N, w)$. If N is odd, then $\mathcal{M}(N, w)$ is ergodic. For the purposes of this extended abstract, we will treat in detail only the case of N odd, if non-ergodicity causes extra complications.

Observation 2. *For any fixed t, we can regard $G_{f_t}[W]$ as $G_f[W]$ in the static case. Hence, by Corollary 1, if $\mu \to 0$ or ∞ then, for t in any fixed bounded time interval, $G_{f_t}[W]$ is either a.a.s. connected or a.a.s. disconnected. So we assume $\mu = \Theta(1)$ for the remaining of the section, since we wish to study only the nontrivial dynamic situations.*

We define $H = H(t)$ to be the random variable that counts the number of d-holes at time t. Under the assumptions in Observation 2, for t in any fixed bounded time interval, $H(t)$ is asymptotically Poisson with expectation $\mu = \Theta(1)$ as studied in C_N.

For the dynamic properties of $G_{f_t}[W]$, we are interested in the probability that a new d-hole appears at a given time. Moreover, we require knowledge of this probability conditional upon the number of d-holes already existing.

If there is a d-hole from u to v at time t and all walkers at u and v move in the same direction on the next step, a new d-hole may appear following one of the neighbours of u (provided no new walkers move in to destroy this). These two d-holes, though being different, are related, and we prefer to think of them as the same thing. A similar comment applies when the exact size of a d-hole following u changes in one step. Define a *d-hole line* to be a maximal sequence of pairs $(h_1, t_1), \ldots, (h_l, t_l)$ where h_i is a d-hole existing at time t_i for $1 \leq i \leq l$, and such that $t_i = t_{i-1} + 1$ and the start vertex of h_i is adjacent to, or equal to, the start vertex of h_{i-1}, for $2 \leq i \leq l$. Fix two consecutive time steps t and $t + 1$. If $t_1 = t + 1$, we say that the line is *born* between t and $t + 1$, if $t_l = t$ the line *dies* between t and $t + 1$, and if $t = t_i$, $i \in \{1, \ldots, l - 1\}$ we say that the line *survives* during the interval $[t, t + 1]$. Note that the time-reversal of the process has a d-hole line born at vertex u between $t + 1$ and t iff the d-hole line dies at u between t and $t + 1$. Define $S = S(t)$, $B = B(t)$ and $D = (t)$ to be the number of d-hole lines surviving, being born and dying between t and $t + 1$. We obviously have $D(t) + S(t) = H(t)$ and $B(t) + S(t) = H(t + 1)$.

Theorem 2. *For t in any fixed bounded time interval, the random variables $S(t)$, $B(t)$ and $D(t)$ are asymptotically jointly independent Poisson, with the expectations*

$$\mathbf{E}[S] \sim \begin{cases} \mu & \text{if } \varrho \to 0, \\ \mu - \lambda & \text{if } \varrho \to c, \\ 3\mu e^{-\varrho} & \text{if } \varrho \to \infty, \end{cases} \quad \text{and} \quad \mathbf{E}[B(t)] = \mathbf{E}[D(t)] \sim \begin{cases} \frac{1}{2}\mu\varrho & \text{if } \varrho \to 0, \\ \lambda & \text{if } \varrho \to c, \\ \mu & \text{if } \varrho \to \infty, \end{cases}$$

where $\lambda = \left(1 - \dfrac{3e^{-\varrho} - e^{-\frac{3}{2}\varrho}}{1 + e^{-\frac{1}{2}\varrho}}\right)\mu$. *Here* $0 < \lambda < \mu$ *for* $\varrho \to c$.

Under the assumptions in Observation 2 and using this result, we can obtain several important consequences. The first gives the probability of having no holes at time t and at least one at time $t+1$. Note that more than one hole is required in C_N to disconnect $G_{f_t}[W]$. As $H(t) = S(t) + D(t)$, it follows from the theorem that $H(t)$ and $B(t)$ are asymptotically independent. We can write $\mathbf{P}(H(t+1) \geq 1 \wedge H(t) = 0) = \mathbf{P}(H(t) = 0 \wedge B(t) \geq 1)$, and immediately obtain the following.

Corollary 2. *Let* λ *be defined as in Theorem 2. The probability of having no holes at time* t *and at least one at time* $t+1$ *is given by*

$$\mathbf{P}\Big(H(t+1) \geq 1 \wedge H(t) = 0\Big) \sim \begin{cases} \frac{1}{2}\mu e^{-\mu}\varrho & \text{if } \varrho \to 0, \\ e^{-\mu}\left(1 - e^{-\lambda}\right) & \text{if } \varrho \to c, \\ e^{-\mu}(1 - e^{-\mu}) & \text{if } \varrho \to \infty, \end{cases}$$

We define the *lifespan* of a d-hole line as the number of time steps for which the line is alive. For any vertex v and time t, the random variable $L_{v,t}$ counts the lifespan of the d-hole line born at vertex v between times t and $t+1$. If this birth does not take place $L_{v,t}$ is defined to be 0. Note that the random variables $L_{v,t}$ are identically distributed for all v and t. Notice that the expected lifespan of any given d-hole line is bounded (this bound depending on N).

Theorem 3. *Let* λ *be defined as in Theorem 2. For any vertex* v *and time* t,

$$\mathbf{E}[L_{v,t}] \sim \begin{cases} 2\varrho^{-1} & \text{if } \varrho \to 0, \\ \frac{\mu}{\lambda} & \text{if } \varrho \to c, \\ 1 & \text{if } \varrho \to \infty, \end{cases}$$

The next theorem gives us the probability that there is one component at time t, but at least two at time $t+1$.

Theorem 4. *Let* λ *be defined as in Theorem 2. The probability that* $G_{f_t}[W]$ *is connected and that* $G_{f_{t+1}}[W]$ *is disconnected is given by*

$$\mathbf{P}\Big(H(t+1) \geq 2 \wedge H(t) < 2\Big) \sim \begin{cases} \frac{1}{2}\mu^2 e^{-\mu}\varrho & \text{if } \varrho \to 0, \\ e^{-\mu}\left(1 + \mu - (1 + \mu + \lambda + \lambda^2)e^{-\lambda}\right) & \text{if } \varrho \to c, \\ (1+\mu)e^{-\mu}(1 - (1+\mu)e^{-\mu}) & \text{if } \varrho \to \infty, \end{cases}$$

For any time t, let us condition upon $G_{f_t}[W]$ being disconnected at time t and becoming connected at $t+1$. Let T_C be a random variable measuring the time $G_{f_t}[W]$ remains connected. Similarly, let us condition upon $G_{f_t}[W]$ being connected at time t and becoming disconnected at $t+1$. Let T_D be a random variable measuring the time $G_{f_t}[W]$ remains disconnected. Their expectations do not depend on the chosen time t and are given in the following theorem.

Theorem 5. *Let λ be defined as in Theorem 2. The expected time that the graph of walkers $G_{f_t}[W]$ will be connected or disconnected (once it becomes so) is*

$$\mathbf{E}[T_C] \sim \begin{cases} 2\frac{1+\mu}{\mu^2}\varrho^{-1} & \text{if } \varrho \to 0, \\ \frac{1+\mu}{1+\mu-(1+\mu+\lambda+\lambda^2)e^{-\lambda}} & \text{if } \varrho \to c, \\ \frac{e^{\mu}}{e^{\mu}-(1+\mu)} & \text{if } \varrho \to \infty. \end{cases}$$

$$\text{and } \mathbf{E}[T_D] \sim \begin{cases} 2\frac{e^{\mu}-1-\mu}{\mu^2}\varrho^{-1} & \text{if } \varrho \to 0, \\ \frac{e^{\mu}-1-\mu}{1+\mu-(1+\mu+\lambda+\lambda^2)e^{-\lambda}} & \text{if } \varrho \to c, \\ \frac{e^{\mu}}{1+\mu} & \text{if } \varrho \to \infty, \end{cases}$$

4 The Grid

Let $G = T_N$ be the toroidal grid with $N = n^2$ nodes. We can refer to vertices by using coordinates in $\mathbb{Z}_n \times \mathbb{Z}_n$. For the grid we encounter significant new obstacles as compared to the cycle; see for instance the Geometric Lemma below. In T_N, we shall express the distance between two vertices as the minimal ℓ_1 distance of their coordinates. (In the long version, similar results are obtained for ℓ_2 norm).

Observation 3. *For T_N, and for $d < n/2$, the number of vertices at distance at most d from any given vertex is $h = 2d(d+1)$. For each $i, j < 8n/d$, let v_{ij} denote the point with coordinates $(\lfloor id/8 \rfloor, \lfloor jd/8 \rfloor)$. Let S_{ij} denote the set of grid points closer to v_{ij} than any of the other $v_{i'j'}$. Then there are $\Theta(N/d^2)$ disjoint sets S_{ij} each containing $\Theta(d^2)$ points. The probability that at least one of these S_{ij} is empty is at most $\Theta(N/d^2)(1-\Theta(d2/N))^w = O(\sqrt{w})e^{-\Omega(\sqrt{w})} \to 0$ if $h = \Omega(N/\sqrt{w})$. Thus, a.a.s. each of these pieces is occupied by at least one walker, and $G_f[W]$ is connected. So we assume for the rest of the section $h = o(N/\sqrt{w})$, or merely $h = o(N)$, i.e. $d = o(n)$.*

We wish to study the connection and disconnection of $G_f[W]$ in a similar way to the cycle. For the grid, the notion of hole does not help, and we deal directly with components. A major role is played by simple components, and we shall prove that, for the interesting values of the parameters, a.a.s. there only exist simple components and one giant one.

Let $\mathcal{C}$ be any given component. The *edges* of $\mathcal{C}$ are the straight edges joining occupied vertices in $\mathcal{C}$ at distance at most d. The associated *empty area* $A_{\mathcal{C}}$ is the set of vertices not in $\mathcal{C}$, but at distance at most d from some vertex in $\mathcal{C}$ (i.e. those vertices which must be free of walkers for $\mathcal{C}$ to exist as a component). The

exterior $\mathcal{E}_{\mathcal{C}}$ of $\mathcal{C}$ is all those vertices not in $\mathcal{C} \cup A_{\mathcal{C}}$. We partition $\mathcal{E}_{\mathcal{C}}$ into *external regions* as follows: two vertices belong to the same external region when they can be joined by a continuous arc not intersecting any edge of $\mathcal{C}$.

Recall that, in the terminology of planar maps, the *bounding cycle* of a face is a walk around the boundary of the face. Given an external region ${\mathcal{E}_{\mathcal{C}}}^i$, let $\mathcal{C}'$ be any connected subgraph of $\mathcal{C}$ that has no edges crossing and such that no vertices of $\mathcal{C}$ are contained in the face F of $\mathcal{C}'$ which contains ${\mathcal{E}_{\mathcal{C}}}^i$. Such graphs exist: for instance, take the spanning tree of $\mathcal{C}$ whose length (sum of lengths of edges) has been minimised in ℓ_1, and, subject to this, has the shortest Euclidean length. The bounding cycle of this face F is defined to be a *boundary walk* β in $\mathcal{C}$ with respect to ${\mathcal{E}_{\mathcal{C}}}^i$. Such a walk is *maximal* if the face F does not properly contain a face of some other subgraph of $\mathcal{C}$. We call a (directed) closed walk in $\mathcal{C}$ *regular* if, for each edge entering a vertex v, the next edge in the walk is the next edge in the clockwise direction around v.

For $i < n$, let us call a *column* of width i any subset of T_N defined by $\{a, \ldots, a+i-1\} \times \mathbb{Z}_n$. We define a *row* of height j similarly. Define a *rectangle* of width i and height j to be the intersection of a column of width i and a row of height j. Notice in a rectangle we can compare vertices inside according to their coordinates, and we shall use statments as v_1 *is more to the left than* v_2 or v_3 *is the uppermost vertex*.

We say that a component $\mathcal{C}$ with at least 2 vertices is a *rectangular component* (*r-component*) if all of its vertices, edges and empty area are contained in a rectangle in the torus of height and width at most $n-1$. In particular, this implies that $\mathcal{C}$ contains no nonseparating cycle of the torus. Otherwise, it is an *nr-component*. For a given r-component $\mathcal{C}$, we define its *origin* as the leftmost of the lower-most vertices of $\mathcal{C}$, with a canonical defintion of left and lower over a containing rectangle. The *outside region* of an r-component is the only external region of the component having points outside any containing rectagle.

Let X, Y and Z be the number of simple components, r-components and nr-components respectively, and put $K = X + Y + Z$. Let $Z = Z_1 + Z_2$, where Z_1 is the number of nr-components which cannot coexist with another nr-component and Z_2 counts those ones which can. Then $\mathbf{E}[Z] = \mathbf{P}(Z_1 = 1) + \mathbf{E}[Z_2]$. Set $\mu = \mu_X = \mathbf{E}[X]$, the expected number of simple components.

Static Properties. Let μ denote the expected number of simple components in the grid. The next theorem gives its value asymptotically.

Theorem 6. *The expected number of simple components satisfies*

$$\mu \sim N\left(1-e^{-\varrho}\right)\left(1-\frac{h}{N}\right)^w \sim \begin{cases} w\left(1-\frac{h}{N}\right)^w & \text{if } \varrho \to 0, \\ N(1-e^{-\varrho})\left(1-\frac{h}{N}\right)^w & \text{if } \varrho \to c, \\ N\left(1-\frac{h}{N}\right)^w & \text{if } \varrho \to \infty. \end{cases}$$

Furthermore, if μ is bounded then X is asymptotically Poisson with mean μ, whilst if μ is bounded away from 0 then $\left(1-\frac{h}{N}\right)^w \sim e^{-h\varrho}$ and we have $\mu \sim N\left(1-e^{-\varrho}\right)e^{-h\varrho}$.

The proof, analogous to the proof of Theorem 1, follows from Lemma 1.

From the previous theorem we can immediately obtain the probability of having no simple components, when w walkers are scattered u.a.r. throughout the vertices of T_N.

Corollary 3. *The probability of having no simple components is* $\mathbf{P}(X = 0) = e^{-\mu} + o(1)$. *Furthermore, if* $h\varrho = O(1)$ *then* $\mu \to \infty$ *and* $G_f[W]$ *is disconnected a.a.s.*

We may now restrict to the condition $h\varrho \to \infty$ in the study of r-components and nr-components.

The next lemma relates the empty area outside a boundary cycle of component with its length, and will play a key role in proving the main results.

Lemma 3 (Geometric Lemma). *Let* $\mathcal{C}$ *be a component in* T_N *with* β *one of its maximal boundary walks, and* $l = length(\beta)$ *its length. Assume that* $\mathcal{C}$ *has at least two occupied sites. Then the size of the empty area* A_β *outside* β *is bounded below by* $|A_\beta| \geq dl/R$, *for some big enough constant* R. *Moreover, if* $\mathcal{C}$ *is rectangular, and* β *is a maximal boundary walk with respect to the outside region, we have* $|A_\beta| \geq h + dl/R$.

Lemma 4. *Let* $\mathcal{C}$ *be an nr-component which can coexist with other nr-components. Then it has a boundary cycle* β *with length*$(\beta) \geq n - o(n)$.

This lemma's proof (omitted) is effected by quantifying the intuitive idea that such a component must "wrap around" the torus.

The next technical result shows that simple components are predominant a.a.s. in T_N. The proof uses the Geometric Lemma.

Lemma 5. *If* $h\varrho \to \infty$, *then* $\mathbf{E}[Y] = o(\mathbf{E}[X])$ *and* $\mathbf{E}[Z_2] = o(\mathbf{E}[X])$.

The following theorem gives the connectivity of $G_f[W]$ in the static case, under various assumptions. The proof follows from Lemma 5, Theorem 6 and Chebyshev inequality

Theorem 7. • *For* $\mu \to \infty$, $G_f[W]$ *is disconnected a.a.s.*
• *For* $\mu = \Theta(1)$, *then* $K = 1 + X$ *a.a.s., and* X *is asymptotically Poisson.*
• *For* $\mu \to 0$, $G_f[W]$ *is connected a.a.s.*

From the previous theorem we immediately obtain that the probability that $G_f[W]$ is connected is $e^{-\mu} + o(1)$. Since $G_f[W]$ is a.a.s. disconnected if $\mu \to \infty$, and a.a.s. connected if $\mu \to 0$, we may restrict attention to $\mu = \Theta(1)$. In this case, we only have a.a.s. simple components and the giant one found in the above proof.

Dynamic Properties. With the static results under our belt, the analysis of the dynamic case is quite similar to that of the cycle (though differing in details and some of the justifications) so we just state the major results.

By analogy with d-hole lines, we define a *simple component line* to be a maximal sequence of pairs $(v_1, t_1), \ldots, (v_l, t_l)$ where v_i is a simple component

existing at time t_i for $1 \leq i \leq l$, and such that $t_i = t_{i-1} + 1$ and the vertex v_i is adjacent to v_{i-1}, for $2 \leq i \leq l$. Birth, death, survival and random variables S, B, D are the defined analogously to the cycle case.

Theorem 8. *For t in any fixed bounded time interval, the random variables $S(t)$, $B(t)$ and $D(t)$ are asymptotically jointly independent Poisson, with the expectations*

$$\mathbf{E}[S(t)] \sim \begin{cases} \mu & \text{if } d\varrho \to 0, \\ \mu - \lambda & \text{if } d\varrho \to c, \\ 4\frac{1-e^{-\varrho/4}}{1-e^{-\varrho}} e^{-(2d+5/4)\varrho}\mu & \text{if } d\varrho \to \infty, \end{cases} \quad \text{and}$$

$$\mathbf{E}[B(t)] = \mathbf{E}[D(t)] \sim \begin{cases} 2d\varrho\mu & \text{if } d\varrho \to 0, \\ \lambda & \text{if } d\varrho \to c, \\ \mu & \text{if } d\varrho \to \infty, \end{cases}$$

where $\lambda = \left(1 - e^{-2d\varrho}\right)\mu$. Here $0 < \lambda < \mu$ for $d\varrho \to c$.

Theorem 9. *Let $\lambda = \lambda(\varrho) = \mu\left(1 - e^{-2d\varrho}\right)$. The probability that $G_{f_t}[W]$ is connected and that $G_{f_{t+1}}[W]$ is disconnected is asymptotic to $2\mu e^{-\mu} d\varrho$ if $d\varrho \to 0$, to $e^{-\mu}(1 - e^{-\lambda})$ if $d\varrho \to c$, and to $e^{-\mu}(1 - e^{-\mu})$ if $d\varrho \to \infty$.*

A sequence of results similar to the case of the cycle yields the following analogue of Theorem 3.

Theorem 10. *The expected life of a simple component line is asymptotic to $\frac{1}{2d\varrho}$ if $\varrho \to 0$, to $\frac{\mu}{\lambda}$ if $\varrho \to c$, and to 1 if $\varrho \to \infty$. where λ is defined as in Theorem 9.*

As in the case of C_N, let T_C be a random variable measuring the time $G_{f_t}[W]$ remains connected from a moment at which it is so, and let T_D be a random variable measuring the time $G_{f_t}[W]$ remains disconnected, from the moment at which it is so. The next theorem gives the expected time that the graph of walkers $G_{f_t}[W]$ will remain connected or disconnected.

Theorem 11. *Let λ be defined as in Theorem 9. Then,*

$$\mathbf{E}[T_C] \sim \begin{cases} \frac{1}{2\mu d\varrho} & \text{if } \varrho \to 0, \\ \frac{1}{1-e^{-\lambda}} & \text{if } \varrho \to c, \\ \frac{1}{1-e^{-\mu}} & \text{if } \varrho \to \infty, \end{cases} \quad \text{and } \mathbf{E}[T_D] \sim \begin{cases} \frac{e^{\mu}-1}{2\mu d\varrho} & \text{if } \varrho \to 0, \\ \frac{e^{\mu}-1}{1-e^{-\lambda}} & \text{if } \varrho \to c, \\ e^{\mu} & \text{if } \varrho \to \infty, \end{cases}$$

5 Conclusions and Open Problems

In this work we have characterised the dynamic connectivity of a very large set of agents which **move** through a prescribed real or virtual graph. We believe it is the first time that this kind of characterisation has been obtained, and could

open an interesting line of research. We gave characterisations for the cycle and the grid for the ℓ_1 norm, which can be extended without problems to the ℓ_2 norm, as applies for instance to robots with movement restricted to orthogonal N-E-S-W directions but with omni-directional radio-frequency communication.

Currently under way is the extension of the results presented here to the hypercube, which is interesting from the mathematical point of view, as the number of neighbours is not constant. Moreover, from the point of view of ad-hoc networks, an interesting case is the random geometric graph (with the ℓ_2 norm), where the walkers can move randomly in any direction taking a step of some random size. We think the present work constitutes a step in this direction.

References

1. P.K. Agarwal, L. Guibas, H. Edelsbrunner, J. Erickson, M. Isard, S. Har-Peled, J. Hershberger, C. Jensen, L. Kavraki, P. Koehl, Ming Lin, D. Manocha, D. Metaxas, B. Mirtich and D. Mount, Algorithmic issues in modeling motion. *ACM Computing Surveys*, 34:550–572, 2002.
2. A.D. Barbour, L. Holst and S. Janson, *Poisson Approximation*. Clarendon, Oxford, 1992.
3. B. Bollobás, *Random Graphs, 2nd edn.* Academic Press, London, 2002.
4. Z. Gao and N.C. Wormald, Asymptotic normality determined by high moments, and submap counts of random maps. *Probability Theory and Related Fields* (to appear).
5. L. Guibas, J. Hershberger, S. Suri and Li Zhang, Kinetic Connectivity for Unit Disks. *Discrete and Computational Geometry*, 25:591–610, 2001.
6. K. H. Kramer, N. Minar and P. Maes, Tutorial: Mobile Software Agents for Dynamic Routing. *Mobile Computing and Communications Review*, 3:12–16, 1999.
7. M. Penrose, On k-connectivity for random geometric graphs *Random Structures and Algorithms*, 15:145–164, 1999.
8. M. Penrose, *Random Geometric Graphs*. Oxford University Press, 2003.
9. P. Santi and D.M. Blough, An Evaluation of Connectivity in Mobile Wireless Ad Hoc Networks. *Proceedings of IEEE Conference on Dependable Systems and Networks*. Washington DC, 89–98, June 2002.

Improved Algorithms for Dynamic Page Migration*

Marcin Bienkowski[1], Miroslaw Dynia[2], and Miroslaw Korzeniowski[1]

[1] International Graduate School of Dynamic Intelligent Systems,
University of Paderborn, Germany
{young, rudy}@upb.de
[2] DFG Graduate College "Automatic Configuration in Open Systems",
University of Paderborn, Germany
mdynia@upb.de

Abstract The dynamic page migration problem [4] is defined in a distributed network of n mobile nodes sharing one indivisible memory page of size D. During runtime, the nodes can both access a unit of data from the page and move with a constant speed, thus changing the costs of communication. The problem is to compute *online* a schedule of page movements to minimize the total communication cost.

In this paper we construct and analyze the first deterministic algorithm for this problem. We prove that it achieves an (up to a constant factor) optimal competitive ratio $\mathcal{O}(n \cdot \sqrt{D})$. We show that the randomization of this algorithm improves this ratio to $\mathcal{O}(\sqrt{D} \cdot \log n)$ (against an oblivious adversary). This substantially improves an $\mathcal{O}(n \cdot \sqrt{D})$ upper bound from [4]. We also give an almost matching lower bound of $\Omega(\sqrt{D} \cdot \sqrt{\log n})$ for this problem.

1 Introduction

The page migration problem [1, 2, 5, 7, 10] arises in a distributed network of processors which share some global data. Shared variables or memory pages are stored at the local memory of these processors. If a processor wants to access (read or write) a single unit of data from a page, and the page is not stored in its local memory, it has to send a request to the processor holding the page, and appropriate data is sent back. Such transactions incur a cost which is proportional to the distance between these two processors. To avoid the problem of maintaining consistency among multiple copies of the page, the model allows only one copy of the page to be stored in the network. However, to reduce the

* Full version of this paper is available under http://wwwhni.upb.de/publikationen/. Partially supported by DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen Entwurfsmethoden Anwendungen" and by the Future and Emerging Technologies programme of the EU under EU Contract 001907 DELIS "Dynamically Evolving, Large Scale Information Systems".

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 365–376, 2005.

communication cost, the system can migrate the page between processors. The migration cost is proportional to the cost of sending one unit of data times the size of the memory page. The problem is to decide, online, when and where to move the page to minimize the total cost of communication over all sequences of requests. The performance of the online algorithm is measured by competitive analysis [9, 6], i.e. by comparing its total cost to the total cost of the optimal offline algorithm on the same input sequence.

The *dynamic page migration* (DPM) problem introduced by Bienkowski, Korzeniowski and Meyer auf der Heide [4] is an extension to this model where the adversary can change the positions of the nodes during the runtime of the algorithm. This is typical in mobile networks, and also models the dynamics of networks that are not exclusively dedicated to the page migration problem. Obviously, the movement changes the distances and corresponding costs of sending data between the processors. The DPM model imposes a *basic restriction* on the adversary, i.e. the speed of the network changes has to be bounded. In each round a new position of each node has to be chosen within a ball of a constant diameter centered at its previous position. Whereas in the standard models of page migration [5, 10] the cost is equal to the distance between two processors, in DPM the cost of accessing one unit of data is defined as the distance between the requesting node and the node holding the page plus a constant overhead for communication. The reason for this overhead is twofold. First, this overhead represents the fact that there are no zero-cost communications, even if nodes are very close. Second, without the overhead in the definition of cost, the problem becomes infeasible, i.e. no algorithm could achieve a finite competitive ratio.

The Model. Following [4], we define the DPM problem as follows. The network is modelled as a set of n nodes (processors) labelled $v_0, v_1, \ldots, v_{n-1}$ placed in a metric space $(\mathcal{X}, d)$. The distances between the nodes are given by the metric d, but we extend the notion of the distance between two nodes in the following way. If v_i and v_j are the same node, which we denote by $v_i \equiv v_j$, then we denote the distance between them by 0_{E}. Note that this is different from the case where they just occupy the same point in the space, in which case we write $v_i = v_j$ and $d(v_i, v_j) = 0$.

We assume discrete time steps $t = 1, 2, \ldots$. We denote the distance between two nodes v_x and v_y in time step t as $d_t(v_x, v_y)$. Since the nodes can move, this distance can change with time. A tuple C_t describing the positions of all nodes in a time step t is called a *configuration at time* t.

An input consists of a *configuration sequence* (C_t) and a *request sequence* (σ_t), both created by an adversary, where σ_t denotes the node that issues a request at time t. The basic restriction mentioned above is formalized as follows.

Definition 1. *A Δ-restricted adversary is allowed to choose a new position of each node within a ball of radius Δ, centered at the previous position of this node.*[1]

[1] All adversaries considered in the previous work [4] were 1-restricted.

For the Δ-restricted adversary and any node v_x, its positions x_t and x_{t+1} in two consecutive configurations C_t and C_{t+1} cannot be too far apart, i.e. $d(x_t, x_{t+1}) \leq \Delta$. Furthermore, by λ we denote a maximum distance allowed between any two nodes at any time step. If we do not impose any such restriction, then $\lambda = \infty$.

Any two nodes are able to communicate directly with each other. The cost of sending a unit of data from node v_x to node v_y at time step t is defined by a cost function $c_t(v_x, v_y)$ as follows. If $v_x \equiv v_y$ then $c_t(v_x, v_y) = 0$. Otherwise $c_t(v_x, v_y) = d_t(v_x, v_y) + 1$. We have one shared, indivisible memory page of size D, initially stored at the node v_0. The cost of moving the whole page from v_x to v_y in time step t is equal to $D \cdot c_t(v_x, v_y)$.

In time step $t \geq 1$, first the positions of the nodes are defined by C_t and then a request is issued at the node v_{σ_t}. For clarity, we abuse the notation and write "*node* σ_t" instead of "*node* v_{σ_t}". In this time step the algorithm has its page in some node denoted by $P_{\mathrm{ALG}}(t)$. First, it has to pay $c_t(P_{\mathrm{ALG}}(t), \sigma_t)$ for serving the request. Then it can optionally move the page to a new position $P'_{\mathrm{ALG}}(t)$ paying the cost $D \cdot c_t(P_{\mathrm{ALG}}(t), P'_{\mathrm{ALG}}(t))$. Sometimes, we will abuse the notation by writing that an algorithm is at v_i or moves to v_j, meaning that the algorithm's page is at v_i or the algorithm moves its page to v_j.

We consider only online algorithms, i.e. the ones which make decision in step t solely on the basis of the initial part of the input up to step t, i.e. on the sequence $C_1, \sigma_1, C_2, \sigma_2, \ldots, C_t, \sigma_t$. To analyze the performance of online algorithm ALG we use competitive analysis [9, 6]. We say that a deterministic algorithm ALG is c-competitive if for all input sequences (σ_t, C_t) it holds $C_{\mathrm{ALG}}((\sigma_t, C_t)) \leq c \cdot C_{\mathrm{OPT}}((\sigma_t, C_t))$.[2] $C_{\mathrm{ALG}}((\sigma_t, C_t))$ and $C_{\mathrm{OPT}}((\sigma_t, C_t))$ are the cost of ALG and the *optimal offline* algorithm, respectively, run on the input sequence (σ_t, C_t). The factor c is called the *competitive ratio* of the algorithm. For a randomized algorithm to be c-competitive, we require that its *expected* cost is not greater than c times the cost of the optimum. The expected value is taken over all possible random choices of the algorithm. In this paper we consider only oblivious adversaries [3], which have no access to the random bits used by the algorithm.

Related Work. For the page migration problem in a static network, Westbrook [10] gave the first randomized $\mathcal{O}(1)$-competitive algorithms against oblivious and adaptive adversaries. This result was improved for some network topologies like trees or uniform graphs by Chrobak et al. [7]. The first constant competitive deterministic algorithm was Move-To-Min given by Awerbuch, Bartal and Fiat in [1] and the competitive ratio was subsequently improved by Bartal, Charikar and Indyk [2].

Bienkowski, Korzeniowski and Meyer auf der Heide [4] defined the DPM model. Their randomized algorithm $\mathrm{ALG}_{\mathrm{DIST}}$ achieved a competitive ratio of $\mathcal{O}(\min\{n \cdot \sqrt{D}, D, \lambda\})$ against an adaptive-online adversary. This ratio was proven to be up to a constant factor optimal. They also gave two trivial deterministic algorithms achieving competitive ratios of $\mathcal{O}(D)$ and $\mathcal{O}(\lambda)$, respectively. Finally,

[2] In literature, this notion is sometimes referred to as *strict competitiveness*.

they proved that the competitive ratio of any randomized algorithm against an oblivious adversary is at least $\Omega(\min\{\sqrt{D}, \lambda\})$.

The deterministic and randomized algorithms presented in this paper use the marking technique, which bears a resemblance to the Least Recently Used (LRU) paging algorithm by Sleator and Tarjan [9] and randomized MARK paging algorithm by Fiat et al. [8], respectively.

Contribution of the Paper. In this paper we improve and extend the results of [4]. In Sect. 3 we give a first deterministic algorithm, MARK, for the DPM problem. Our algorithm achieves the competitive ratio of $\mathcal{O}(n \cdot \sqrt{D})$. It can be combined with two trivial algorithms from [4], yielding an $\mathcal{O}(\min\{n \cdot \sqrt{D}, D, \lambda\})$-competitive algorithm. The constants hidden in $\mathcal{O}$ notation are moderate; in fact they are much better that the constants from the proof of competitiveness of $\mathrm{ALG}_{\mathrm{DIST}}$ in [4].

In Sect. 4 we randomize MARK and get an algorithm R-MARK which is $\mathcal{O}(\sqrt{D} \cdot \log n)$-competitive against an oblivious adversary. Again, it can be combined with $\mathcal{O}(D)$ and $\mathcal{O}(\lambda)$-competitive algorithms, resulting in an $\mathcal{O}(\min\{\sqrt{D} \cdot \log n, D, \lambda\})$-competitive algorithm. In Sect. 5 we prove that this ratio is almost optimal by showing that the lower bound on competitiveness of any randomized algorithm against an oblivious adversary is $\Omega(\min\{\sqrt{D \cdot \log n}, D^{2/3}, \lambda\})$.

2 Preliminaries

In this paper we consider only $\frac{1}{2}$-restricted adversaries. This assures that the distance between any two nodes can change only by 1 per time step. The presented proofs can be extended to any constant-restricted adversary with the Reduction Lemma proven in the full version of the paper.

Lemma 1 (Reduction Lemma). *Assume that there exists a k-competitive (possibly randomized) algorithm ALG_A against an A-restricted adversary. Then ALG_A is k-competitive against a B-restricted adversary for $B \leq A$. Additionally, for any $B \geq A$ there exists a (randomized) algorithm ALG_B which is $\frac{B}{A} \cdot k$-competitive against a B-restricted adversary.*

Throughout this paper we will use the following notation. OPT denotes the optimal offline algorithm. If ALG is any algorithm and $\mathcal{S}$ any sequence of consecutive time steps, then by $C_{\mathrm{ALG}}(\mathcal{S})$ we denote the cost of the algorithm ALG on $\mathcal{S}$. In particular, $C_{\mathrm{OPT}}(\mathcal{S})$ is the cost of the optimal algorithm on sequence $\mathcal{S}$. By $C^{\mathrm{req}}_{\mathrm{ALG}}(\mathcal{S})$ we denote the cost of serving requests by ALG, i.e. not counting the cost of page movements in $\mathcal{S}$.

In the proof of competitiveness of any algorithm ALG we will follow the following scheme. We take any input sequence (C_t, σ_t), and we run ALG and OPT "in parallel" on this sequence. By $P_{\mathrm{ALG}}(t)$ and $P_{\mathrm{OPT}}(t)$ we denote the node holding ALG's and OPT's pages, respectively, in the t-th time step. At each step we will be able to compare the cost paid so far, respectively by ALG and by OPT.

Jump Sets. Before we construct and analyze our algorithms for DPM problem, we prove a useful property of the costs incurred during the runtime. We assume that $\sqrt{D}$ is an integer, and we consider any sequence I of $K = \sqrt{D}$ steps, numbered from 1 to K. We call sequences of such lengths *intervals*. Let σ_i be the node which issues a request in the i-th step of I and $d_i(\cdot)$, $c_i(\cdot)$ be the distance and cost functions in the i-th step.

Definition 2. *A gravity center for I is a vertex v which minimizes the sum $\sum_{i=1}^{K} c_K(v, \sigma_i)$. If there is more than one such vertex, then the gravity center is the one labelled with the smallest index.*[3]

Definition 3. *Let I be any interval (of length K) and let $v_{\min}$ be its gravity center. Then a jump set $\mathcal{G}(I)$ consists of all the nodes whose K-th step distance to $v_{\min}$ is not greater than $9 \cdot \sqrt{D}$, i.e. $\mathcal{G}(I) := \{v \in V : d_K(v, v_{\min}) \leq 9 \cdot \sqrt{D}\}$.*

Intuitively, the gravity center and the jump set try to approximate a "good" position to place a page in the last I steps. This intuition is formalized in the following lemma, whose proof can be found in the full version of the paper.

Lemma 2. *Consider any interval I of K requests and corresponding jump set $\mathcal{G}(I)$ with gravity center at $v_{\min}$. Consider any algorithm* ALG *which at the end of the K-th step is in a node $P_{\text{ALG}} \notin \mathcal{G}(I)$. Then $C_{\text{ALG}}(I) \geq D/4$.*

3 A Deterministic Algorithm

In this section we construct and analyze a deterministic algorithm MARK. MARK divides the requests into intervals of length K. In each interval MARK remains at one node and serves all the requests issued. Then, at the end of the interval, it makes its decision, whether and where to move the page.

Intervals are grouped in epochs; the first epoch starting with the beginning of the input sequence. MARK begins each epoch without any nodes marked. Subsequently, for each node v we measure the cost incurred by requests issued so far in the current epoch $\mathcal{E}$, on the algorithm which remains in v and never moves. We denote this amount by $C_v^{\text{req}}(\mathcal{E})$ and when it reaches $D/4$, then node v becomes marked.

MARK waits in one node, denoted P_{MARK} for one or more intervals, and moves at the end of an interval in which P_{MARK} becomes marked. When MARK wants to move, it computes a jump set $\mathcal{G}$ on the basis of the last interval and moves to v^*, an arbitrarily chosen node from the not yet marked nodes of $\mathcal{G}$. If there is no unmarked node in $\mathcal{G}$, then prior to choosing v^* the algorithm erases the marks on all the nodes and a new epoch begins.

The pseudocode of MARK is presented in Fig. 1. The algorithm is initialized by setting $\mathcal{E} = \emptyset$ and *unmarking* all the nodes.

Theorem 1. MARK *achieves competitive ratio of $\mathcal{O}(n \cdot \sqrt{D})$ for DPM problem.*

[3] We could apply any tie breaking method here.

```
Serve requests in interval I
E := E ∪ I
for each v ∈ V
    if C_v^req(E) ≥ D/4 then mark v
if P_MARK is marked then
    G := G(I)
    if all nodes in G are marked then
        unmark all nodes from V
        E = ∅
    Choose any not marked node v* from G
    Move the page to v*
```

Fig. 1. Algorithm MARK for one interval I

Proof. To make the proof concise, we introduce a notion of a *phase*. A phase is a sequence of consecutive intervals in which MARK remains at one node. In other words MARK's page movement divides each epoch into a sequence of phases. Lemma 2 implies that after any phase $\mathcal{P} = (I_1, \ldots, I_p)$ for all the nodes v outside $\mathcal{G}(I_p)$ holds $C_v^{\mathrm{req}}(\mathcal{P}) \geq C_v^{\mathrm{req}}(I_p) \geq D/4$. Thus, these nodes are already marked, and, in fact, at the end of $\mathcal{P}$ the algorithm chooses an arbitrary node v^* from not yet marked nodes.

As an immediate consequence we get the following two properties. First, each epoch ends exactly when all the nodes are marked. Therefore, although the division into phases depends on the choices of the algorithm, the division into epochs depends only on the input sequence. Second, since each epoch begins with no node marked, and in each phase at least one node becomes marked, each epoch consists of at most n phases.

For the proof we use amortized analysis and define a potential function Φ_t in time step t. $\Phi_t = 2 \cdot D \cdot L_t$, where L_t is the t-th step distance between P_{MARK} and P_{OPT}.

If $\mathcal{S}$ is some sequence of ℓ steps, numbered from 1 to ℓ, then by Φ_0 we denote the potential just before this sequence and we define $\Delta\Phi(\mathcal{S})$ as the difference between the potential after and before phase $\mathcal{S}$, i.e. $\Delta\Phi(\mathcal{S}) := \Phi_\ell - \Phi_0$. Since in the beginning of the runtime $\Phi_0 = 0$ and Φ_t is non negative in every time step t, it is sufficient to prove that for any input sequence $\mathcal{S}$, it holds $C_{\mathrm{MARK}}(\mathcal{S}) + \Delta\Phi(\mathcal{S}) \leq \mathcal{O}(n \cdot \sqrt{D}) \cdot C_{\mathrm{OPT}}(\mathcal{S})$.

First, we prove a bound on the cost of MARK in one phase.

Lemma 3. *In each phase $\mathcal{P}$ of the algorithm* MARK*, the amortized cost of the algorithm $C_{\mathrm{MARK}}(\mathcal{P}) + \Delta\Phi(\mathcal{P})$ is not greater than $\mathcal{O}(\sqrt{D}) \cdot C_{\mathrm{OPT}}(\mathcal{P}) + \mathcal{O}(D \cdot \sqrt{D})$.*

Proof. In this proof we often use the triangle inequality, the inequality $c_t(x, y) \leq 1 + d_t(x, y)$, and the fact that the adversary is 1/2-restricted and therefore the distance between two nodes can change by at most K in one interval.

We assume that $\mathcal{P}$ consists of p intervals $I_1, I_2, \ldots, I_p$ and that the algorithm has its page in P_{MARK} throughout the whole phase $\mathcal{P}$. First, we bound the amor-

tized cost in the intervals $(I_1, I_2, \ldots, I_{p-1})$. The algorithm does not move within these intervals, thus the total cost of serving requests is at most $D/4$, because otherwise P_{MARK} would become marked and the phase would be finished earlier. Therefore, $C_{\text{MARK}}(I_1, \ldots, I_{p-1}) = C^{\text{req}}_{\text{MARK}}(I_1, \ldots, I_{p-1}) < D/4$. For bounding the change of the potential in these intervals we use the following lemma, which is proven in the full version of the paper.

Lemma 4. *If the algorithm* MARK *remains in the same node for a sequence of requests* $\mathcal{S}$ *and* $C^{\text{req}}_{\text{ALG}}(\mathcal{S}) \leq D$*, then* $\Delta\Phi(\mathcal{S}) \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) + \mathcal{O}(D \cdot \sqrt{D})$

In consequence, $\Delta\Phi(I_1, \ldots, I_{p-1}) \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(I_1, \ldots, I_{p-1}) + \mathcal{O}(D \cdot \sqrt{D})$ and it is sufficient to bound the amortized cost in the interval I_p.

In the remaining part of the proof of Lemma 3 we use some ideas from the the proof of 7-competitiveness of Move-To-Min algorithm [1]. However, we have to take into account the movement of the nodes and shorter interval lengths.

We number the time steps within I_p from 1 to K. Let v^* be the new node chosen after phase $\mathcal{P}$. We have $c_K(P_{\text{MARK}}, v^*) \leq 1 + d_K(P_{\text{MARK}}, v_{\min}) + d_K(v_{\min}, v^*)$. Since $v^* \in \mathcal{G}(I_p)$, $d_K(v_{\min}, v^*) \leq 9\sqrt{D}$, and therefore $C_{\text{MARK}}(I_p) = C^{\text{req}}_{\text{MARK}}(I_p) + D \cdot c_K(P_{\text{MARK}}, v^*) \leq C^{\text{req}}_{\text{MARK}}(I_p) + D \cdot d_K(P_{\text{MARK}}, v_{\min}) + \mathcal{O}(D \cdot \sqrt{D})$.

By a_{i-1} and a_i we denote the position of OPT, respectively at the beginning and at the end of the i-th step. Therefore, the optimal algorithm's cost in interval I_p is equal to $C_{\text{OPT}}(I_p) = \sum_{i=1}^{K} (c_i(a_{i-1}, \sigma_i) + D \cdot c_i(a_{i-1}, a_i))$ We need two technical lemmas which are proven in the full version of the paper.

Lemma 5. *Let* $X_t = \sum_{i=1}^{K} d_i(a_t, \sigma_i)$ *be the cost of serving all the requests from node* a_t*. Then for all* $0 \leq t \leq K$ *it holds* $X_t \leq C_{\text{OPT}}(I_p) + D$*.*

Lemma 6. *Let* $Y_t = D \cdot d_K(a_t, v_{\min})$*. Then for all* $0 \leq t \leq K$ *it holds* $Y_t \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(I_p) + \mathcal{O}(D \cdot \sqrt{D})$*.*

We need to bound the amortized cost of MARK in the interval I_p; this cost is equal to $C^{\text{req}}_{\text{MARK}}(I_p) + D \cdot d_K(P_{\text{MARK}}, v_{\min}) + \mathcal{O}(D \cdot \sqrt{D}) + \Phi_K - \Phi_0$. The summands can be bounded as described below. The same bounds can be applied to any sequence of at most K steps.

$$\begin{aligned} C^{\text{req}}_{\text{MARK}}(I_p) &= \sum_{i=1}^{K} c_i(P_{\text{MARK}}, \sigma_i) \leq \sum_{i=1}^{K} (1 + d_i(P_{\text{MARK}}, a_0) + d_i(a_0, \sigma_i)) \\ &\leq K + \sum_{i=1}^{K} (K + d_0(P_{\text{ALG}}, a_0)) + X_0 \\ &\leq \Phi_0/2 + C_{\text{OPT}}(I_p) + \mathcal{O}(D) \end{aligned} \tag{1}$$

$$\begin{aligned} &\Phi_K + D \cdot d_K(P_{\text{MARK}}, v_{\min}) \\ &\quad \leq 2 \cdot D \cdot d_K(v^*, a_K) + D \cdot d_K(P_{\text{MARK}}, a_0) + D \cdot d_K(a_0, v_{\min}) \\ &\quad \leq 2 \cdot Y_K + \mathcal{O}(D \cdot \sqrt{D}) + D \cdot (d_0(P_{\text{MARK}}, a_0) + K) + Y_0 \\ &\quad \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(I_p) + \Phi_0/2 + \mathcal{O}(D \cdot \sqrt{D}) \end{aligned} \tag{2}$$

Thus, the amortized cost in the interval I_p is bounded by $\mathcal{O}(\sqrt{D}) \cdot C_{\mathrm{OPT}}(I_p) + \mathcal{O}(D \cdot \sqrt{D})$, which finishes the proof of Lemma 3. □

Closely observing the proof of Lemma 3, we can show that the cost of serving requests in a phase which is not yet finished, can be either paid from the potential or is amortized against the cost of the optimal algorithm in this phase.

Lemma 7. *Let $\mathcal{P}'$ be the first ℓ steps of phase $\mathcal{P}$ of algorithm* MARK*. Then $C_{\mathrm{MARK}}(\mathcal{P}') \leq \Phi_B + \mathcal{O}(\sqrt{D}) \cdot C_{\mathrm{OPT}}(\mathcal{P}') + \mathcal{O}(D \cdot \sqrt{D})$, where Φ_B is the potential at the beginning of the phase $\mathcal{P}$.*

Proof. We divide the phase into p intervals, i.e. $\mathcal{P}' = (I_1, I_2, \ldots, I_p)$, where the last interval possibly has less than K steps. Let Φ_0 be the potential at the beginning of I_p. From the proof of Lemma 3 follows that $C_{\mathrm{MARK}}(I_1, \ldots, I_{p-1}) + \Phi_0 \leq \Phi_B + \mathcal{O}(\sqrt{D}) \cdot C_{\mathrm{OPT}}(I_1, \ldots, I_{p-1}) + \mathcal{O}(D \cdot \sqrt{D})$. Therefore, it is sufficient to show that $C_{\mathrm{MARK}}(I_p) \leq \Phi_0 + C_{\mathrm{OPT}}(I_p) + \mathcal{O}(D \cdot \sqrt{D})$. This follows from (1) in the proof of Lemma 3. □

As shown in Lemmas 3 and 7 we would be able to prove good bounds for each phase if we could neglect the additive term of $\mathcal{O}(D \cdot \sqrt{D})$. Thus, we have to find some additional lower bounds on the cost of the optimal algorithm. We can do this by using a property of our marking scheme.

Lemma 8. *The cost of* OPT *in any finished $\mathcal{E}$ is at least $\Omega(D)$. Moreover,* OPT*'s cost in the first epoch $\mathcal{E}_1$ (even if it is unfinished) is either $C_{\mathrm{MARK}}(\mathcal{E}_1)$ or at least $\Omega(D)$.*

Proof. First, we consider a finished epoch $\mathcal{E}$. If OPT moves its page within $\mathcal{E}$, then its cost is at least D. Otherwise, OPT remains at one node P_{OPT}, which becomes marked at some point of the epoch. Therefore, it must have paid a cost of at least $C^{\mathrm{req}}_{P_{\mathrm{OPT}}} \geq D/4$ in this epoch.

Second, we consider $\mathcal{E}_1$. If OPT moves within $\mathcal{E}_1$, then it pays at least D. Otherwise, it remains in v_0 for the whole $\mathcal{E}_1$ and we have two cases. If $\mathcal{E}_1$ consists of one unfinished phase only, then $C_{\mathrm{OPT}}(\mathcal{E}_1) = C^{\mathrm{req}}_{v_0}(\mathcal{E}_1) = C_{\mathrm{MARK}}(\mathcal{E}_1)$. Otherwise, let $\mathcal{P}_1$ be the first (finished) phase of $\mathcal{E}_1$. Since v_0 becomes marked at the end of $\mathcal{P}_1$, $C_{\mathrm{OPT}}(\mathcal{E}_1) \geq C_{\mathrm{OPT}}(\mathcal{P}_1) \geq C^{\mathrm{req}}_{v_0} \geq D/4$.

Therefore, for any sequence $\mathcal{S}$ consisting of k epochs, where the last epoch is possibly unfinished, either $C_{\mathrm{OPT}}(\mathcal{S}) = C_{\mathrm{MARK}}(\mathcal{S})$ or $C_{\mathrm{OPT}}(\mathcal{S}) \geq \Omega(k \cdot D)$. In the latter case, we combine the result with Lemmas 3 and 7 obtaining

$$\begin{aligned} C_{\mathrm{MARK}}(\mathcal{S}) &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\mathrm{OPT}}(\mathcal{S}) + k \cdot n \cdot \mathcal{O}(D \cdot \sqrt{D}) \\ &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\mathrm{OPT}}(\mathcal{S}) + \mathcal{O}(n \cdot \sqrt{D}) \cdot C_{\mathrm{OPT}}(\mathcal{S}) \ , \end{aligned}$$

which finishes the proof of Thm. 1. □

In [4] two simple deterministic algorithms attaining competitive ratios of $\mathcal{O}(D)$ and $\mathcal{O}(\lambda)$, respectively, were presented. From the analysis of their potential functions follows that it is possible to combine them with MARK in one

algorithm which achieves the competitive ratio of $\mathcal{O}(\min\{n \cdot \sqrt{D}, D, \lambda\})$. We omit the details here.

This result is up to a constant factor optimal, since the $\Omega(\min\{n \cdot \sqrt{D}, D, \lambda\})$ lower bound for a randomized algorithm against an adaptive-online adversary presented in [4] applies also to the deterministic case (see [3] for comparison of the power of different adversaries).

4 Randomization Against an Oblivious Adversary

In this section we show a direct randomization of MARK which yields an algorithm R-MARK. In the formulation of the algorithm, instead of choosing v^* in deterministic fashion from not yet marked nodes of $\mathcal{G}$ (see Fig. 1), we choose v^* uniformly at random from not yet marked nodes of $\mathcal{G}$.

Theorem 2. R-MARK *is* $\mathcal{O}(\sqrt{D} \cdot \log n)$*-competitive against an oblivious adversary.*

Proof. For the analysis we use the same potential function Φ as for MARK. Since Lemmas 3 and 7 hold for *any* choice of v^* from not yet marked nodes of $\mathcal{G}$, they hold also in the case when v^* is chosen randomly. Therefore, for any finished phase $\mathcal{P}$ it holds $C_{\text{R-MARK}}(\mathcal{P}) + \Delta\Phi(\mathcal{P}) \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{P}) + \mathcal{O}(D \cdot \sqrt{D})$. Similar result holds for an unfinished phase (see Lemma 7). However, for the algorithm R-MARK we can have a bound better than n (at least on expectation) for the number of phases in an epoch.

Lemma 9. *For any epoch* $\mathcal{E}$*, the expected number of phases in this epoch is* $\mathcal{O}(\log n)$*. The expectation is taken over all possible random choices of* R-MARK.

Proof. Analogously to the proof of Thm. 1 we can forget about the set $\mathcal{G}$ and assume that R-MARK chooses v^* uniformly at random from all not yet marked nodes.

Let $\{b_i\}_{i=1}^{n}$ be the nodes in the order they are becoming marked in epoch $\mathcal{E}$. Assume that in a phase $\mathcal{P}$ the algorithm is in a node b_k. Then at the end of phase $\mathcal{P}$ it chooses a new node v^* uniformly at random from $n - k$ nodes i.e. from the set $B_k^+ := \{b_i : k + 1 \leq i \leq n\}$. Actually, it might happen that some of nodes from B_k^+ were also marked at the end of phase $\mathcal{P}$, in which case the algorithm has even fewer than $n - k$ nodes to choose randomly from.

At the beginning of each epoch, except from the first one, the algorithm also chooses a place uniformly at random from the whole set V. Let T_i be the expected number of phases, provided that i nodes are still unmarked. We have a recursive formula

$$T_0 \;=\; 0, \quad T_k \;=\; 1 + \sum_{i=0}^{k-1} \frac{1}{k} \cdot T_i \;\;,$$

where the second equality follows from the linearity of expected value. It can be proven by induction that $T_i = H_i$, the i-th harmonic number. In fact, the

expected number of phases can be even smaller if some nodes get marked concurrently and R-MARK has fewer nodes to choose from. T_n is the bound on the expected number of phases in each epoch except the first one, because the very first node is not chosen randomly. However, this incurs at most one additional phase after which we have our random process. Therefore, the expected number of phases in any epoch is at most $T_n + 1 = H_n + 1 = \mathcal{O}(\log n)$. □

Since Lemma 8 does not depend on the way of choosing v^*, either, as long it is chosen from the not yet marked nodes, we have that for any sequence $\mathcal{S}$ consisting of k epochs we have

$$\begin{aligned}\mathbf{E}[C_{\text{R-MARK}}(\mathcal{S})] &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) + k \cdot \mathcal{O}(\log n) \cdot \mathcal{O}(D \cdot \sqrt{D}) \\ &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) + \mathcal{O}(\log n \cdot \sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) \ .\end{aligned}$$

This finishes the proof of Thm. 2. □

It is also possible to combine R-MARK with $\mathcal{O}(D)$ and $\mathcal{O}(\lambda)$-competitive algorithms obtaining an algorithm which is $\mathcal{O}(\min\{\sqrt{D}\cdot\log n, D, \lambda\})$-competitive against an oblivious adversary. We omit the details.

5 Lower Bound Against an Oblivious Adversary

In this section we show that no online algorithm can achieve a competitive ratio better than $\Omega(\min\{\sqrt{D \cdot \log n}, D^{2/3}, \lambda\})$. This improves an $\Omega(\min\{\sqrt{D}, \lambda\})$ lower bound result from [4]. We present two theorems which, combined, immediately give the result.

Theorem 3. *If $D \geq \log^3 n$, then no randomized algorithm can achieve better competitive ratio against an oblivious adversary than $\Omega(\min\{\sqrt{D \cdot \log n}, \lambda\}$.*

Theorem 4. *If $D \leq \log^3 n$, then no randomized algorithm can achieve better competitive ratio against an oblivious adversary than $\Omega(\min\{D^{2/3}, \lambda\}$.*

The proofs are almost identical, and therefore we present only the first one. The second one can be found in the full version of the paper.

Proof (of Thm. 3). We construct a probability distribution over inputs and prove that no deterministic algorithm, which knows this distribution, can have a competitive ratio better than $\Omega(\min\{\sqrt{D \cdot \log n}, \lambda\})$. Then one can apply Yao minmax theorem [11, 6] and Thm. 3 follows immediately.

We assume that n is a power of 2. If it is not the case, then the adversary can give requests only in the first $2^{\lfloor \log n \rfloor}$ nodes and put the other nodes exactly in the same point of space $\mathcal{X}$ as v_0. Then, for any algorithm ALG, that uses these additional nodes, there exists an algorithm ALG$'$ which uses v_0 instead and has cost at most as large as ALG. We can also assume that $\lambda \geq \sqrt{D}$. Otherwise we could use the proof from [4] and obtain the lower bound of $\Omega(\lambda)$.

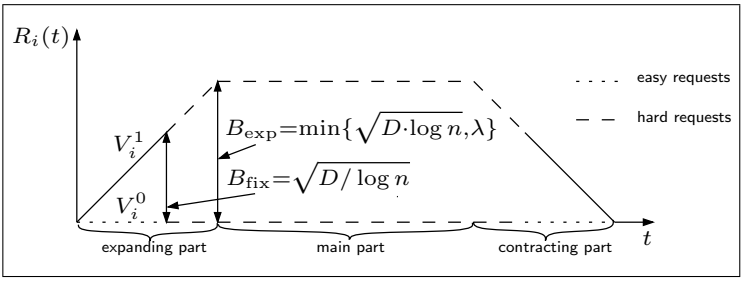


Fig. 2. Lower bound: The i-th phase of an epoch

We number all nodes from 0 to $n-1$. In the following we identify the nodes with their numbers. For all $1 \leq i \leq \log n$, let V_i^0 be the set of all the node numbers whose bit representation has i-th bit set to 0. V_i^1 is the set of all remaining nodes, i.e. the ones whose bit representation has the i-th bit set to 1. Let $B_{\text{main}} = D/\log n$, $B_{\text{exp}} = \min\{\sqrt{D \cdot \log n}, \lambda\}$, and $B_{\text{fix}} = \sqrt{D/\log n}$.

We divide time into epochs, each epoch containing $\log n$ phases, each of length $B_{\text{main}} + 2 \cdot B_{\text{exp}}$. In the i-th phase we divide the set V into two sets V_i^0 and V_i^1. All the nodes from set V_i^0 occupy a certain point in space $\mathcal{X}$; the same holds for nodes from V_i^1. Therefore, in step t of phase i, the distance between the sets V_i^0 and V_i^1 is well defined and we call it $R_i(t)$.

First, we describe the movement of nodes in the i-th phase. The situation is depicted in Fig. 2. Each phase consists of an *expanding part*, which lasts for B_{exp} steps, in which the sets V_i^0 and V_i^1 are moved apart, a *main part*, lasting for B_{main} steps, and a *contracting part* also of length B_{exp}, in which the sets V_i^0 and V_i^1 are brought closer. In the j-th step of the expanding part R_i is set to $j-1$. In all the steps of the main part the distance between sets V_i^0 and V_i^1 is exactly B_{exp}. Finally, in the j-th step of the contracting part, R_i is equal to $B_{\text{exp}} - j$.

The first B_{fix} requests in the expanding part of a phase and the last B_{fix} requests in the contracting part are given always at the node v_0. We call these requests *easy*. The requests in the remaining steps of a phase are given in one node — with probability 1/2 all are given at v_0 (belonging to V_i^0), and with probability 1/2 all are given at node $v_{2^{i-1}}$ (belonging to V_i^1). We call these requests *hard*, and we call the set, whose node issued these requests, a *requesting set*. Note that the main part is a (usually proper) subset of a sequence containing all the hard requests.

Consider any randomly generated $\log n$ phases of epoch $\mathcal{E}$. There exists a node v^+ (exactly one) which is in the requesting set for all the phases. As its distance to hard requests in each phase is 0, the cost incurred is at most 1 per request, which sums up to $\log n \cdot (B_{\text{main}} + 2 \cdot B_{\text{exp}}) = \mathcal{O}(D)$ in total. The optimal algorithm could move to v^+ at the beginning of the epoch, paying D. Finally, the cost incurred by easy requests is at most $2 \cdot \sum_{j=1}^{B_{\text{fix}}} j = \mathcal{O}(D/\log n)$ in each phase which accounts for $\mathcal{O}(D)$ for all requests in epoch $\mathcal{E}$.

On the other hand, for all i, when hard requests start in the i-th phase, any deterministic algorithm ALG has its page in the "wrong" set with probability 1/2.

At this point $R_i(t) = B_{\text{fix}}$. If ALG moves its page to the other set within the hard requests, it pays at least $D \cdot B_{\text{fix}}$. Otherwise, it has to pay B_{exp} for serving each of B_{main} hard requests in the main part of a phase. Therefore, from the linearity of expectation follows that the expected cost of ALG in the whole epoch $\mathcal{E}$ is

$$\mathbf{E}[C_{\text{ALG}}(\mathcal{E})] = \log n \cdot \frac{\min\{D \cdot B_{\text{fix}}, B_{\text{exp}} \cdot B_{\text{main}}\}}{2} = \Omega(\min\{D\lambda, D\sqrt{D \log n}\}) \ .$$

Thus, in any epoch the competitive ratio is at least $\Omega(\min\{\lambda, \sqrt{D \cdot \log n}\})$. We can generate an arbitrary number of epochs and the bound on ratio still holds. Therefore, the theorem follows. □

References

[1] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. In *Proc. of the 25th ACM Symp. on Theory of Computing (STOC)*, pages 164–173, 1993.

[2] Y. Bartal, M. Charikar, and P. Indyk. On page migration and other relaxed task systems. In *Proc. of the 8th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 43–52, 1997.

[3] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *Proc. of the 22nd ACM Symp. on Theory of Computing (STOC)*, pages 379–386, 1990.

[4] M. Bienkowski, M. Korzeniowski, and F. Meyer auf der Heide. Fighting against two adversaries: Page migration in dynamic networks. In *Proc. of the 16th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 64–73, 2004.

[5] D. L. Black and D. D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.

[6] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[7] M. Chrobak, L. L. Larmore, N. Reingold, and J. Westbrook. Page migration algorithms using work functions. In *Proc. of the 4th International Symp. on Algorithms and Computation (ISAAC)*, pages 406–415, 1993.

[8] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[9] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[10] J. Westbrook. Randomized algorithms for multiprocessor page migration. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:135–150, 1992.

[11] A. C.-C. Yao. Probabilistic computation: towards a uniform measure of complexity. In *Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.

Approximate Range Mode and Range Median Queries*

Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang

School of Computer Science, Carleton University,
Ottawa, Ontario, K1S 5B6, Canada
{jit, kranakis, morin, y_tang}@scs.carleton.ca

Abstract. We consider data structures and algorithms for preprocessing a labelled list of length n so that, for any given indices i and j we can answer queries of the form: What is the mode or median label in the sequence of labels between indices i and j. Our results are on approximate versions of this problem. Using $O(\frac{n}{1-\alpha})$ space, our data structure can find in $O(\log\log\frac{1}{\alpha}\, n)$ time an element whose number of occurrences is at least α times that of the mode, for some user-specified parameter $0 < \alpha < 1$. Data structures are proposed to achieve constant query time for $\alpha = 1/2, 1/3$ and $1/4$, using storage space of $O(n\log n), O(n\log\log n)$ and $O(n)$, respectively. Finally, if the elements are comparable, we construct an $O(\frac{n}{1-\alpha})$ space data structure that answers approximate range median queries. Specifically, given indices i and j, in $O(1)$ time, an element whose rank is at least $\alpha \times \lfloor |j-i+1|/2 \rfloor$ and at most $(2-\alpha) \times \lfloor |j-i+1|/2 \rfloor$ is returned for $0 < \alpha < 1$.

1 Introduction

Let $A = a_1, \ldots, a_n$ be a list of elements of some data type. We wish to construct data structures on A, such that we can quickly answer *range queries*. These queries take two indices i, j with $1 \le i \le j \le n$ and require computing $F(a_i, \ldots, a_j) = a_i \circ a_{i+1} \circ \cdots \circ a_{j-1} \circ a_j$. If the inverse of the operation "$\circ$" exists, then range queries have a trivial solution of linear space and constant query time. For example, if "$\circ$" is arithmetic addition (subtraction being its inverse), we precompute all the partial sums $b_i = a_1 + \cdots + a_i, i = 1, \ldots, n$, and the range query $F(a_i, \ldots, a_j) = a_i + \cdots + a_j$ can be answered in constant time by computing $b_j - b_{i-1}$. Yao [13] (see also Alon and Schieber [1]) showed that if "$\circ$" is a constant time semigroup operation (such as maximum or minimum) for which no inverse operation is allowed, and $a \circ b$ can be computed in constant time then it is possible to answer range queries in $O(\lambda(k, n))$ time using a data structure of $O(kn)$ size, for any integer $k \ge 1$. Here $\lambda(k, \cdot)$ is a slowly growing

* This work is supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 377–388, 2005.

function at the $\lfloor k/2 \rfloor$-th level of the primitive recursive hierarchy. For example, $\lambda(2,n) = O(\log n)$, $\lambda(3,n) = O(\log \log n)$ and $\lambda(4,n) = O(\log^* n)$.

Krizanc *et al* [10] studied the storage space versus query time tradeoffs for range mode and range median queries. These occur when F is the function that returns the mode or median of its input. Mode and median are two of the most important statistics [2, 3, 11, 12]. Given a set of n elements, a *mode* is an element that occurs at least as frequently as any other element of the set. If the elements are comparable (for example, real numbers), the *rank* of an element is its position in the sorted order of the input. For example, the rank of the minimum element is 1, and that of the maximum element is n. The ϕ-quantile is the element with rank $\lfloor \phi n \rfloor$. The 1/2-quantile is also called the *median*. Note the trivial solution does not work for range mode or range median queries as no inverse exists for either operation. Yao's approach does not apply either because neither range mode nor range median is associative and therefore not a semigroup query. Also, given two sets S_1 and S_2 and their modes (or medians), the mode (or median) of the union $S_1 \bigcup S_2$ cannot be computed in constant time. New data structures are needed for range mode and range median queries. Krizanc *et al* [10] gave a data structure of size $O(n^{2-2\epsilon})$ that can answer range mode queries in $O(n^\epsilon \log n)$ time, where $0 < \epsilon \leq 1/2$ is a constant representing storage space-query time tradeoff. For range median queries, they show that a data structure of size $O(n)$ can answer range median queries in $O(n^\epsilon)$ time and a faster $O(\log n)$ query time can be achieved using $O(\frac{n \log^2 n}{\log \log n})$ space.

In this paper we consider the approximate versions of range mode and range median queries. We show that if a small error is tolerable, range mode and range median queries can be answered much more efficiently in terms of storage space and query time. Given a sublist $S = a_i, a_{i+1}, \ldots, a_j$, an element is said to be an *approximate mode* of S if its number of occurrences is at least α times that of the actual mode of S, where $0 < \alpha < 1$ is a user-specified approximation factor. If the elements are comparable, the *median* is the element with rank (relative to the sublist) $\lfloor (j-i+1)/2 \rfloor$. An *$\alpha$-approximate median* of S is an element whose rank is between $\alpha \times \lfloor (j-i+1)/2 \rfloor$ and $(2-\alpha) \times \lfloor (j-i+1)/2 \rfloor$. Clearly, there could be several approximate modes and medians.

We show that approximate range mode queries can be answered in $O(\log \log_{\frac{1}{\alpha}} n)$ time using a data structure of size $O(n)$. We also show that constant query time can be achieved for $\alpha = 1/2, 1/3$ and $1/4$ using storage space of size $O(n \log n)$, $O(n \log \log n)$ and $O(n)$, respectively. We introduce a constant query time data structure for answering approximate range median queries. We also study the preprocessing time required for the construction of these data structures.

To the best of our knowledge, there is no previous work on approximate range mode or median queries. Two problems related to range mode and range median queries are *frequent elements* and *quantile summaries* over sliding windows [2, 11]. For many applications, data takes the form of continuous data streams, as opposed to finite stored data sets. Examples of such applications include network monitoring and traffic measurements, financial transaction logs and phonecall

records. All these applications view recently arrived data as more important than those a long time back. This preference for recent data is referred to as the *sliding window* model [6] in which the queries are answered regarding only the most recently observed n data elements. Lin *et al* [11] studied the problem of continuously maintaining quantile summaries over sliding windows. They devised an algorithm for approximate quantiles with an error of at most ϵn using $O(\frac{\log \epsilon^2 n}{\epsilon} + \frac{1}{\epsilon^2})$ space in the worst case for a fixed window size n. For windows of variable size at most n (such as timestamp-based windows in which the exact number of arriving elements cannot be predetermined), $O(\frac{\log^2 \epsilon n}{\epsilon^2})$ storage space is required. Arasu and Manku [2] improved both bounds to $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log n)$ and $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log \epsilon n \log n)$ respectively. They also proposed deterministic algorithms for the problem of finding all frequent elements (*i.e.,* elements with a minimum frequency of ϵn) using $O(\frac{1}{\epsilon} \log^2 \frac{1}{\epsilon})$ and $O(\frac{1}{\epsilon} \log^2 \frac{1}{\epsilon} \log \epsilon n)$ worst case space for fixed- and variable-size windows, respectively.

2 Approximate Range Mode Queries

Given a list of elements $a_1, \ldots, a_n$ and an approximation factor $0 < \alpha < 1$, the *approximate range mode queries* can be specified formally as follows.

INPUT: Two indices i, j with $1 \leq i \leq j \leq n$.
OUTPUT: An element x in $a_i, \ldots, a_j$ such that $F_x(a_i, \ldots, a_j) \geq \alpha \times F(a_i, \ldots, a_j)$, where $F_x(a_i, \ldots, a_j)$ is the frequency[1] of x in $a_i, \ldots, a_j$ and $F(a_i, \ldots, a_j) = \max_x F_x(a_i, \ldots, a_j)$ is the number of occurrences of a mode in $a_i, \ldots, a_j$.

Our data structure is based on the observation that given a fixed left end i of a query range, as the right end j of the range increases, the number of times the approximate mode changes as j varies from i to n is at most $\log_{\frac{1}{\alpha}}(n - i)$. This is because the same element can be output as approximate mode as long as no other element's frequency exceeds $1/\alpha$ times that of the current approximate mode. When the actual mode's frequency has exceeded $1/\alpha$ times that of the approximate mode, the approximate mode is replaced and the actual mode becomes the new approximate mode.

For example, given the list of 20 elements shown in Figure 1 and approximation factor $\alpha = 1/2$, b is an approximate mode of $a_1, \ldots, a_9$ because b occurs 2 times in the sublist, while the actual mode a occurs 4 times in the same sublist. But this is no longer true for query $a_1, \ldots, a_{10}$, as the number of occurrences of b is still 2 while the actual mode a occurs 5 times in the sublist ($F_a(a_1, \ldots, a_{10}) = 5$). In this case, either a or c ($F_c(a_1, \ldots, a_{10}) = 3$) is a valid approximate mode.

Assuming a is chosen to be the new approximate mode, it remains a valid approximate mode as the right end of the query range increases until $j = 19$ at which point the actual mode c occurs 11 times ($F_c(a_1, \ldots, a_{19}) = 11$). Since

[1] We use frequency and the number of occurrences interchangeably throughout the paper.

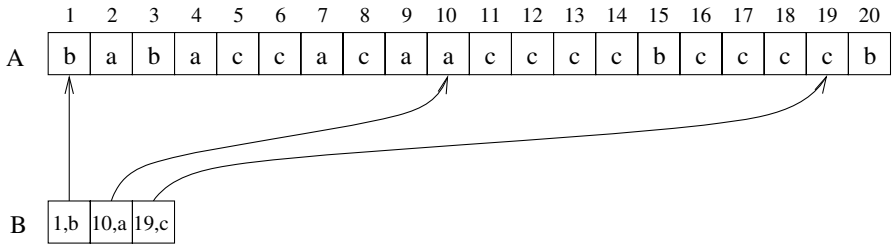


Fig. 1. $\alpha = 1/2$. A lookup table of size 3 is used for answering queries $a_1, \ldots, a_j$, $j = 1, \ldots, 20$. For example, a is an approximate mode of $a_1, \ldots, a_{15}$ because a occurs at least 5 times in the query range ($F_a(a_1, \ldots, a_{15}) = 5$) while no other element occurs more than 10 times until $j = 19$ ($F_c(a_1, \ldots, a_{19}) = 11$)

no other element (a or b) occurs more than or equal to half of the actual mode ($F_a(a_1, \ldots, a_{19}) = 5, F_b(a_1, \ldots, a_{19}) = 3$), c is now the only approximate mode until $j = 20$. Since an approximate mode remains valid until another element occurs more than $1/\alpha$ times the current approximate mode, the number of approximate modes that have to be stored is much less than the number of elements of the original list. As shown in the example of Figure 1, instead of storing the complete original array of 20 elements, a table of 3 approximate modes is used to answer all approximate range mode queries $a_1, \ldots, a_j, 1 \leq j \leq 20$.

Given an approximation factor α, all approximate range mode queries with a_1 being the left end: $a_1, \ldots, a_j$ $(1 \leq j \leq n)$ can be answered using $O(\log_{\frac{1}{\alpha}} n)$ storage space. The data structure is a lookup table $B = a_{c_1}, \ldots, a_{c_m} (1 \leq c_1 < c_2 < \ldots < c_m \leq n)$ in which we store m approximate modes. The first entry is always a_1 ($c_1 = 1$). The second entry a_{c_2} is the first element in A that occurs $\lceil 1/\alpha \rceil$ times, *i.e.*, $F_{a_{c_2}}(a_1, \ldots, a_{c_2}) = \lceil 1/\alpha \rceil$ and $F_{a_{c_2}}(a_1, \ldots, a_{c_2}) > F_{a_i}(a_1, \ldots, a_{c_2})$ for $\forall i \neq c_2$. In general, the kth entry in the table is the first element in A that occurs $\lceil 1/\alpha^{k-1} \rceil$ times in the sublist as the right end of the query range increases. Note that a_{c_k} is an approximate mode of $a_1, \ldots, a_j$ for any $c_k \leq j < c_{k+1}$ since a_{c_k} occurs at least $\lceil 1/\alpha^{k-1} \rceil$ times in $a_1, \ldots, a_j$ ($F_{a_{c_k}}(a_1, \ldots, a_j) \geq F_{a_{c_k}}(a_1, \ldots, a_{c_k}) = \lceil 1/\alpha^{k-1} \rceil$), while no other element occurs more than $1/\alpha^k$ times in the same range ($F_x(a_1, \ldots, a_j) < F_{c_{k+1}}(a_1, \ldots, a_{c_{k+1}}) = \lceil 1/\alpha^k \rceil$).

The last approximate mode in the table, a_{c_m}, occurs at least $\lceil 1/\alpha^{m-1} \rceil$ times in $a_1, \ldots, a_n$. It follows immediately that the number of approximate modes stored in the lookup table m is at most $\log_{\frac{1}{\alpha}} n + 1$.

To answer approximate range mode queries in the range $a_1, \ldots, a_j$, binary search is used to find in $O(\log \log_{\frac{1}{\alpha}} n)$ time the largest c_k that is less than or equal to j and output a_{c_k} as the answer.

Lemma 1. *There is a data structure of size $O(\log_{\frac{1}{\alpha}} n)$ that can answer approximate range mode queries in the range $a_1, \ldots, a_j$ $(1 \leq j \leq n)$ in $O(\log \log_{\frac{1}{\alpha}} n)$ time.*

An immediate application of Lemma 1 is a data structure for answering approximate range mode queries with arbitrary ends. The data structure is a collec-

tion of n lookup tables ($T_i, i = 1, \ldots, n$), one table for each left end. An auxiliary array of n pointers is used to locate a table in $O(1)$ time. A query $a_i, \ldots, a_j$ can be answered by first locating table T_i in $O(1)$ time, and then searching in T_i to find the approximate mode of $a_i, \ldots, a_j$, which takes $O(\log\log_{\frac{1}{\alpha}} n)$ time since T_i contains at most $O(\log_{\frac{1}{\alpha}}(n-i)) = O(\log_{\frac{1}{\alpha}} n)$ approximate modes.

Corollary 1. *There is a data structure of size $O(n \log_{\frac{1}{\alpha}} n)$ that can answer approximate range queries in $O(\log\log_{\frac{1}{\alpha}} n)$ time.*

2.1 An Improvement Based on Persistent Search Trees

We have seen that by maintaining a lookup table T_i of size $O(\log_{\frac{1}{\alpha}} n)$ for each left end i ($1 \le i \le n$) and using $O(n \log_{\frac{1}{\alpha}} n)$ total storage space, any approximate range mode query in the range $a_i, \ldots, a_j$ can be answered in $O(\log\log_{\frac{1}{\alpha}} n)$ time. Given a fixed left end i, storing an answer for each right end j is not necessary since the answer to the query changes less frequently as j varies. The approximate modes of two query ranges with adjacent right ends are unlikely to be different. In this section, we pursue this idea and show that storage of a complete lookup table for each left end is not necessary because of the similarity between two tables with adjacent left ends.

To see how the approximate range mode changes gradually as the two ends of a query range move, we need a systematic way to keep track of the range within which the current approximate mode remains a valid approximation of the actual mode and its number of occurrences in that range. As the query range changes, the frequency of the current approximate mode may also change. Once it drops below a predetermined threshold value (f_{low}, the calculation of which will be discussed next), a new approximate mode is chosen and the query range updated.

As shown in Table 1, each entry in the lookup table is a 5-tuple (f_{low_r}, f_{high_r}, q_r, ans_r, f_{ans_r}). Given an approximation factor α, $[f_{low_r}, f_{high_r}]$ are precomputed for $r = 1, \ldots, 2\lceil \log_{\frac{1}{\alpha}} n \rceil$ and remain the same for all tables.

As noted before, the ith table T_i corresponds to all the range queries with the same left end i. A counter is set for each element to keep track of its frequency as the right end j varies. Given the fixed left end i, as the right end j proceeds,

Table 1. $f_{low_1} = 1$, $f_{high_1} = 1$, $f_{low_{r+1}} = f_{high_r} + 1$, $f_{high_{r+1}} = \lceil f_{low_r}/\alpha \rceil + 1$, $F(a_i, \ldots, a_{q_r}) = f_{high_r}$, $f_{ans_r} = F_{ans_r}(a_i, \ldots, a_{q_r})$, $f_{low_r} \le f_{ans_r} \le f_{high_r}$

Frequency Range	Query Range	Answer
...		
$[f_{low_r}, f_{high_r}]$	q_r	(ans_r, f_{ans_r})
$[f_{low_{r+1}}, f_{high_{r+1}}]$	q_{r+1}	$(ans_{r+1}, f_{ans_{r+1}})$
...		

ans_r is the first element whose frequency in $a_i, \ldots, a_j$ reaches f_{high_r}, and q_{r+1} is the rightmost point up to which ans_r remains a valid approximate mode, *i.e.,* no other element has a frequency higher than f_{high_r}/α. Given a query $a_i, \ldots, a_j$ with $q_r \leq j < q_{r+1}$, ans_r is a valid approximate mode since its frequency is at least f_{high_r} while no other element has a frequency higher than or equal to $f_{high_{r+1}} - 1 = \lceil f_{low_r}/\alpha \rceil$. To see how the subsequent tables are built based on T_i with minimum number of changes, the right end of the query range is fixed, as the left end of the query range proceeds, ans_r's frequency may decrease, but it remains a valid approximate mode as long as $f_{ans_r} \geq f_{low_r}$ and it is copied to the next table along with a possibly smaller f_{ans_r} (Note that f_{ans_r} is needed only for bookkeeping purposes). The only time that ans_r must change for a table is when its frequency drops below f_{low_r}. At this point we update ans_r and the new approximate mode is the first element whose frequency reaches f_{high_r} with respect to the current left end of query range. The query range q_r is also updated to reflect the change on the approximate mode ($F_{ans_r}(a_i, \ldots, a_{q_r}) = f_{high_r}$).

Table 2 shows the data structure for answering approximate range mode queries on the same list as in Figure 1. For example, to look up the approximate mode of $a_4, \ldots, a_{12}$, we search in T_4 and find the entry with the largest q_r that is smaller than 12: $\{[4, 5], 10, (a, 4)\}$. This tells us that, in the sequence of $a_4, \ldots, a_{12}$, a occurs at least 4 times ($F_a(a_4, \ldots, a_{12}) \geq F_a(a_4, \ldots, a_{10}) = 4$) and no element occurs more than 8 times ($F_x(a_2, \ldots, a_{12}) \leq F(a_2, \ldots, a_{17}) - 1 = 8$).

After T_1 is built, T_i ($i \geq 2$) is built based on T_{i-1} with necessary updates. The number of updates made is given by the following lemma.

Lemma 2. *If the rth row of the table is updated in T_i, then it does not need to be updated in T_k for any $i < k < i + 1/\alpha^{\lfloor r/2 \rfloor}$.*

Proof. When the rth row is updated in T_i, we set ans_r to be the first element such that $F_{ans_r}(a_i, \ldots, a_{q_r}) = f_{high_r}$. Its frequency f_{ans_r} is initially f_{high_r} in T_i. Although f_{ans_r} may decrease as i increases, ans_r does not need to be updated again until f_{ans_r} drops below f_{low_r}, which takes at least $f_{high_r} - (f_{low_r} - 1) = f_{high_r} - f_{high_{r-1}} = 1/\alpha^{\lfloor r/2 \rfloor}$ steps.

Note that there are no more than $2\lceil \log_{\frac{1}{\alpha}} n \rceil$ rows in a table and every time we build a new table, the first row needs to be updated. Lemma 2 shows that the rth ($r \geq 2$) row changes no more than $\alpha^{\lfloor r/2 \rfloor} n$ times during the construction of all n tables. The total number of updates we have to make is given by the following theorem.

Theorem 1. *The total number of updates we have to make is $O(n/(1-\alpha))$.*

Proof. Total number of updates $\leq n + \sum_{r=2}^{2\lceil \log_{\frac{1}{\alpha}} n \rceil} \alpha^{\lfloor r/2 \rfloor} n = O(\frac{n}{1-\alpha})$.

Theorem 1 says that, the majority of the table entries can be reconstructed by referring to other tables. In other words, although n lookup tables are needed to answer approximate range mode queries, many of them share common entries. A persistent search tree [8] is used to store the tables efficiently. It has the property

Table 2. An example showing the data structure for answering 1/2-approximate range mode queries on a list of 20 elements. Updates are in bold

T_i	T_1	T_2	T_3	T_4	T_5
a_i	b	a	b	a	c
$[1,1]$	$1,(\mathbf{b},1)$	$2,(\mathbf{a},1)$	$3,(\mathbf{b},1)$	$4,(\mathbf{a},1)$	$5,(\mathbf{c},1)$
$[2,3]$	$7,(\mathbf{a},3)$	$7,(a,3)$	$7,(a,2)$	$7,(a,2)$	$8,(\mathbf{c},3)$
$[4,5]$	$10,(\mathbf{a},5)$	$10,(a,5)$	$10,(a,4)$	$10,(a,4)$	$12,(\mathbf{c},5)$
$[6,9]$	$17,(\mathbf{c},9)$	$17,(c,9)$	$17,(c,9)$	$17,(c,9)$	$17,(c,9)$
$[10,13]$	$20,(\mathbf{c},11)$	$20,(c,11)$	$20,(c,11)$	$20,(c,11)$	$20,(c,11)$

T_i	T_6	T_7	T_8	T_9	T_{10}
a_i	c	a	c	a	a
$[1,1]$	$6,(\mathbf{c},1)$	$7,(\mathbf{a},1)$	$8,(\mathbf{c},1)$	$9,(\mathbf{a},1)$	$10,(\mathbf{a},1)$
$[2,3]$	$8,(c,2)$	$10,(\mathbf{a},3)$	$10,(a,2)$	$10,(a,2)$	$13,(\mathbf{c},3)$
$[4,5]$	$12,(c,4)$	$14,(\mathbf{c},5)$	$14,(c,5)$	$14,(c,4)$	$14,(c,4)$
$[6,9]$	$17,(c,8)$	$17,(c,7)$	$17,(c,7)$	$17,(c,6)$	$17,(c,6)$
$[10,13]$	$20,(c,10)$	—	—	—	—

T_i	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
a_i	c	c	c	c	b
$[1,1]$	$11,(\mathbf{c},1)$	$12,(\mathbf{c},1)$	$13,(\mathbf{c},1)$	$14,(\mathbf{c},1)$	$15,(\mathbf{b},1)$
$[2,3]$	$13,(\mathbf{c},3)$	$13,(c,2)$	$16,(\mathbf{c},3)$	$16,(c,2)$	$18,(\mathbf{c},3)$
$[4,5]$	$14,(c,4)$	$17,(\mathbf{c},5)$	$17,(c,4)$	$19,(\mathbf{c},5)$	$19,(c,4)$
$[6,9]$	$17,(c,6)$	$19,(\mathbf{c},7)$	$19,(c,6)$	—	—
$[10,13]$	—	—	—	—	—

T_i	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}
a_i	c	c	c	c	b
$[1,1]$	$16,(\mathbf{c},1)$	$17,(\mathbf{c},1)$	$18,(\mathbf{c},1)$	$19,(\mathbf{c},1)$	$20,(\mathbf{b},1)$
$[2,3]$	$18,(c,3)$	$18,(c,2)$	$19,(\mathbf{c},2)$	—	—
$[4,5]$	$19,(c,4)$	—	—	—	—
$[6,9]$	—	—	—	—	—
$[10,13]$	—	—	—	—	—

that the query time is $O(\log m)$ where m is the number of entries in each table, and the storage space is $O(1)$ per update. In the case of approximate range mode queries, although each table can have as many as $2\lceil \log_{\frac{1}{\alpha}} n \rceil$ entries, many tables share the same entries and the number of different nodes in the persistent tree is $O(n/(1-\alpha))$, one for each update, and the query time for a node is $O(\log \log_{\frac{1}{\alpha}} n)$.

To build the search tree, we need to keep track of the frequency of each element as query range varies. The idea presented in [7] leads to an algorithm that maintains a counter for each element and the total preprocessing time is $O(n \log_{\frac{1}{\alpha}} n + n \log n)$.

Theorem 2. *There exists a data structure of size $O(n/(1-\alpha))$ that can answer approximate range mode queries in $O(\log\log_{\frac{1}{\alpha}} n)$ time, and can be constructed in $O(n\log_{\frac{1}{\alpha}} n + n\log n)$ time.*

2.2 Lower Bounds

Next we show there is no faster worst case algorithm to compute the approximate mode for any fixed approximation factor α. To see this, let A be a list of $n/\lceil 1/\alpha\rceil$ elements and $B = A \ldots A = b_1, \ldots, b_n$ is a list of length n obtained by repeating A $\lceil 1/\alpha\rceil$ times. The problem of testing whether there exist two identical elements in A (also called *element uniqueness*) can be reduced to asking if the mode of B occurs more than $\lceil 1/\alpha\rceil$ times. In the case of approximate range mode query, the answer to query $b_1, \ldots, b_n$ is an element whose frequency is greater than 1 if and only if the actual mode of B occurs more than $\lceil 1/\alpha\rceil$ times.

In the algebraic decision tree model of computation, the running time of determining whether all the elements of A are unique is known to have a complexity of $\Omega(n\log n)$ [4]. However, this problem can also be solved by doing a single approximate range mode query $b_1, \ldots, b_n$ after preprocessing B, which implies the same lower bound holds for approximate range mode queries.

Theorem 3. *Let $P(n)$ and $Q(n)$ be the preprocessing and query times, respectively, of a data structure for answering approximate mode queries, we have $P(n) + Q(n) = \Omega(n\log n)$.*

On the other hand, $\Omega(n)$ storage space is required by any data structure that supports approximate range mode queries since the original list can be reconstructed by doing queries $(a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n)$, regardless of what value α is.

2.3 Constant Query Time

Yao [13] (see also Alon *et al* [1]) showed that if a query $a_i, \ldots, a_j$ can be answered by combining answers of queries $a_i, \ldots, a_x$ and $a_{x+1}, \ldots, a_j$ in constant time, then $\Theta(n\lambda(k, n))$ time and space is both necessary and sufficient to answer range queries in at most k steps. We adapt the same approach to develop constant query time data structures for some special cases of approximate range mode queries. Namely, the approximation factor $\alpha = 1/k$ where k is some positive integer.

The following lemma says that, if we can partition the range $a_i, \ldots, a_j$ into k intervals and we know the mode of each interval, then one of these is an approximate mode, for $\alpha = 1/k$.

Lemma 3. *If $\{B_1, \ldots, B_k\}$ is a partition of $a_i, \ldots, a_j$ then $max_p F(B_p) \geq F(a_i, \ldots, a_j)/k$.*

Proof. By contradiction. Otherwise for any element x we have $F_x(a_i, \ldots, a_j) = \sum_{p=1}^{k} F_x(B_p) \leq k \times max_p F(B_p) < F(a_i, \ldots, a_j)$.

Yao [13] and Alon *et al* [1] gave an optimal scheme of using a minimum set of intervals such that any range $a_i, \ldots, a_j$ can be covered by at most k such intervals.

Lemma 4. *(Yao [13], Alon et al [1]) There exists a set of $O(n\lambda(k,n))$ intervals such that any query range $a_i, \ldots, a_j$ can be partitioned into at most k of these intervals. Furthermore, given i and j, these intervals can be found in $O(k)$ time.*

Given Lemma 3 and Lemma 4, we immediately obtain a constant query time solution to approximate range mode queries with approximation factor $1/k$. By precomputing the mode of each interval, a query can be answered by first fetching the partition of the query range, which is a set of at most k intervals, and then outputting the one with the highest frequency among k modes of these intervals.

Theorem 4. *There exists a data structure of size $O(n\lambda(k,n))$ that can answer approximate range mode in $O(k)$ time, for $\alpha = 1/k$.*

The results in Theorem 4 can be further improved using a table lookup trick for $k \geq 4$. We partition the list into $n/\log n$ blocks of size $\log n$, $B_i = a_{(i-1)\log n+1}, \ldots, a_{i\log n}, i = 1, \ldots, n/\log n$. By Lemma 4, there exists a set of $O((n/\log n)\lambda(2, n/\log n)) = O(n)$ intervals such that any range with both ends at the boundaries of the blocks can be covered with at most 2 of these intervals. The exact modes of these intervals are precomputed. Inside every block, exact modes of 2 intervals are precomputed for each element, one interval is between the element and the beginning of the block and the other interval between the element and the end of the block. Any query range that spans more than one block can be partitioned into at most 4 intervals. The first one is the (possibly partial) block in which the range starts; the last one is the the (possibly partial) block in which the range ends and the other (at most) two intervals in between cover all the remaining blocks (if any). Of these intervals the modes are all precomputed, and the one with the highest frequency is a 1/4-approximation of the actual mode.

It remains to show that a query within a block can also be answered in $O(1)$ time. This is done by recursively partitioning the $\log n$ block into $\log n/\log\log n$ blocks of size $\log\log n$. The same method above is used to preprocess these blocks, and the result is a data structure of $O(n)$ size that can answer any query that spans more than one $\log\log n$-block in $O(1)$ time.

To answer queries within a $\log\log n$-block, a standard data structure trick [9] of canonical subproblems is used. Note that we can normalize each block by replacing each element with the index of its first occurrence within the block. Because such index is a non-negative integer that is at most $\log\log n$ and each block consists of $\log\log n$ such values, there are at most $(\log\log n)^{\log\log n}$ different blocks. Among all $n/\log\log n$ blocks of size $\log\log n$, many are of the same type. Thus, preprocessing of each block is unnecessary, and storage space can be reduced by preprocessing a block once and reusing the results for all blocks of the same type. The data structure used is a $\log\log n \times \log\log n$ matrix that can answer range mode query in constant time. All the queries in blocks of the same

type are done in the same matrix. There are at most $(\log\log n)^{\log\log n}$ possible matrices which require $O((\log\log n)^{\log\log n}(\log\log n)^2) = o(n)$ storage space.

Theorem 5. *There exists a data structure of size $O(n)$ that can answer approximate range mode queries in $O(1)$ time, for $\alpha = 1/4$.*

3 Approximate Range Median Queries

In this section, we consider approximate range median queries on a list of comparable elements $A = a_1, \ldots, a_n$. Given an approximation factor $0 < \alpha < 1$, our task is to preprocess A so that, given indices $1 \le i \le j \le n$, we can quickly return an element of $a_i, \ldots, a_j$ whose rank is between $\alpha \times \lfloor (j-i+1)/2 \rfloor$ and $(2-\alpha) \times \lfloor (j-i+1)/2 \rfloor$.

To simplify the presentation we assume $n = 2^d$ for some integer $d \ge 1$. Generalization to arbitrary n is straightforward. As shown in Figure 2, d levels of partitions are used. In level i, the list is partitioned into 2^i non-overlapping blocks of size $n/2^i$. Exact medians of sublists with both ends at the boundaries of the blocks (up to $2\lceil 2\alpha/(1-\alpha)\rceil$ blocks away) are precomputed. The idea behind our algorithm is that, if a query $a_i, \ldots, a_j$ spans many blocks, then the contribution of the first and last block is minimal and can be ignored. Instead, we could simply answer the (precomputed) median of the union of the internal blocks. On the other hand, since we are using many different block sizes, we can choose a partition level so that $a_i, \ldots, a_j$ spans just enough blocks for the strategy above to give a valid approximation. This ensures that we do not have to precompute too many medians.

At the lowest level, $a_1, \ldots, a_n$ is partitioned into n blocks each consisting of a single element. We precompute for each $i = 1, \ldots, n$ all the medians of $a_i, \ldots, a_j$, for $i \le j \le i + 2\lceil 2\alpha/(1-\alpha)\rceil - 1$. This enables us to answer queries of length no more than $2\lceil 2\alpha/(1-\alpha)\rceil$ in $O(1)$ time using $O(n/(1-\alpha))$ space. To answer queries of length greater than $\lceil 2\alpha/(1-\alpha)\rceil - 1$, we search in a higher level where the query spans at least $\lceil 2\alpha/(1-\alpha)\rceil$ but no more than $2\lceil 2\alpha/(1-\alpha)\rceil$ complete blocks. Suppose the query spans $\lceil 2\alpha/(1-\alpha)\rceil \le c \le 2\lceil 2\alpha/(1-\alpha)\rceil$ complete blocks in level i, let l denote the length of the query, we have $cn/2^i \le l < (c+2)n/2^i$. The median of the union of these c blocks is precomputed and its rank in the query range is at least $cn/2^{i+1} \ge \alpha l/2$ and at most $cn/2^{i+1} + (l - cn/2^i) \le (2-\alpha)l/2$, in other words, it is an α-approximate median of the query range.

In the subsequent subsections we give the preprocessing time, storage space and query time of our data structure for answering approximate range median queries.

3.1 $O(n \log n/(1-\alpha)^2)$ Preprocessing Time

We preprocess $A = a_1, \ldots, a_n$ and build d lookup tables as follows. To build T_i $(1 \le i \le d)$, we partition A into 2^i blocks each of size $n/2^i$: $B_{i_j} = a_{(j-1)\times n/2^i+1}$, $\ldots$, $a_{j\times n/2^i}$, $j = 1, \ldots, 2^i$. T_i has 2^i entries $(T_{i_j}, j = 1, 2, \ldots, 2^i)$, each corre-

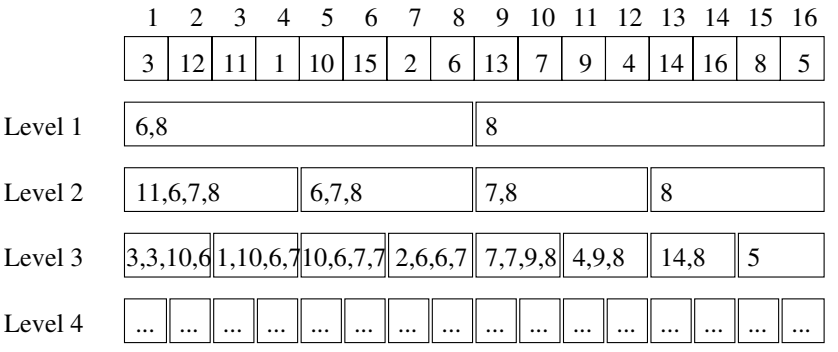


Fig. 2. $\alpha = 1/2$. For each block up to $2\lceil 2\alpha/(1-\alpha)\rceil = 4$ medians are precomputed. For example, associated with the $2nd$ block in level 3 are 4 medians, each corresponds to a union of up to 4 consecutive blocks: $1 = Median(B_{3_2})$; $10 = Median(B_{3_2} \bigcup B_{3_3})$; $6 = Median(B_{3_2} \bigcup \ldots \bigcup B_{3_4})$; $7 = Median(B_{3_2} \bigcup \ldots \bigcup B_{3_5})$. Note that a 1/2-approximate range median query that spans more than 4 complete blocks also spans at least 2 complete blocks in the next higher level and therefore can be answered in a higher level with sufficient accuracy. Range median queries are answered by looking in the level where the query range spans just enough number of complete blocks. For example, query $a_2, \ldots, a_{11}$ spans 4 complete level 3 blocks ($B_{3_2} \bigcup \ldots \bigcup B_{3_5}$) but only 1 complete level 2 block (B_{2_2}). Therefore, the $4th$ entry in the $2nd$ level 3 block ($T_{3_2}(4) = Median(B_{3_2} \bigcup \ldots \bigcup B_{3_5}) = 7$, whose rank in $a_2, \ldots, a_{11}$ is 4) is output as the approximate median, while the rank of the actual median is 5 in the sublist of 10 elements

sponds to a block B_{i_j} and contains a pointer to a list of $2\lceil 2\alpha/(1-\alpha)\rceil$ elements of A: $T_{i_j}(k) = Median(B_{i_j} \bigcup \ldots \bigcup B_{i_{j+k-1}})$, $k = 1, \ldots, 2\lceil 2\alpha/(1-\alpha)\rceil$. $Median(B_{i_j} \bigcup \ldots \bigcup B_{i_{j+k-1}})$ is the median of $B_{i_j} \bigcup \ldots \bigcup B_{i_{j+k-1}}$, which can be computed in $O(kn/2^i)$ time [5]. There are $\log n$ tables to be computed. It follows that the total preprocessing time is $\sum_{i=1}^{\log n} \sum_{j=1}^{2^i} \sum_{k=1}^{2\lceil \frac{2\alpha}{1-\alpha}\rceil} O(\frac{kn}{2^i}) = O\left(\frac{n \log n}{(1-\alpha)^2}\right)$.

3.2 $O(n/(1-\alpha))$ Storage Space

The data structure for answering approximate range median queries is a set of $\log n$ lookup tables. Each table T_i $(1 \le i \le \log n)$ has $O(2^i)$ entries and each entry is a list of at most $2\lceil 2\alpha/(1-\alpha)\rceil$ precomputed range medians, the total space needed to store all $\log n$ tables is $\sum_{i=1}^{\log n} O(2^i \alpha/(1-\alpha)) = O(n\alpha/(1-\alpha)) = O(n/(1-\alpha))$.

3.3 $O(1)$ Query Time

Next we show how to compute an approximate range median of $a_i, \ldots, a_j$.

1. Compute the length of the query $l = j - i + 1$, then locate table T_p in which to continue the search: $p = \lceil \log \frac{2\alpha n}{(1-\alpha)l} \rceil$.

2. Compute $b_i = \lceil \frac{i2^p}{n} \rceil$ and $b_j = \lfloor \frac{j2^p}{n} \rfloor$. Since $p = \lceil \log \frac{2\alpha n}{(1-\alpha)l} \rceil < \log \frac{2\alpha n}{(1-\alpha)l} + 1 = \log \frac{4\alpha}{(1-\alpha)l}$, we have $2^p < \frac{4\alpha n}{(1-\alpha)l}$ and $b_j - b_i = \lfloor \frac{j2^p}{n} \rfloor - \lceil \frac{i2^p}{n} \rceil \leq \frac{(j-i)2^p}{n} \leq \frac{4(j-i)\alpha}{(1-\alpha)l} \leq \frac{4\alpha}{1-\alpha}$. In other words, $Median(B_{p_{b_i}} \bigcup \ldots \bigcup B_{p_{b_j}})$ is stored in a list to which a pointer is stored in $T_{p_{b_i}}$.
3. Output $T_{p_{b_i}}(b_j - b_i) = Median(B_{p_{b_i}} \bigcup \ldots \bigcup B_{p_{b_j}})$ as the answer.

Because each of the three steps above takes $O(1)$ time, the time required for answering the approximate range median query is $O(1)$.

Theorem 6. *There exists a data structure of size $O(n/(1-\alpha))$ that can answer approximate range median queries in $O(1)$ time, and can be built in $O(n \log n/(1-\alpha)^2)$ time.*

References

1. N. Alon and B. Schieber. Optimal preprocessing for answering on-line product queries. Technical Report 71/87, Tel-Aviv University, 1987.
2. A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2004.
3. S. Battiato, D. Cantone, D. Catalano, G. Cincotti, and M. Hofri. An efficient algorithm for the approximate median selection problem. In *Proceedings of the Fourth Italian Conference*, 2000.
4. M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th annual ACM Symposium on the Theory of Computing*, pages 80–86, 1983.
5. M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.
6. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal of Computing*, 31(6):1794–1813, 2002.
7. E. D. Demaine, A. Lópex-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 348–360, 2002.
8. J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.
9. H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985.
10. D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. In *Proceedings of the 14th Annual International Symposium on Algorithms and Computation (ISAAC 2003)*, 2003.
11. X. Lin, H. Liu, J. Xu, and J. X. Yu. Continuously maintaining quantile summaries of the most recent n elements over a data stream. In *Proceedings of the 20th International Conference on Data Engineering*, 2004.
12. G. S. Manku, S. Rajagopalan, and B. Lindsay. Approximate median and other quantiles in one pass and with limited memory. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 426–435, 1998.
13. A. C. Yao. Space-time tradeoff for answering range queries. In *Proceedings of the 14th annual ACM Symposium on the Theory of Computing*, pages 128–136, 1982.

Topological Automata

Emmanuel Jeandel

LIP (UMR CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1),
École Normale Supérieure de Lyon,
46 allée d'Italie 69364 LYON cedex 07 France
Emmanuel.Jeandel@ens-lyon.fr

Abstract. We give here a new, topological, definition of automata that extends previous definitions of probabilistic and quantum automata. We then prove in an unified framework that deterministic or non-deterministic probabilistic and quantum automata with an isolated threshold recognize only regular languages.

Keywords: Finite Automata, Formal Languages, Probabilistic Automata, Quantum Automata.

1 Introduction

A result in probabilistic automata theory states that any language accepted with an isolated threshold λ (no word is accepted with a probability falling in some fixed neighborhood of λ) is a regular language [1]. The same fact also surprisingly holds for quantum automata [2, 3].

In fact, the two proofs are very similar, and it is tempting to generalize them to many others models of automata. Bozapalidis [4] gives such a generalization, considering automata as vector spaces on which the monoid $\Sigma^\star$ acts linearly. This work allows him to give a unified proof of the fact that the two previous models of automata recognize only regular languages when limited to isolated threshold. However, no result about non-deterministic automata has been obtained in this way.

We will provide here a new way to define automata, based on some topological considerations, which will extend naturally quantum and probabilistic automata. We will then prove how some properties about topological spaces (mainly compactness and spaces with finitely many components) ensure that languages accepted by separation, the topological equivalent on an isolated threshold, are regular.

The beauty of topological spaces is that the notion of non-deterministic automata can be defined naturally, and the determinization process of classical finite automata is easy to transpose: instead of non-empty sets, which have no special properties for topological spaces, we will use non-empty compact sets. We will then see that the fundamental property is also true for non-deterministic quantum or probabilistic automata: they accept only regular languages when limited to isolated threshold. This result is new for quantum automata, and provide an easier proof than [5] for probabilistic automata.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 389–398, 2005.

2 Topological Automata

We will assume that the reader is familiar with the classical theory of formal languages and finite automata, in particular the Myhill-Nerode theorem. We refer the reader to [6] for more details. The empty word will be denoted by ϵ.

Many models of automata behave in the same way: from an initial configuration, the automata flows through configurations by reading letters. When the last letter is read, we decide if the word is accepted by observing the configuration. We translate here this procedure in a topological fashion.

2.1 Definitions

Definition 1. *A class of automata is a tuple* $(\mathcal{O}, \mathcal{C}, \mathcal{M})$ *where*

- $\mathcal{O}$ *is a topological monoid :* $\mathcal{O}$ *is a metric space and also a monoid for which the multiplication* $\circ$ *is a continuous map.* $\mathcal{O}$ *is called the set of* operators.
- $\mathcal{C}$ *is a metric space;* $\mathcal{O}$ *right acts continuously on* $\mathcal{C}$*. The set* $\mathcal{C}$ *is called the set of* configurations. *The action will be denoted* $\bullet$.
- $\mathcal{M}$ *is a set of continuous maps from* $\mathcal{C}$ *to* $\mathbb{R}$*.* $\mathcal{M}$ *is called the set of* measures. *Furthermore, for each* $m \in \mathcal{M}$*, and each operator* $o \in \mathcal{O}$*, the map* $c \mapsto m(c \bullet o)$ *is in* $\mathcal{M}$.

We will now see how the classical definitions of automata fit in this framework

- The class of (deterministic) finite automata with n states is defined by :
  - The set of configurations is $\mathcal{C} = \{1 \dots n\}$ also written Q.
  - The set of operators $\mathcal{O}$ is the set of maps from Q to Q. $\mathcal{O}$ is a monoid for the multiplication $(f \circ g) : c \mapsto g(f(c))$.
    $\mathcal{O}$ right acts canonically on $\mathcal{C} : \bullet : (c, f) \mapsto f(c)$
  - The set of measures $\mathcal{M}$ is the set of characteristic functions of (non-empty) subsets of Q.

  All the sets are used with the discrete topology. As all the metric spaces defined are finite, all maps are obviously continuous.
- The class of probabilistic automata with n states is defined as follows
  - The set of configurations is the set of row $n \times 1$ real vectors $\mathbf{v}^T$ with positive coefficients and such that $\sum \mathbf{v}_i = 1$
  - The set of operators is the set of $n \times n$ stochastic matrices. $\mathcal{O}$ right acts canonically on $\mathcal{C} : \bullet : (\mathbf{v}^T, \mathcal{M}) \mapsto \mathbf{v}^T \mathcal{M}$
  - The set of measures $\mathcal{M}$ is the set of maps $\mathbf{v}^T \mapsto \mathbf{v}^T \mathbf{w}$ for all column vectors $\mathbf{w}$ with coefficients in $[0, 1]$.

  All involved spaces are naturally used with the underlying topology from $\mathbb{R}^n$ and $\mathbb{R}^{n^2}$, so that all previously defined maps are continuous.
- We define exactly in the same manner the class of (Measure-Once) quantum automata [2] with n states : $\mathcal{C}$ is the set of $n \times 1$ vectors of euclidean norm 1, $\mathcal{O}$ is the set of $n \times n$ unitary matrices, $\mathcal{M}$ is the set of maps $\mathbf{v}^T \mapsto \|\mathbf{v}^T \mathcal{P}\|^2$ for all projection matrices $\mathcal{P}$.

The definition of an automaton is now natural :

Definition 2. *A deterministic automaton of a class* $(\mathcal{O}, \mathcal{C}, \mathcal{M})$ *over an alphabet* Σ *is given by :*

- *A configuration* c_0 *of* $\mathcal{C}$ *called the* initial configuration
- *An operator* $X_i \in \mathcal{C}$ *for each* $i \in \Sigma$
- *A measure* $m \in \mathcal{M}$

Definition 3. *Given an automaton* $\mathcal{A} = (c_0, (X_i)_i, m)$*, the value of a word* ω *is* $Val_{\mathcal{A}}(\omega) = m(c_0 \cdot X_\omega)$*, where* X_ω *denotes* $X_{\omega_1} \circ \cdots \circ X_{\omega_{|\omega|}}$ *(*X_ϵ *is the unit of* $\mathcal{O}$*)*

These definitions are coherent in the sense that the value of a (probabilistic/quantum) finite automaton is precisely its probability acceptance.

An automaton of a class does in fact not depend on all the class : The set of configurations $\mathcal{A}$ may visit is the set $\{c_0 X_\omega, \omega \in \Sigma^\star\}$ which may be not all the configurations. This may be the case even for some finite automaton where some states cannot be reached. We now define the purged automaton :

Definition 4. *Given an automaton* $\mathcal{A} = (c_0, (X_i)_i, m)$ *of a class* $(\mathcal{O}, \mathcal{C}, \mathcal{M})$*, the domain of* $\mathcal{A}$*, denoted by* $\mathrm{Dom}(\mathcal{A})$ *is the class defined by*

- *The set of operators* $\hat{\mathcal{O}}$ *is the closure of the set* $\{X_\omega, \omega \in \Sigma^\star\}$ *with the induced topology from* $\mathcal{O}$
- *The set of configurations* $\hat{\mathcal{C}}$ *is the closure of* $c_0 \cdot \hat{\mathcal{O}}$ *with the induced topology from* $\mathcal{C}$*.*
- *The set of measures* $\hat{\mathcal{M}}$ *is given by the maps* $c \mapsto m(c \cdot o)$ *for* $o \in \hat{\mathcal{O}}$

The purge of $\mathcal{A}$ *is the automaton* $\hat{\mathcal{A}} = (c_0, (X_i)_i, m')$ *of the class* $\mathrm{Dom}(\mathcal{A})$ *where* m' *is the restriction of* m *to* $\hat{\mathcal{C}}$*.*

The class $\mathrm{Dom}(\mathcal{A})$ is important as every property about the automata $\mathcal{A}$ will depend on $\mathrm{Dom}(\mathcal{A})$ rather than on its class. Note that we use in the definition the closure of the sets rather than the sets themselves. This is done so that the various topological spaces preserve some properties : if $\mathcal{C}$ is compact (resp. complete), this is also the case of $\hat{\mathcal{C}}$. Note also that for a purged automaton, the set $\{X_u, u \in \Sigma^\star\}$ is by construction dense in $\hat{\mathcal{O}}$. This fact will be useful later.

2.2 Recognized Languages

We may now define numerous definitions for recognizing languages , based on the usual definitions for probabilistic and quantum automata :

Definition 5. *Let* $\mathcal{L}$ *be a language and* $\mathcal{A}$ *be an automaton.*

- $\mathcal{L}$ *is said to be recognized by* $\mathcal{A}$ *with strict threshold* λ *if* $\mathcal{L}$ *is the set of words accepted with a value strictly greater than* λ *:* $\omega \in \mathcal{L} \iff Val_{\mathcal{A}}(\omega) > \lambda$*.*
- $\mathcal{L}$ *is said to be recognized by* $\mathcal{A}$ *with threshold* λ *if* $\mathcal{L}$ *is the set of words accepted with a value greater than* λ *:* $\omega \in \mathcal{L} \iff Val_{\mathcal{A}}(\omega) \geq \lambda$*.*
- $\mathcal{L}$ *is said to be recognized by separation by* $\mathcal{A}$ *if there exists two disjoints closed sets* A *and* $A^\top$ *such that*

  - $\omega \in \mathcal{L} \implies Val_{\mathcal{A}}(\omega) \in A$,
  - $\omega \notin \mathcal{L} \implies Val_{\mathcal{A}}(\omega) \in A^{\top}$,
- *If the sharper condition* $\forall x \in A \forall y \in A^{\top}, |x-y| > \epsilon$ *is verified for some* $\epsilon > 0$*, then* $\mathcal{L}$ *is said to be recognized by* $\mathcal{A}$ *with (bounded) error* ϵ*.*

The very definition of a class of automata permits us to state some closure properties for these languages.

Lemma 1. *Let* $\mathcal{A}$ *be an automaton of a class* $\mathfrak{C}$*. Then*

- *For all words* $u \in \Sigma^{\star}$*, there exists an automaton* $\mathcal{B}$ *of the class* $\mathfrak{C}$ *such that* $Val_{\mathcal{B}}(\omega) = Val_{\mathcal{A}}(u\omega)$
- *For all words* $v \in \Sigma^{\star}$*, there exists an automaton* $\mathcal{B}$ *of the class* $\mathfrak{C}$ *such that* $Val_{\mathcal{B}}(\omega) = Val_{\mathcal{A}}(\omega v)$
- *For every alphabet* Δ *and every morphism* h *from* $\Delta^{\star}$ *to* $\Sigma^{\star}$*, there exists an automaton* $\mathcal{B}$ *of* $\mathfrak{C}$ *such that* $Val_{\mathcal{B}}(\omega) = Val_{\mathcal{A}}(h(\omega))$

Proof. Write $\mathcal{A} = (c_0, (X_i)_{i\in\Sigma}, m)$ and consider the automata

- $\mathcal{B} = (c_0 \cdot X_u, (X_i)_{i\in\Sigma}, m)$,
- $\mathcal{B} = (c_0, (X_i)_{i\in\Sigma}, m')$ with $m'(x) = m(x \cdot X_v)$
- $\mathcal{B} = (c_0, (Y_i)_{i\in\Delta}, m)$ with $Y_i = X_{h(i)}$. □

Corollary 1. *The class of languages recognized (with one of the previous definitions) by a class of automata is closed by quotients and inverse morphism.*

Furthermore, it is trivial to see that the class of languages recognized by separation or separation with error ϵ is closed by complementation.

2.3 Regular Languages

We will now give sufficient conditions for a language accepted by a topological automaton to be regular. The separating condition will be in this context precious. We have indeed the first theorem :

Theorem 2. *Let* $\mathcal{L}$ *be a language recognized by separation by an automaton* $\mathcal{A} = (c_0, (X_i)_i, m)$*. If* $\overline{\{X_i, i \in \Sigma^{\star}\}}$ *has finitely many connected components then* $\mathcal{L}$ *is regular.*

$\overline{\{X_i, i \in \Sigma^{\star}\}}$ stands for the closure of $\{X_i, i \in \Sigma^{\star}\}$.

Proof. We can always change $\mathcal{A}$ into $\hat{\mathcal{A}}$. In this context,

- $\{X_i, i \in \Sigma^{\star}\}$ is dense in $\mathcal{O}$, so that for all $o \in \mathcal{O}, m(c_0 \cdot o) \in A \cup A^{\top}$.
- $\mathcal{O}$ has finitely many connected components.

We will show that L has finitely many left quotients $u^{-1}L = \{v | uv \in L\}$.

For every $X \in \mathcal{O}$, set $M(X) = \{o | m(c_0 \cdot X \cdot o) \in A\}$. It is easy to see that if $M(X_u) = M(X_w)$ then $u^{-1}L = w^{-1}L$. It is then sufficient to prove that $M(X)$ can take only finitely many values.

Now, $M(X)$ is closed as the preimage of a closed set by a continuous map. However, $M(X)$ is also open as its complementary is by separation the set $\{o|m(c_0 \cdot X \cdot o) \notin A\} = \{o|m(c_0 \cdot X \cdot o) \in A^\top\}$. The set $M(X)$ is then a clopen set, hence a union of connected components. As there are finitely many components, this concludes the proof. □

As any compact group has finitely many components, this gives an alternative proof of the fact that Measure-Once quantum automata recognize only regular languages with an isolated threshold.

However, the following theorem will be in practice more useful

Theorem 3. *Let $\mathcal{L}$ be a language recognized by separation by an automaton $\mathcal{A} = (c_0, (X_i)_i, m)$. If $\overline{\{X_i, i \in \Sigma^\star\}}$ is compact (that is the monoid generated by the X_i is relatively compact), then $\mathcal{L}$ is regular.*

Proof. For compact sets, separation is the same as separation with some error ϵ. We will use this alternate formulation.

We might use the same ideas, but we will here rather proceed differently to capture precisely the role of ϵ. This will give bounds in the next theorem.

We will use the Myhill-Nerode theorem[6] and show that the relation

$$uRv \iff (\forall t, ut \in \mathcal{L} \iff vt \in \mathcal{L})$$

has finitely many classes.

We again change $\mathcal{A}$ into $\hat{\mathcal{A}}$. The hypothesis then implies that $\mathcal{C}$ and $\mathcal{O}$ are compact.

Now consider the family of functions from $\mathcal{C}$ to $\mathbb{R}$ $\{f_x : c \mapsto m(c \cdot x), x \in \mathcal{O}\}$. As $\mathcal{O}$ is compact, this family is equicontinuous, and even uniformly equicontinuous, as $\mathcal{C}$ is compact : For every $\epsilon > 0$ there exists δ such that the property $\forall c, c' \in \mathcal{C}, \forall x \in \mathcal{O}, d(c, c') \leq \delta \implies |m(c \cdot x) - m(c' \cdot x)| \leq \epsilon$ holds.

We use this assertion with the ϵ given by the fact that $\mathcal{L}$ is recognized in an isolated way.

Now, let u et v be two words such that $u \not R\ v$. There exists t such that $ut \in \mathcal{L}$, and $vt \notin \mathcal{L}$, or conversely. Then $|m(c_0 \cdot X_{ut}) - m(c_0 \cdot X_{vt})| > \epsilon$, that is $|m((c_0 \cdot X_u) \cdot X_t) - m((c_0 \cdot X_v) \cdot X_t)| > \epsilon$. Hence we have $d(c_0 \cdot X_u, c_0 \cdot X_v) > \delta$.

We now just have obtained the following result : If u et v are not in the same class, then $d(c_0 \cdot X_u, c_0 \cdot X_v) > \delta$.

Now, by compactness, we cover $\mathcal{C}$ with finitely many balls of radius $\delta/2$. Then if u et v are such that $c_0 \cdot X_u$ and $c_0 \cdot X_v$ are in the same ball, they must be in the same class. Therefore there are finitely many classes. □

A closer look at the proof shows that we can have a somehow stronger result:

Theorem 4. *Let $(\mathcal{O}, \mathcal{C}, \mathcal{M})$ be a class of automata. If $\mathcal{O}$ and $\mathcal{C}$ are compact and $\mathcal{M}$ is equicontinuous, then for every ϵ there exists a p such that any language accepted by separation with error ϵ by an automaton of this class is regular and its minimal automaton has less than p states.*

Indeed, we now have that the family of functions from $\mathcal{C}$ to $\mathbb{R}$ given by $\{c \mapsto m(c \centerdot x) \ : \ x \in \mathcal{C}, m \in \mathcal{M}\}$ is uniformly equicontinuous; given δ, we can thus set p to be the cardinal of a minimal covering of $\mathcal{C}$ by balls of radius $\delta/2$.

This stronger property is verified by classical, probabilistic and quantum automata with n states. We can furthermore show that we can take $p(n, \epsilon) = n$ for classical automata, $p(n, \epsilon) = \lceil (2/\epsilon)^n \rceil$ for probabilistic automata. A similar bound has been obtained for quantum automata [7].

We will give here an example to explain why the hypotheses are necessary.

Take the class of automata given by $\mathcal{O} = \mathbb{R}$, where the multiplication operator $\circ$ is the usual addition over $\mathbb{R}$, $\mathcal{C} = \mathbb{R}$ where $\mathcal{O}$ acts naturally over $\mathcal{C}$ by addition, and the set of measures is the set of continuous maps from $\mathbb{R}$ to $\mathbb{R}$. Now, let $\mathcal{A}$ be the automaton defined by $X_a = 1, X_b = -1$, $c_0 = 0$, $m : x \mapsto x$. Note that $\{X_i\}^\star = \mathbb{Z}$ is not relatively compact and has infinitely many connected components.

Obviously, $Val_{\mathcal{A}}(\omega) = |\omega|_a - |\omega|_b$, where for $i \in \Sigma$, $|\omega|_i$ is the number of occurrences of i in ω. $\mathcal{L} = \{\omega : |\omega|_a = |\omega|_b\}$ is therefore recognized by separation by $\mathcal{A}$ but is not regular.

3 Non-deterministic Automata

We will see how non-deterministic automata fit in our framework, and how the proof that the recognized languages are regular can be easily obtained by means of the Hausdorff metric on compact sets.

3.1 Definition

As it is the case for classical finite automata, we can now define many notions of non-determinism :

- We may add so-called $\mathcal{E}$-transitions.
- Instead of having just one operator per letter, we may have several ones.

We will show that adding $\mathcal{E}$-transitions still produces regular languages. This is still true for the second notion of non-determinism, and the proof is very similar to the proof given here.

Let us define precisely this notion.

Definition 6. *A non-deterministic automaton $\mathcal{A}$ over Σ of a class of automata is given by a deterministic automaton $\mathcal{B}$ over $\Sigma \cup \{\mathcal{E}\}$ where the value of a word $\omega \in \Sigma$ is now defined as :*

$Val_{\mathcal{A}}(\omega) = \sup_{h(u)=\omega} Val_{\mathcal{B}}(u)$

where h is the morphism that maps $x \in \Sigma$ to x and $\mathcal{E}$ to the empty word ϵ.

We can express it intuitively in the following way : the automaton acts as a normal automaton, except $\mathcal{E}$-transition can occur at every step; The value of a word is then the supremum of all values we can obtain.

We can also define the value in the following manner :

Proposition 1. *Define recursively*

- $K_\epsilon = c_0 \centerdot \{X_\mathcal{E}\}^\star = \left\{c_0 \centerdot x : x \in \{X_\mathcal{E}\}^\star\right\}$
- *for* $i \in \Sigma$, $K_{vi} = K_v \centerdot X_i \centerdot \{X_\mathcal{E}\}^\star = \left\{c \centerdot X_i \centerdot x : x \in \{X_\mathcal{E}\}^\star, c \in K_v\right\}$

Then $Val_\mathcal{A}(\omega) = \sup m(K_\omega)$.

It can easily be shown that we can replace $\{X_\mathcal{E}\}^\star$ by $\overline{\{X_\mathcal{E}\}^\star}$. Indeed, if we call J_ω the sets obtained this way, then $K_\omega \subseteq J_\omega \subseteq \overline{K_\omega}$, so that $\sup m(K_\omega) = \sup m(J_\omega)$.

We will now prove that if $\{X_i\}^\star_{i\in\Sigma\cup\{\mathcal{E}\}}$ is relatively compact and $\mathcal{L}$ recognized by separation, then $\mathcal{L}$ is regular.

Please note that, as the language over $\Sigma \cup \{\mathcal{E}\}$ is not necessarily separated, we can not use directly the results from the previous section.

We will in fact determinize the automaton. For classical automata, this involves sets of configurations. The natural, topological equivalent will be compact sets of configurations.

3.2 Hausdorff Metric

Unless specified, all compact sets will be taken non-empty.

There exists a natural topology on compact sets of a metric space, given by the Hausdorff metric :

Definition 7. *Given a metric space* (E, d)*, and two non-empty compact sets* K_1 *and* K_2 *of* E*, the Hausdorff distance is defined by*

$$d_H(K_1, K_2) = \max_{x_1,x_2\in K_1\times K_2} \max\{d(x_1, K_2), d(x_2, K_1)\}$$

where $d(x, K)$ *is the distance of* x *from* K *:* $d(x, K) = min_{y\in K} d(x, y)$

d_H is a metric over the set of non-empty compact sets of E, denoted by $\mathfrak{E}$. This space is called the Hausdorff metric space induced by E.

This metric extends naturally the metric d. In fact, we can "lift" the functions :

Proposition 2. *Let* (E, d) *and* (F, d') *be two metric spaces, and* $(\mathfrak{E}, d_H)$, $(\mathfrak{F}, d'_H)$ *their induced Hausdorff metric spaces.*

Let $f : E \to F$ *be a continuous map.*

Then $\mathfrak{f} : \begin{array}{l} \mathfrak{E} \to \mathfrak{F} \\ K \mapsto f(K) \end{array}$ *is continuous.*

Similar results of this kind will be used in the following section. We will only prove this fact, but the proof contains in substance all the other results.

Proof. First note that $\mathfrak{f}$ takes values in $\mathfrak{F}$, so that $\mathfrak{f}$ is well defined.

Let K be a compact of E and $\epsilon > 0$.

For every $x \in K$, there exists $\delta_x > 0$ such that if $y \in E$ and $d(x, y) \leq \delta_x$, then $d'(f(x), f(y)) \leq \epsilon$. Covering K with balls of center x and radius $\delta_x/2$, we

see by compactness that there exists $\delta > 0$ such that if $x \in K$ and $y \in E$ verify $d(x, y) \leq \delta$, then $d'(f(x), f(y)) \leq 2\epsilon$ (Please note that we have obtained $y \in E$ and not $y \in K$).

Now, let N be a compact such that $d_H(N, K) \leq \delta$.

Let $x \in N$. There exists $y \in K$ such that $d(x, y) \leq \delta$, so that we obtain $d'(f(x), f(y)) \leq 2\epsilon$. Then $d'(x, f(K)) \leq 2\epsilon$.

Let $y \in K$. There exists $n \in N$ such that $d(x, y) \leq \delta$, so that we have $d'(f(x), f(y)) \leq 2\epsilon$. Then $d'(y, f(N)) \leq 2\epsilon$.

Hence $d'_H(f(K), f(N)) \leq 2\epsilon$.

We have shown that for every $\epsilon > 0$, there exists $\delta > 0$ such that for every N such that $d_H(N, K) \leq \delta$, we have $d'_H(f(K), f(N)) \leq 2\epsilon$. □

Now, the next important result we need is the following, whose proof can be found in [8].

Theorem 5. *The set of compact sets of a compact metric space is compact with respect to the Hausdorff metric.*

3.3 Determinization

We now have all we need to determinize our automata.

Given a class $\theta = (\mathcal{C}, \mathcal{O}, \mathcal{M})$ of automata, we define the class of θ-determinized automata by :

- $\mathcal{C}'$ are the compact sets of $\mathcal{C}$ with the Hausdorff metric.
- $\mathcal{O}'$ are the compact sets of $\mathcal{O}$ with the Hausdorff metric. $\mathcal{O}'$ is a monoid for the multiplication $X \circ Y = \{x \circ y, x \in X, y \in Y\}$ which is well defined. $\mathcal{O}'$ acts naturally over $\mathcal{C}' : C \centerdot X = \{c \centerdot x, c \in C, x \in X\}$. Using variants of proposition 2, we can show easily that the multiplication and the action are continuous maps.
- $\mathcal{M}'$ is the set of functions $C \mapsto \sup_{c \in C} m(c)$ for $m \in \mathcal{M}$. Again, we may prove that these functions are continuous by similar arguments.

Given a non-deterministic automaton $\mathcal{A} = \{c_0, \{X_i\}_{i \in \Sigma}, X_{\mathcal{E}}, m)$ of the class θ, where $K = \overline{\{X_i, i \in \{\mathcal{E}\}^\star\}}$ is compact, we define the deterministic automaton $\mathcal{B}$ of the class of θ-determinized automata by

- the initial configuration is $C_0 = c_0 \centerdot K = \{c \centerdot k, k \in K\}$
- the operators are $X'_i = X_i \circ K = \{X_i \circ k, k \in K\}$
- the measure is now $m' : C \mapsto \sup_{c \in C} m(c)$

Proposition 3. *For any word ω, $Val_{\mathcal{A}}(\omega) = Val_{\mathcal{B}}(\omega)$.*

Proof. This is obvious using the previous definition of $\mathcal{B}$ and the proposition 1. □

We denote by $\Sigma_{\mathcal{E}}$ the set $\Sigma \cup \{\mathcal{E}\}$. We can now conclude :

Theorem 6. *Let $\mathcal{L}$ be a language recognized in an isolated way by a non-deterministic automaton $\mathcal{A} = (c_0, (X_i)_i, X_{\mathcal{E}}, m)$.*

If $\overline{\{X_i, i \in \Sigma_{\mathcal{E}}^\star\}}$ is compact, then $\mathcal{L}$ is regular.

Proof. As $K = \overline{\{X_i, i \in \{\mathcal{E}\}^\star\}}$ is compact and stable by multiplication, we can define $\mathcal{B}$, the deterministic automata corresponding to $\mathcal{A}$.

Now, it suffices to show that $M = \{X'_i, i \in \Sigma^\star\}$ is relatively compact.

Let note $E = \overline{\{X_i, i \in \Sigma^\star_{\mathcal{E}}\}}$.

Using theorem 5, we see that $\mathfrak{E}$, the set of compacts of E is compact with respect to the Hausdorff metric. Every element of M is in $\mathfrak{E}$, so that M is a subset of $\mathfrak{E}$, hence $\overline{M}$ is compact. □

We have in particular proven that non-deterministic probabilistic and quantum automata accept only regular languages when limited to isolated acceptance. These results are new for quantum automata; This also gives an alternative to the proof from [5] for probabilistic automata.

Finally note that if we can cover a compact C with n balls of radius δ, it is easy to show that we can cover the set of non-empty compact sets of C with $2^n - 1$ balls of radius δ. We can then obtain a constructive proof of the statement. We have proven for deterministic automata of a certain class that the number of states is bounded by the number of balls of the set of configurations of radius δ, where δ depends on ϵ. A detailed analysis of the previous proof shows that in fact we have the same dependency over ϵ. That is : the number of states for languages recognized by non-deterministic topological automata is bounded by the number of balls of $\mathcal{C}'$ of radius δ for the same δ. This is summed up in the following theorem:

Theorem 7. *Let $(\mathcal{O}, \mathcal{C}, \mathcal{M})$ be a class of automata. If $\mathcal{O}$ and $\mathcal{C}$ are compact and $\mathcal{M}$ is equicontinuous, then for every ϵ there exists a p such that*

- *any language accepted by separation with error ϵ by an automaton of this class is regular and its minimal automaton has less than p states.*
- *any language accepted by separation with error ϵ by a non-deterministic automaton of this class is regular and its minimal automaton has less than $2^p - 1$ states.*

Furthermore we can take $p(n, \epsilon) = n$ for classical automata, $p(n, \epsilon) = \lceil (2/\epsilon)^n \rceil$ for probabilistic automata, where n is the number of internal states.

It would be interesting to simulate non-deterministic probabilistic or quantum automata by their deterministic counterpart, and show how efficient (in term of states) this simulation is. However, languages accepted by non-deterministic quantum automata cannot always be accepted by deterministic quantum automata, so that this simulation is not always possible.

4 Conclusion

We have given a new definition of automata which extends the notions of classical, probabilistic or quantum automata, and have shown how some properties of the languages they accept are just a consequence of topological properties mainly compactness. This framework also gives us a way to deal with non-deterministic automata, for which we have proven that they accept only regular languages by

separation provided some compactness condition for the set of operators. This gives an alternative simpler proof of the fact that non-deterministic probabilistic automata recognize only regular languages by separation.

It would be interesting to see how other topological concepts fit in this framework and how some may correspond to other classes of languages. A good candidate is locally compact spaces. However there are not that many classes of languages which have as good characterizations as regular languages : A characterization in terms of machinery is not easy to use in the context of topological spaces.

References

1. Rabin, M.O.: Probabilistic automata. Information and Control **6** (1963)
2. Moore, C., Crutchfield, J.: Quantum automata and quantum grammars. Theoretical Computer Science **237** (2002) 257–306
3. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: IEEE Symposium on Foundations of Computer Science. (1997) 66–75
4. Bozapalidis, S.: Extending stochastic and quantum functions. Theory of Computing Systems **36** (2003) 183–197
5. Condon, A., Hellerstein, L., Pottle, S., Wigderson, A.: On the power of finite automata with both nondeterministic and probabilistic states. SIAM Journal on Computing **27** (1998) 739–762
6. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading, Mass. (1979)
7. Ablayev, F., Gainutdinova, A.: On the Lower Bounds for One-Way Quantum Automata. In Nielsen, M., Rovan, B., eds.: Proceedings MFCS 2000. (2000) 132–140
8. Kuratowski, K.: Topology, Vol. II, 3rd edition. NY: Academic Press (1966)

Minimizing NFA's and Regular Expressions

Gregor Gramlich and Georg Schnitger*

Institut für Informatik, Johann Wolfgang Goethe–Universität Frankfurt,
Robert-Mayer-Straße 11-15, 60054 Frankfurt am Main, Germany
{gramlich, georg}@thi.informatik.uni-frankfurt.de
Fax: +49 - 69 - 798-28814

Abstract. We show inapproximability results concerning minimization of nondeterministic finite automata (nfa's) as well as regular expressions relative to given nfa's, regular expressions or deterministic finite automata (dfa's). We show that it is impossible to efficiently minimize a given nfa or regular expression with n states, transitions, resp. symbols within the factor $o(n)$, unless $P = \text{PSPACE}$. Our inapproximability results for a given dfa with n states are based on cryptographic assumptions and we show that any efficient algorithm will have an approximation factor of at least $\frac{n}{\text{poly}(\log n)}$. Our setup also allows us to analyze the minimum consistent dfa problem.

Classification: Automata and Formal Languages, Computational Complexity, Approximability.

1 Introduction

Among the most basic objects of formal language theory are regular languages and their acceptance devices, finite automata and regular expressions. Regular expressions describe lexical tokens for syntactic specifications, textual patterns in text manipulation systems and they are the basis of standard utilities such as scanner generators, editors or programming languages (perl, awk, php). Internally regular expressions are converted to (nondeterministic) finite automata and the succinctness of this representation crucially determines the running time of the applied algorithms.

Contrary to the problem of minimizing dfa's, which is efficiently possible, it is well known that nfa or regular expression minimization is computationally hard, namely PSPACE-complete [10]. Jiang and Ravikumar [7] show moreover that the minimization problem for nfa's or regular expressions remains PSPACE-complete, even when specifying the regular language by a dfa.

We consider the problem of *approximating* a minimal nfa or a minimal regular expression. In [3] it is shown that unary nfa's are hard to approximate and in particular efficient approximation algorithms require an approximation

* Partially supported by DFG project SCHN503/2-1.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 399–411, 2005.

factor of at least $\frac{\sqrt{n}}{\ln n}$ for given nfa's or regular expressions of size n, provided $P \neq NP$. On the other hand, there are several approaches to nfa minimization [1, 4, 5, 9] without approximation guarantees or running in at least exponential time. This article explains why such guarantees cannot be expected for efficient algorithms.

We investigate the approximation problem in two scenarios. In the first scenario the language is specified by a dfa which makes proofs of inapproximability hard, since the input is not specified concisely and thus more time compared to concise inputs such as nfa's or regular expressions is available. Jiang and Ravikumar [7] ask to determine the approximation complexity of converting dfa's into nfa's, and in particular ask whether efficient approximation algorithms with a polynomial approximation factor exist. Corollary 1 shows that such an approximation is at least as hard as factoring Blum integers and therefore efficient approximation algorithms with polynomial approximation factor are unlikely.

We show in Theorem 1 that efficient approximation algorithms determine regular expressions of length at least $\frac{k}{\text{poly}(\log k)}$ for a given dfa of size k, even if optimal regular expressions of length $\text{poly}(\log k)$ exist. We have to assume however that strong pseudo-random functions exist in non-uniform NC^1. The concept of a strong pseudo-random function is introduced by Razborov and Rudich [14]. Naor and Reingold [11] show that strong pseudo-random functions exist even in TC^0, provided factoring Blum integers requires time $2^{\Omega(n^\varepsilon)}$ (for some $\varepsilon > 0$).

We show similar results for approximating nfa's in Corollary 1, but now relative to the assumption that strong pseudo-random functions exist in non-uniform Logspace. We also apply our technique to the minimum consistent dfa problem [8, 12] in which a dfa of minimum size, consistent with a set of classified inputs, is to be determined.

Thus in the first scenario we follow the cryptographic approach of Kearns and Valiant [8] when analyzing the complexity of approximation, but work with pseudo-random functions instead of one-way functions. In the second scenario we assume that the language is specified by either an nfa or a regular expression. For the *unary* case we improve in Theorem 3 the approximation factor from $\frac{\sqrt{n}}{\ln n}$ [3] to $n^{1-\delta}$ for every $\delta > 0$, provided $P \neq NP$ and provided we require the approximation algorithm to determine a small equivalent nfa or regular expression, opposed to just determining the number of states.

Furthermore we show a PSPACE-completeness result for approximating the minimal size of *general* nfa's or regular expressions. Specifically Theorem 4 shows that it is impossible to efficiently minimize a given nfa or regular expression with n states, n transitions resp. n symbols within the factor $o(n)$, unless $P =$ PSPACE. The proof of Theorem 4 is based on the PSPACE-completeness of the "regular expression non-universality" problem.

We introduce strong pseudo-random functions in section 2 and investigate the complexity of approximating minimal regular expressions or nfa's, relative to a given dfa, in subsections 2.1 and 2.2. The minimum consistent dfa problem is

considered in subsection 2.3. Finally the complexity of approximately minimizing unary resp. general nfa's or regular expressions, relative to a given nfa or regular expression, is determined in section 3.

2 Pseudo-Random Functions and Approximation

We consider the question of computing small equivalent nfa's or regular expressions for given dfa's. Inapproximability results seem to be hard to prove, since, intuitively, it takes large dfa's to specify hard inputs and consequently the allowed running time increases. Therefore we investigate the approximation complexity for minimum nfa's or regular expressions when given the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ and utilize the natural proof setup of Razborov and Rudich [14]. In particular, we utilize the concept of strong pseudo-random functions, but replace circuits by probabilistic Turing machines and require only a constant probability of separating pseudo-randomness from true randomness. Obviously strong pseudo-random functions exist in our setting, provided strong pseudo-random functions exist in the sense of Razborov and Rudich.

Definition 1. *Let $f_n = (f_n^s)_{s\in S}$ be a function ensemble with functions $f_n^s : \{0,1\}^n \to \{0,1\}$ for a seed $s \in S$ and let $(r_n^i)_{i\in\{1,\ldots,2^{2^n}\}}$ be the ensemble of all n-bit boolean functions. We call f_n a strong pseudo-random ensemble with parameter ε iff for any randomized algorithm A*

$$|\text{prob}[A(f_n) = 1] - \text{prob}[A(r_n) = 1]| < \frac{1}{3},$$

provided A runs in time $2^{O(n^\varepsilon)}$ and has access to f_n^s, resp. r_n^i, via a membership oracle. The probability is defined by the random choices of A and the uniform sampling of s from S, resp. the uniform sampling of i from $\{1,\ldots,2^{2^n}\}$.

It is widely believed that there is some $\varepsilon > 0$, such that any algorithm running in time $2^{O(n^\varepsilon)}$ cannot factor Blum integers well on average. Naor and Reingold [11] construct TC^0 functions which are strong pseudo-random functions, provided factoring Blum integers requires time $2^{\Omega(n^\varepsilon)}$ for some ε.

Definition 2. *B_n is the set of all n-bit boolean functions. We define the compression $k_m : B_n \to B_m$ for $m < n$ by $(k_m(f))(x) = f(0^{n-m}x)$ for $x \in \{0,1\}^m$.*

We say, that a functional $G = (G_n)_n$ with $G_n : B_n \to \mathbb{N}$ separates a function class $\mathcal{C}$ from random functions with thresholds $t_1(\cdot)$ and $t_2(\cdot)$ iff $G_n(f) < t_1(n)$ holds for every function $f \in \mathcal{C} \cap B_n$, whereas $G_n(\rho) > t_2(n)$ for most functions in B_n, i.e., $|\{\rho \in B_n | G_n(\rho) \le t_2(n)\}| = o(|B_n|)$ holds. Moreover we require that $G_m(k_m(f)) \le t_1(n) \cdot \text{poly}(n)$ for any function $f \in \mathcal{C} \cap B_n$ and any $m < n$.

It is not surprising that a functional G, which separates a function class $\mathcal{C}$ containing pseudo-random functions from random functions, cannot be efficiently approximated. We allow randomized approximation algorithms which may even underestimate the minimum.

Definition 3. *Let $|x|$ be the length of input x. We say that a randomized algorithm $App : X \to \mathbb{N}$ with approximation factor $\mu(|x|)$ for a minimization problem opt has overestimation error $\epsilon_+ = \sup_{x \in X} \text{prob}[App(x) > \mu(|x|) \cdot opt(x)]$ and underestimation error $\epsilon_- = \sup_{x \in X} \text{prob}[App(x) < opt(x)]$. The probabilities are defined by the random choices of App.*

We state a generic lemma for approximation algorithms on compressed inputs allowing us to replace oracle access by truth table presentation.

Lemma 1. *Assume that the functional G separates $\mathcal{C}$ from random functions with thresholds t_1, t_2 and suppose that $\mathcal{C}$ contains a strong pseudo-random ensemble with parameter ε.*

Let App be a randomized approximation algorithm that approximately determines $G_m(h_m)$, when given the truth table of a function $h_m \in B_m$. Then for all $l \geq 1$, if App runs in time $2^{O(m^l)}$ with errors $\epsilon_+ + \epsilon_- < \frac{2}{3}$, then App can only achieve an approximation factor $\mu_m \geq \frac{t_2(m)}{t_1(m^{l/\varepsilon}) \text{poly}(m^{l/\varepsilon})}$.

Proof. By assumption $\mathcal{C}$ contains strong pseudo-random functions with parameter ε. Let *App* be an algorithm which approximates $G_m(f_m)$ when given the truth table of f_m (with running time $2^{O(m^l)}$ for some $l \geq 1$, approximation factor $1 \leq \mu_m < \frac{t_2(m)}{t_1(m^{l/\varepsilon}) \text{poly}(m^{l/\varepsilon})}$ and errors $\epsilon_+ + \epsilon_- < \frac{2}{3}$). We construct an algorithm A which uses *App* to distinguish n-bit functions in $\mathcal{C}$ from n-bit random functions. We set $m = \lfloor n^{\varepsilon/l} \rfloor$.

A has oracle access to the input $h_n \in B_n$ and builds the truth table for the restriction $k_m(h_n)$. Then A runs *App* on $k_m(h_n)$ and accepts (i.e. $A(h_n) = 1$), if $App(k_m(h_n)) \leq t_2(m)$, and rejects (i.e. $A(h_n) = 0$) otherwise. So $|\text{prob}[A(f_n) = 1] - \text{prob}[A(r_n) = 1]| = |\text{prob}[App(k_m(f_n)) \leq t_2(m)] - \text{prob}[App(k_m(r_n)) \leq t_2(m)]|$ holds, where probabilities are taken over the probabilistic choices of *App* as well as the random sampling of seeds for f_n, respectively the uniform random sampling of functions $r_n \in B_n$.

G separates $\mathcal{C}$ from random functions and hence we have $G_m(k_m(f_n)) \leq t_1(n) \cdot \text{poly}(n)$ for $f_n \subseteq \mathcal{C}$. Finally observe that $\mu_m \cdot t_1(n) \cdot \text{poly}(n) < t_2(m)$ holds by assumption on μ_m.

$$\begin{aligned} \text{prob}[App(k_m(f_n)) \leq t_2(m)] &\geq \text{prob}[App(k_m(f_n)) \leq \mu_m \cdot t_1(n) \cdot \text{poly}(n)] \\ &= 1 - \text{prob}[App(k_m(f_n)) > \mu_m \cdot t_1(n) \cdot \text{poly}(n)] \\ &\geq 1 - \text{prob}[App(k_m(f_n)) > \mu_m \cdot G_m(k_m(f_n))] \end{aligned}$$

and $\text{prob}[App(k_m(f_n)) \leq t_2(m)] \geq 1 - \epsilon_+$ follows. We utilize that a uniformly sampled function r_n from B_n leads to a uniformly sampled restriction from B_m.

$$\begin{aligned} \text{prob}[App(k_m(r_n)) \leq t_2(m)] &\leq \text{prob}[G_m(k_m(r_n)) \leq t_2(m)] + \epsilon_- \\ &= \frac{|\{\rho_m | G_m(\rho_m) \leq t_2(m)\}|}{|B_m|} + \epsilon_- = \epsilon_- + o(1). \end{aligned}$$

Thus $|\text{prob}[App(k_m(f_n)) \leq t_2(m)] - \text{prob}[App(k_m(r_n)) \leq t_2(m)]| \geq 1 - \epsilon_+ - \epsilon_- - o(1) > \frac{1}{3}$ holds for sufficiently large m. Since A runs in time $O(2^m) + 2^{O(m^l)} = 2^{O(n^\varepsilon)}$ this contradicts the assumption that $\mathcal{C}$ contains a strong pseudo-random ensemble with parameter ε. □

In our first applications of Lemma 1, $G(f)$ will be the minimum length of regular expressions, respectively the minimum size of nfa's that accept, for some T, the complement of

$$L_T(f) = \{x^T | f(x) = 1\}.$$

2.1 Regular Expressions and Logarithmic Formula Depth

Definition 4. *A formula is a binary tree with $\wedge$ and $\vee$ gates as interior nodes; leaves are marked by labels from $\{x_1, \overline{x_1}, \ldots, x_i, \overline{x_i}, \ldots\}$. For a formula $\mathbf{f}$ let $\ell(\mathbf{f})$ be the length, i.e., the number of leaves of $\mathbf{f}$. The length $\ell(R)$ of a regular expression R is the number of symbols from the alphabet Σ appearing in R. The rpn-length of a regular expression R is the number of symbols from $\Sigma \cup \{+, \circ, ^*, \varepsilon, \emptyset\}$ appearing in R, when R is written in reverse Polish notation.*

Naor and Reingold [11] show that NC^1 contains a strong pseudo-random ensemble for some parameter $\varepsilon > 0$, provided factoring Blum integers is sufficiently hard. More precisely there is some constant c and a hard pseudo-random ensemble $\mathcal{C}_1$ with formula depth at most $c \cdot \log m$ for functions in $\mathcal{C}_1 \cap B_m$. Thus all functions in $\mathcal{C}_1 \cap B_m$ have formula length at most $T_1(m) = m^c$.

We define the functional $G^{(1)}$ by setting $G^{(1)}_m(f_m)$ to equal the minimum length of a regular expression for the complement of $L_{T_1}(f_m) = \{x^{T_1} | f_m(x) = 1\}$.

We associate regular expressions with formulae and show that the length of the regular expression is exponentially related to the depth of the formula.

Definition 5. *Let $\mathbf{f}$ be a formula for a function $f : \{0,1\}^m \to \{0,1\}$. We define the regular expression $R(\mathbf{f})$ recursively as follows:*

- *If $\mathbf{f} = x_i$, then $R(\mathbf{f}) := (0+1)^{i-1}\ 1\ (0+1)^{m-i}$.*
- *If $\mathbf{f} = \overline{x_i}$, then $R(\mathbf{f}) := (0+1)^{i-1}\ 0\ (0+1)^{m-i}$.*
- *If $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$, then $R(\mathbf{f}) := R(\mathbf{f}_1) \circ R(\mathbf{f}_2)$.*
- *If $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$, then $R(\mathbf{f}) := R(\mathbf{f}_1) \circ (0+1)^{m \cdot \ell(\mathbf{f}_2)} + (0+1)^{m \cdot \ell(\mathbf{f}_1)} \circ R(\mathbf{f}_2)$.*

Lemma 2. *Let $W = \{w | \exists x \in \{0,1\}^m \wedge w \in \{x\}^*\}$ be the language of repeated inputs of length m.*

(a) $L(R(\mathbf{f})) \cap W = \{x^{\ell(\mathbf{f})} | f(x) = 1\} = L_{\ell(\mathbf{f})}(f)$.

(b) For a given formula $\mathbf{f}$ of depth k there is a regular expression $\overline{R_\mathbf{f}}$ which describes the complement of $L(R(\mathbf{f})) \cap W$. $\overline{R_\mathbf{f}}$ has length $O(4^k m)$ and can be constructed in time $\text{poly}(4^k m)$.

(c) In particular, $\overline{L_{T_1}(f_m)}$ has regular expressions of length $t_1^{(1)} = O(m^{2c+1})$ for any $f_m \in \mathcal{C}_1 \cap B_m$.

Proof. (a) can be shown by an induction on the structure of formula $\mathbf{f}$.

(b) Moreover, such an induction shows that for a given formula $\mathbf{f}$ of depth k the regular expression $R(\mathbf{f})$ has length at most $2 \cdot 4^k m$ and can be constructed in time $\text{poly}(4^k m)$. Observe that $\overline{L(R(\mathbf{f})) \cap W} = \overline{L(R(\mathbf{f}))} \cup \overline{W}$. We check whether the input does not consist of repetitions with the regular expression

$$\big((0+1)^*\ 1\ (0+1)^{m-1}\ 0\ (0+1)^*\big)\ +\ \big((0+1)^*\ 0\ (0+1)^{m-1}\ 1\ (0+1)^*\big)$$

and cover words of wrong length by $(0+1+\varepsilon)^{m\cdot\ell(\mathbf{f})-1}\ +\ (0+1)^{m\cdot\ell(\mathbf{f})+1}(0+1)^*$. We negate $\mathbf{f}$ with DeMorgan and observe that depth does not increase.

(c) Since all functions in $\mathcal{C}_1 \cap B_m$ have formula depth at most $c \cdot \log m$, we may assume that all these functions have formulae of depth exactly $c \cdot \log m$ and length exactly $T_1(m) = m^c$. Thus with part (a) $L_{T_1}(f_m)$ coincides with $L(R(\mathbf{f})) \cap W$ and, with part (b), $\overline{L_{T_1}(f_m)}$ has regular expressions of length $O(4^{c \log m} m) = O(m^{2c+1})$. □

Thus we know that (strong pseudo-random) functions of formula depth at most $c \cdot \log m$ have short regular expressions of length at most $t_1(m) = \text{poly}(m)$, whereas we show next that most m-bit functions have only regular expressions of length at least $\Omega(2^m)$.

Lemma 3. *The number of languages described by regular expressions of length at most $t_2^{(1)}(m) = \frac{2^m}{40}$ is bounded by $\sqrt{2^{2^m}} = o(|B_m|)$.*

Proof. A regular expression of length at most t has rpn-length at most $6t$ [5]. At any position in the regular expression in reverse Polish notation there may be one of the seven distinct symbols $0, 1, +, \circ, ^*, \varepsilon, \emptyset$. Thus we can have at most $\sum_{j \le 6t} 7^j \le 7^{7t} \le 2^{20t}$ distinct regular expressions of rpn-length at most $6t$. The claim follows, since $2^{20 t_2(m)} = 2^{20 \frac{2^m}{40}} = 2^{2^{m-1}}$. □

$G_m^{(1)}(k_m(f_n)) \le t_1^{(1)}(n)$ holds for functions $f_n \in \mathcal{C}_1 \cap B_n$, because a formula for f_n can be transformed into a formula for $k_m(f_n)$ of same depth. Hence as a consequence of Lemma 2 and Lemma 3, $G^{(1)}$ separates $\mathcal{C}_1$ from random functions with thresholds $t_1^{(1)}(m) = O(m^{2c+1})$ and $t_2^{(1)}(m) = \frac{2^m}{40}$.

Thus we may apply the generic Lemma 1 and obtain that efficient algorithms approximating the length of a shortest regular expression for $\overline{L_{T_1}(f)}$ do not exist. However we have to specify the input not by a truth table but by a dfa.

Proposition 1. *Let $f \in B_m$ and let T be some function of m, then there is a dfa $D_T(f)$ with $O(2^m \cdot T)$ states that accepts $\overline{L_T(f)}$.*

Proof. The dfa $D_T(f)$ consists of a binary tree of depth m rooted at the initial state. A leaf that corresponds to a word x with $f(x) = 0$ gets a self loop, a leaf that corresponds to a word x with $f(x) = 1$ is starting point of a path of length $(T-1)m$ that can only be followed by inputs with $T-1$ repetitions of x. Each such path leads to a rejecting state and any wrong letter on this path, resp. any word longer than $(T-1)m$ (measured on the path only) leads to an

accepting trap state. Each state is accepting, except for those already described as rejecting. The dfa $D_T(f)$ has $O(2^m \cdot T)$ states. □

Thus our first main result is now an immediate consequence of Lemma 1.

Theorem 1. *Suppose that strong pseudo-random functions with parameter ε and formula depth bounded by $c \cdot \log m$ exist for some c.*

Let App be a randomized approximation algorithm that approximately determines the length of a shortest equivalent regular expression, when given a dfa with k states. Then for all $l \geq 1$, if App runs in time $2^{O((\log k)^l)}$ with $\epsilon_+ + \epsilon_- < \frac{2}{3}$, then App can only achieve an approximation factor $\mu \geq \frac{k}{\text{poly}((\log k)^{l/\varepsilon})}$.

The argument shows that there are always dfa's with optimal nfa's of size $\text{poly}(\log k)$, such that an "efficient" approximation algorithm can only determine nfa's of size $\frac{k}{\text{poly}(\log k)}$. Thus the original question of Jiang and Ravikumar [7] phrased for regular expressions instead of nfa's, namely whether it is possible to approximate within a polynomial, has a negative answer modulo cryptographic assumptions.

2.2 NFA's and Two-Way Automata of Polynomial Size

Here we use the functionals $G^{(2)}$ and $G^{(3)}$ defined by $G_m^{(2)}(f_m)$, resp. $G_m^{(3)}(f_m)$, to equal the minimum number of states, resp. transitions, of an nfa recognizing $\overline{L_{T_1}(f_m)}$. We choose T_1 as defined in the previous section and define $t_1^{(2)} = t_1^{(1)}$, $t_1^{(3)} = \left(t_1^{(1)}\right)^2$. We observe that the number of states of a minimum nfa is not larger than the length ℓ of an equivalent regular expression and the number of transitions is at most quadratic in ℓ. Thus all functions in $\mathcal{C}_1$ have nfa's of "size" at most $t_1^{(2)}$, resp. $t_1^{(3)}$. Moreover all but a negligible fraction of languages require nfa's with at least $t_2^{(2)}(m) = 2^{\frac{m}{2}-1}$ states, resp. $t_2^{(3)}(m) = \frac{2^m}{20m}$ transitions.

Lemma 4. *(a) The number of languages accepted by nfa's with at most $t_2^{(2)}(m) = 2^{\frac{m}{2}-1}$ states is bounded by $\sqrt{2^{m+2^m}} = o(|B_m|)$.*

(b) The number of languages accepted by nfa's with at most $t_2^{(3)}(m) = \frac{2^m}{20m}$ transitions is bounded by $\sqrt{2^{2^m}} = o(|B_m|)$.

Proof. (a) Let $N(k)$ be the number of distinct languages accepted by nfa's with at most k states over a two-letter alphabet. Then $N(k) \leq 2k \cdot 2^{2 \cdot k^2}$ [2] and hence $N(t_2^{(2)}(m)) \leq \frac{2}{2} \cdot 2^{\frac{m}{2}} \cdot 2^{2 \cdot \left(2^{\frac{m}{2}}/2\right)^2} = 2^{\frac{m}{2}} \cdot 2^{\frac{2}{4} \cdot \left(2^{(m/2)}\right)^2} = 2^{\frac{m}{2}} \cdot 2^{\frac{2^m}{2}} = \sqrt{2^{m+2^m}}$.

(b) We show that there are at most $M(k) = k^{10k}$ languages accepted by nfa's with at most k transitions over a two-letter alphabet. This establishes the claim, if we set $t_2^{(3)}(m) = \frac{2^m}{20m}$, since $M(t_2^{(3)}(m)) = \left(\frac{2^m}{20m}\right)^{10\frac{2^m}{20m}} \leq 2^{10m \cdot \frac{2^m}{20m}} = \sqrt{2^{2^m}}$.

For any nfa N with s states and k transitions there is an equivalent nfa N' with $s+1$ states, at most $2k$ transitions and exactly one final state. Just add a final state f, make every other state non-final and for every transition in N that leads to a final state in N, add a transition to f and keep every other transition.

There are at most $\binom{(s+1)^2}{2k}^2 \cdot s^2 \leq s^{8k+2}$ distinct languages over $\{0,1\}$ accepted by nfa's with s states and k transitions, since this is an upper bound for the number of possibilities to place $2k$ transitions for each letter of the alphabet $\{0,1\}$ and the number of choices for the initial and the final state.

We can assume that the number of states is bounded by the number of transitions and hence we have at most $k^{8k+2} \leq k^{10k}$ distinct languages. □

We apply Lemma 1 again and obtain:

Corollary 1. *Suppose that strong pseudo-random functions with parameter ε and formula depth bounded by $c \cdot \log m$ exist for some c.*

Let App be a randomized approximation algorithm that approximately determines the number of states (resp. number of transitions) of a minimum equivalent nfa, when given a dfa with k states. Then for all $l \geq 1$, if App runs in time $2^{O((\log k)^l)}$ with $\epsilon_+ + \epsilon_- < \frac{2}{3}$, then App can only achieve an approximation factor $\mu \geq \frac{\sqrt{k}}{\mathrm{poly}((\log k)^{l/\varepsilon})}$ (resp. $\mu \geq \frac{k}{\mathrm{poly}((\log k)^{l/\varepsilon})}$).

We finally mention that the assumption of strong pseudo-random functions with small formula depth can be replaced by the weaker assumption of strong pseudo-random functions with two-way dfa's of polynomial size. (Observe that two-way dfa's of polynomial size have the power of non-uniform Logspace, which is at least as powerful as non-uniform NC^1.) We show that two-way dfa's can be simulated efficiently by nfa's after repeating the input suitably often.

Lemma 5. *Let $m, k \in \mathbb{N}$ and let A_m be a two-way deterministic finite automaton with at most m^k states. Then there is a polynomial $T(m)$ and an nfa N_m with $O(T(m))$ states that accepts the complement of*

$$L_T(A_m) := \{x^{T(m)} \mid x \in \{0,1\}^m \wedge A_m \text{ accepts } x\}.$$

Proof. Obviously A_m runs for at most $T(m) = m \cdot m^k$ steps, since no cell can be visited twice in the same state. As shown in [13], A_m on input $x \in \{0,1\}^m$ can be simulated by a dfa D_m with $T(m)$ states working on input $x^{T(m)}$. The nfa N_m decides nondeterministically to run D_m (with final and non-final states interchanged) or to check whether the input is syntactically incorrect, i.e., verifying inequality or incorrect length. N_m has $t_1(m) = \mathrm{poly}(m)$ states, resp. transitions. □

When applying Lemma 1, we have to first redefine the number of repetitions to make sure that a class $\mathcal{C}_2$ of pseudo-random functions can be recognized by two-way dfa's of size m^k. We therefore set $T_2(m) = m^{k+1}$ and are guaranteed to find an equivalent nfa recognizing $\overline{L_{T_2}(f_m)}$ (for $f_m \in \mathcal{C}_2 \cap B_m$) with $t_1^{(2)}(m) = t_1^{(3)}(m) = O(T_2(m))$ states, resp. transitions.

2.3 The Minimum Consistent DFA Problem

In the *minimum consistent dfa* problem, sets $POS, NEG \subseteq \{0,1\}^*$ with $POS \cap NEG = \emptyset$ are given. The goal is to determine the minimum size of a dfa D such that $POS \subseteq L(D)$ and $NEG \cap L(D) = \emptyset$.

We again work with $T_2(m)$ repetitions and define $G_m^{(4)}(f_m)$ as the minimum size of a dfa accepting $POS = \{x^{T_2} | f_m(x) = 1\}$ and rejecting $NEG = \{x^{T_2} | f_m(x) = 0\}$. Observe that for any function $f_m \in \mathcal{C}_2 \cap B_m$ we have $G_m^{(4)}(f_m) \leq t_1^{(4)}(m) := m^{k+1}$, since any two-way dfa with m^k states can be simulated by a dfa with m^{k+1} states, if the input $x \in \{0,1\}^m$ is repeated $T_2(m) = m^{k+1}$ times. (See the proof of Lemma 5).

Lemma 6. *$G_m^{(4)}(f_m) \leq t_2^{(4)}(m) = \frac{2^m}{6m}$ holds for at most $\sqrt{2^{2^m}} = o(|B_m|)$ functions in B_m.*

Proof. Let $K(s)$ be the number of distinct languages accepted by dfa's with at most s states over a two-letter alphabet. Then $K(s) \leq s^{3s}$ [2] and hence $K(t_2^{(4)}(m)) \leq \left(\frac{2^m}{6m}\right)^{3\frac{2^m}{6m}} \leq 2^{3m\frac{2^m}{6m}} = \sqrt{2^{2^m}}$. The claim holds, since different functions f_m have different consistent dfa's. □

Thus $G_m^{(4)}$ separates $\mathcal{C}_2$ from random functions with thresholds $t_1^{(4)}, t_2^{(4)}$ and we obtain the following Theorem.

Theorem 2. *Suppose that strong pseudo-random functions with parameter ε and two-way dfa's with at most m^k states exist for some k.*

Let App be a randomized approximation algorithm that approximately determines the number of states of a minimum consistent dfa. For input length $N = \sum_{x \in POS \cup NEG} |x|$ and for all $l \geq 1$, if App runs in time $2^{O\left((\log N)^l\right)}$ with $\epsilon_+ + \epsilon_- < \frac{2}{3}$, then App can only achieve an approximation factor $\mu \geq \frac{N}{\text{poly}((\log N)^{l/\varepsilon})}$.

Efficient approximation algorithms determine, for $d \leq N$ examples, consistent dfa's of size $\frac{N}{\text{poly}(\log N)}$, whereas optimal dfa's have size $opt = \text{poly}(\log N)$. Thus upper bounds have as many as $2^{opt^{\frac{1}{l}}} \cdot d^\beta$ states, where $\beta < 1$ and l is sufficiently large. This result is stronger than the result of at least $opt^\alpha \cdot d^\beta$ due to Kearns and Valiant [8]. The stronger result is a consequence of our use of pseudo-random functions instead of one-way functions. (See also Naor and Reingold [11].)

3 Minimizing NFA's or Regular Expressions

We now assume that the language is specified concisely, i.e., as an nfa or a regular expression and prove in this scenario strong inapproximability results. We begin by investigating unary languages, i.e., languages over a one-letter alphabet, and show that no significant approximation is achievable, provided $P \neq NP$. This statement holds for size interpreted as number of states, transitions, resp. symbols.

Efficient approximations for state minimization within the factor $\frac{\sqrt{m}}{\ln m}$ are known *not* to exist, if $P \neq NP$ [3]. This result remains true for the number of transitions (resp. number of symbols in regular expressions), since the nfa

(resp. regular expression) built by the transformation in the proof [3, 15] has as many states as transitions (resp. symbols), and the number of states is a lower bound for the number of transitions of a minimal equivalent nfa (resp. symbols of a minimal equivalent regular expression). We can improve the inapproximability result, if we require the *construction* of a small nfa or regular expression.

Theorem 3. *Let A be an arbitrary unary nfa or regular expression of size m. Let opt be the size of a minimal equivalent nfa, resp. regular expression. For any $\delta > 0$, if $P \neq NP$, then no efficient algorithm can determine an nfa or regular expression A' equivalent to A with size at most $opt \cdot m^{1-\delta}$.*

Proof. Let A be an nfa (regular expression) constructed in the NP-completeness proof [3, 15]. A has the property that either $opt = 1$ or $opt > \frac{\sqrt{m}}{\ln m}$ and it is NP-complete to distinguish the two cases.

Suppose that there is a constant $\delta > 0$ and an efficient algorithm M that computes an nfa (regular expression) $M(A)$ equivalent to A with $\text{size}(M(A)) \leq opt \cdot \text{size}(A)^{1-\delta}$. If we apply M on its output again, then $\text{size}(M(M(A))) \leq opt \cdot \text{size}(M(A))^{1-\delta} \leq opt^2 \cdot \text{size}(A)^{(1-\delta)^2}$. If we repeat this process k times, then $\text{size}(M^k(A)) \leq opt^k \cdot \text{size}(A)^{(1-\delta)^k}$. So for $k \geq \frac{-2}{\log(1-\delta)}$, we have $\text{size}(M^k(A)) \leq opt^k \cdot \text{size}(A)^{\frac{1}{4}}$, hence for m large enough, $\text{size}(M^k(A)) \leq \frac{\sqrt{m}}{\ln m}$ if $opt = 1$ and $\text{size}(M^k(A)) \geq opt > \frac{\sqrt{m}}{\ln m}$ otherwise. □

Our negative results for *general* alphabets are based on the well known proof [10] of the PSPACE-completeness of "regular expression non-universality": Given a regular expression R, is $L(R) \neq \Sigma^*$? The PSPACE-completeness of regular expression non-universality implies the PSPACE-completeness of the exact minimization of nfa's and regular expressions.

The proof of [10] shows, that for an arbitrary language $L \in$ PSPACE there is a (generic) polynomial time transformation T such that $w \in L \Leftrightarrow L(T(w)) \neq \Sigma^*$, where $L(T(w))$ is the language described by the nfa, resp. regular expression $T(w)$. We restrict ourselves to languages $L \in \mathcal{L}$ where $\mathcal{L}$ is the class of languages that can be accepted by deterministic in-place Turing machines[1]. Our inapproximability result utilizes the following observation.

Lemma 7. *For any given language $L \in \mathcal{L}$ there is a deterministic in-place Turing machine M_L recognizing L with a single accepting state. M_L runs for at least 2^n steps on every input $w \in L$ of length n.*

Proof. Let M be some deterministic in-place Turing machine which accepts L and has only one accepting state q_f. We construct a Turing machine M_L that has all the states and transitions M has. However, whenever M_L enters q_f, it counts in binary from 0^n to 1^n, changes to a new state q'_f, when reaching

[1] $\mathcal{L}$ coincides with DSPACE($O(n)$), but considering only Turing machines that work in-place simplifies the proof.

1^n, and stops. q'_f is the only state in which M_L accepts and q'_f causes M_L to stop. □

Assume that M is a Turing machine with the properties stated in Lemma 7 which recognizes the PSPACE-complete language $L(M)$. (A padding argument shows that $\mathcal{L}$ contains PSPACE-complete languages.) We reduce the word problem for $L(M)$ to the minimization problem for nfa's. In particular for an input w of M, we construct an nfa A_w, which accepts exactly all words which are *not* concatenations of consecutive legal configurations starting from configuration q_0w leading to the unique accepting state. The exact description of the construction of A_w is omitted. It shows that A_w has $m = O(|w|)$ states.

If M rejects w, then $L(A_w)$ coincides with Σ^*. However, if M accepts w, then the configuration sequence x corresponding to the accepting computation is rejected by A_w and it is the only rejected word.

We show that $\Sigma^* \setminus \{x\}$ requires nfa's with at least $|w|$ states. Any accepting computation has length at least $2^{|w|}$, since M is a Turing-Machine as described in Lemma 7. Every dfa which excludes a single word of length at least $2^{|w|}$ needs at least $2^{|w|}$ states, thus every equivalent nfa needs at least $|w|$ states. Hence, if $L(A_w) = \Sigma^* \setminus \{x\}$ for some x with $|x| \geq 2^{|w|}$, then every nfa which accepts $L(A_w)$ needs at least $|w|$ states.

Thus, if $w \notin L(M)$, then $L(A_w)$ can be recognized by an nfa with one state, whereas for $w \in L(M)$, nfa's with at least $|w|$ states are required. Since A_w has $m = O(|w|)$ states, we have found the desired gap.

The inapproximability result for the number of transitions of nfa's and the number of symbols in regular expressions follows along the same lines.

Theorem 4. *Unless* $P =$ PSPACE*, it is impossible to efficiently approximate the size of a minimal nfa or regular expression describing* $L(A)$ *within an approximation factor of* $o(m)$ *when given an nfa or a regular expression* A *with* m *states, transitions or symbols respectively.*

Standard encoding arguments show that this PSPACE-completeness result is true for regular expressions or nfa's over any alphabet Σ with $|\Sigma| \geq 2$.

4 Conclusions and an Overview

We have been able to verify inapproximability of nfa's or regular expressions either for given nfa's or regular expressions (utilizing $P \neq NP$, resp. $P \neq$ PSPACE) or for given dfa's (assuming the existence of strong pseudo-random functions in NC^1, resp. Logspace).

The most notably open problem is a negative result for given dfa's utilizing only $P \neq NP$. Furthermore, what is the approximation complexity, when specifying a regular language $L \subseteq \{0,1\}^n$ by a truth table? Below we list our results and additionally mention nfa minimization for a given unary dfa as a third important open problem.

NFA Minimization Instance: An nfa N with k states over a binary alphabet. Solution: The size of a smallest nfa equivalent with N. Measure: Number of transitions or number of states. Bad News: Not approximable within $o(k)$. Assumption: $P \neq$ PSPACE. Reference: Theorem 4	Regular Expression Minimization Instance: A regular expression R with k symbols over a binary alphabet. Solution: The size of a smallest regular expression equivalent with R. Measure: Number of symbols. Bad News: Not approximable within $o(k)$. Assumption: $P \neq$ PSPACE. Reference: Theorem 4
The same is true for nfa → regular expression minimization and vice versa.	
Unary NFA Minimization Instance: An nfa N with k states over a unary alphabet. Solution: The size of a smallest nfa equivalent with N. Measure: Number of transitions or number of states. Bad News: Not approximable within $\frac{\sqrt{k}}{\ln k}$. Assumption: $P \neq NP$. Reference: [3]	Constructive Unary NFA Minimization Instance: An nfa N with k states over a unary alphabet. Solution: A smallest nfa equivalent with N. Measure: Number of transitions or number of states. Bad News: Not approximable within $k^{1-\delta}$ for any δ. Assumption: $P \neq NP$. Reference: Theorem 3
DFA → NFA Minimization (States) Instance: A dfa D with k states over a binary alphabet. Solution: The size of a smallest nfa equivalent with D. Measure: Number of states. Bad News: Not approximable within $\frac{\sqrt{k}}{\text{poly}(\log k)}$. Assumption: Strong pseudo-random functions in Logspace. Reference: Corollary 1	DFA → NFA Minimization (Transitions) Instance: A dfa D with k states over a binary alphabet. Solution: The size of a smallest nfa equivalent with D. Measure: Number of transitions. Bad News: Not approximable within $\frac{k}{\text{poly}(\log k)}$. Assumption: Strong pseudo-random functions in Logspace. Reference: Corollary 1
Unary DFA → NFA Minimization Instance: A dfa D with k states over a unary alphabet. Solution: The size of a smallest nfa equivalent with D. Measure: Number of states or transitions. Bad News: Optimal solution cannot be determined efficiently. Assumption: $NP \not\subseteq DTIME(n^{O(\log n)})$ Good News: Cyclic case can be approximated within $1 + \ln k$. Reference: [6], [3]	DFA → Regular Expression Minimization Instance: A dfa D with k states over a binary alphabet. Solution: The size of a smallest regular expression equivalent with D. Measure: Number of symbols. Bad News: Not approximable within $\frac{k}{\text{poly}(\log k)}$. Assumption: Strong pseudo-random functions in NC^1. Reference: Theorem 1
Minimum Consistent DFA Instance: Two finite sets P, N of binary strings. Solution: The minimal size of a dfa accepting all strings in P and rejecting all strings in N. Measure: Number of states in the automaton. Bad News: Not approximable within $\frac{\lvert P\rvert+\lvert N\rvert}{\text{poly}(\log(\lvert P\rvert+\lvert N\rvert))}$. Assumption: Strong pseudo-random functions in Logspace. Reference: Theorem 2	

References

1. Champarnaud, J.-M., Coulon, F.: NFA Reduction Algorithms by Means of Regular Inequalities, Developments in Language Theory, Springer, LNCS 2710, pp. 194-205.
2. Domaratzki, M., Kisman, D. Shallit, J.: On the Number of Distinct Languages Accepted by Finite Automata with n States, Journal of Automata, Languages and Combinatorics, 7(4), 2002.
3. Gramlich, G.: Probabilistic and Nondeterministic Unary Automata, Proc. of Math. Foundations of Computer Science, Springer, LNCS 2747, 2003, pp. 460-469.

4. Ilie, L., Navarro, G., Yu, S.: On NFA reductions, in J. Karhumaki, H. Maurer, G. Paun, G. Rozenberg, eds., Theory is Forever (Salomaa Festschrift), Springer, LNCS 3113, 2004, pp. 112-124.
5. Ilie, L., Yu, S.: Follow automata, Informat. & Computation 186, 2003, pp. 140-162.
6. Jiang, T., McDowell, E., Ravikumar, B.: The structure and complexity of minimal NFA's over a unary alphabet, Int. J. Found. of Comp. Sci., 2, 1991, pp. 163-182.
7. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard, SIAM Journal on Computing, 22(1), 1993, pp. 1117-1141.
8. Kearns, M., Valiant, L. G.: Cryptographic Limitations on Learning Boolean Formulae and Finite Automata, Journal of the ACM, 41(1), 1994, pp. 67-95.
9. Matz, O., Potthoff, A.: Computing small nondeterministic finite automata, Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Dpt. of CS., Univ. of Aarhus 1995, pp. 74-88.
10. Meyer, A. R., Stockmeyer, L. J.: The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space, Proc. 13th Ann. IEEE Symp. on Switching and Automata Theory, 1972, pp. 125-129.
11. Naor, M., Reingold, O.: Number-Theoretic constructions of efficient pseudo-random functions, Journal of the ACM, 51(2), 2004, pp. 231-262.
12. Pitt, L., Warmuth, M. K.: The Minimum Consistent DFA Problem Cannot be Approximated within any Polynomial, Journ. of the ACM, 1993, 40, pp. 95-142.
13. Pitt, L., Warmuth, M. K.: Prediction-Preserving Reducibility, Journal of Computer and System Science, 41(3), 1990, pp. 430-467.
14. Razborov, A. A., Rudich, S.: Natural Proofs, Journal of Computer and Systems Sciences, 55, 1997, pp. 24-35.
15. Stockmeyer, L., Meyer, A.: Word Problems Requiring Exponential Time, Proc. of the 5th Annual ACM Symposium on Theory of Computing, 1973, pp. 1-9.

Increasing Kolmogorov Complexity

Harry Buhrman[1,2], Lance Fortnow[3],
Ilan Newman[4], and Nikolai Vereshchagin[5,⋆]

[1] CWI, Amsterdam, the Netherlands
buhrman@cwi.nl
[2] ILLC, University of Amsterdam, the Netherlands
[3] Dept. of Computer Science, University of Chicago
fortnow@cs.uchicago.edu
[4] Dept. of Computer Science, Haifa University, Israel
ilan@cs.haifa.ac.il
[5] Moscow State University
kolya@ver.mccme.ru

Classification: Kolmogorov complexity, computational complexity.

1 Introduction

How much do we have to change a string to increase its Kolmogorov complexity? We show that we can increase the complexity of any non-random string of length n by flipping $O(\sqrt{n})$ bits and some strings require $\Omega(\sqrt{n})$ bit flips. For a given m, we also give bounds for increasing the complexity of a string by flipping m bits.

By using constructible expanding graphs we give an efficient algorithm that given any non-random string of length n will give a small list of strings of the same length, at least one of which will have higher Kolmogorov complexity. As an application, we show that BPP is contained in P relative to the set of Kolmogorov random strings. Allender, Buhrman, Koucký, van Melkbeek and Ronneberger [2] building on our techniques later improved this result to show that all of PSPACE reduces to P with an oracle for the random strings.

2 Increasing Complexity by Flipping Bits

Using the notation of Li and Vitányi, we use $C_U(x)$ to represent the size of the smallest program p such that $U(p) = x$. We fix a universal reference computer U and let $C(x) = C_U(x)$.

Assume we are given a binary string x. How much we can increase its complexity by flipping at most m bits of x? Let $N^m(x)$ denote the set of all strings

⋆ The work was done while visiting CWI; also supported in part by the RFBR grant 02-01-22001.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 412–421, 2005.

with Hamming distance at most m from x. Let $N^m(A)$ stand for the union of $N^m(x)$ over $x \in A$.

We use the notation $O(1), c, c_1, \ldots$ for constants depending on the reference machine U and $d, d_1, \ldots$ for absolute constants. The following, rather general theorem, asserting that the complexity of any 'typical' string in a set can be increased by flipping m bits to the expected $\log |N^m(A)|$ is an immediate implication of the 'cardinality' lower bound for Kolmogorov complexity.

Theorem 1. *Let $k, m, a \leq n$ be such that the following condition hold*

(*) for every set $A \subseteq \{0,1\}^n$ with $|A| > 2^a$, $N^m(A) \geq 2^k$ for $k < n$, or $N^m(A) \geq 2^n(1 - 1/c_2)$ for $k = n$.

Then, there are constants c_1, c_2 depending on the reference computer such that for every string x of complexity at least $C(x|n) \geq a + 2C(k, m|n, a) + c_1$ there is a string y obtained from x by flipping at most m bits such that $C(y|n) \geq k$.

Proof. Consider the following set

$$B = \{x \in \{0,1\}^n \mid C(y|n) < k \text{ for all } y \in N^m(x)\}.$$

As the Kolmogorov complexity of all strings in $N^m(B)$ is less than k we have $|N^m(B)| < 2^k$. In the case $n = k$ we may upper bound $|N^m(B)|$ better. Recall the following lower bound for the number of random strings (for the proof see [5]): for appropriate choice of c_2 for every n the number of strings y of length n with $C(y|n) \geq n$ is more than $2^n/c_2$. Therefore in the case $k = n$ we have $|N^m(B)| < 2^n(1 - 1/c_2)$.

In both cases we thus obtain $|B| \leq 2^a$. The set B may be enumerated given k, m, n. Therefore every string $x \in B$ can be described by m, n, k and its index in B of bit length a. Thus $C(x|n) < a + 2C(k, m|n, a) + c_1$ for all $x \in B$, where c_1 is a constant depending on the reference computer. In other words, for every x such that the last inequality is false there is $y \in N^m(x)$ with $C(y|n) \geq k$.

Theorem 1 is rather general and applies to any graph rather just the Boolean cube, when we replace 'flipping bits' with going to neighbors. This will be discussed in Section 3.

We now want to apply Theorem 1. For this we need to analyze the expanding properties of the Boolean cube. The complete analysis is given by the following theorem. We first introduce a notation. Let $b(n, l)$ denote the binomial sum: $b(n, l) = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{l}$.

Theorem 2 (Harper). *Let $J \leq 2^n$. Take all the strings with less than l ones and take the $J - l$ first strings with l ones in lexicographical order, where l is chosen so that $b(n, l-1) < J \leq b(n, l)$. Then the resulting set has the least $|N^1(A)|$ among all sets A with $|A| = J - l$.*

We will use the following corollary of Harper's theorem.

Corollary 1. *If $|N^m(A)| \leq b(n, l)$ and $l < n$ then $|A| \leq b(n, l - m)$ and $\frac{|N^m(A)|}{|A|} > (\frac{n-l}{l})^m$.*

We note that the second bound is very weak and becomes trivial for $l > n/2$. It will be sufficient though for our applications.

Proof. It is enough to prove the theorem in the case $m = 1$. For $m > 1$ we can use induction where inductive step is due to the case $m = 1$.

The first statement immediately follows from Harper's theorem. Let us prove the second one assuming that $l \leq n/2$. Let $J = |A|$. It suffices to establish the inequality assuming that A is the worst case set defined in the Harper's theorem. We have

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{l'-1} < |A| = J \leq \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{l'}$$

for some l'. We claim that $l' < l$. Indeed, otherwise $|A| > b(n, l-1)$ and therefore A has a string with l ones, thus $N(A)$ has a string with $l+1$ ones hence $|N(A)| > b(n, l)$, a contradiction. For the worst case set A we will prove that $|\Delta(A)|/|A| \geq (n-l')/(l'+1) \geq (n-l+1)/l$ where $\Delta(A)$ stands for the set of strings obtained from strings in A by changing a 0 to 1 (but not vice verse). (Actually $\Delta(A)$ and $N(A)$ differ by only one string, $00\ldots0$.)

Let B consist of all strings with less than l' ones thus $B \subset A$. Obviously, $\Delta(A)$ and $\Delta(B-A)$ do not intersect, as every string in the first set has at most l' ones and every string in the second set has $l'+1$ ones. Therefore it suffices to prove that $|\Delta(B)|/|B| \geq (n-l')/(l'+1)$ and $|\Delta(B-A)|/|B-A| \geq (n-l')/(l'+1)$.

The first inequality is proved as follows: $\Delta(B)$ is the set of all strings with at most l' ones except $00\ldots0$, so $|\Delta(B)| = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{l'}$. And $|B| = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{l'-1}$. The ratio of ith term the first sum and ith term in the second sum is $\binom{n}{i}/\binom{n}{i-1} = (n-i+1)/i \geq (n-l'+1)/l' \geq (n-l')/(l'+1)$.

Let us prove the second inequality. Let x be a string with l' ones and let C_x denote the set of all strings with l' ones that are less than or equal to x. We claim $|\Delta(C_x)|/|C_x|$ is a non-increasing function in x. To prove this claim it suffices to show that $|\Delta(C_x \cup \{x'\}) - \Delta(C_x)|$ is a non-increasing function in x where x' denotes the successor of x. The set $\Delta(C_x \cup \{x'\}) - \Delta(C_x)$ consists of all strings obtained by flipping all zeros in x' preceding the leading 1 (all other flips result in strings that are already in $\Delta(C_x)$). Hence $\Delta(C_x \cup \{x'\}) - \Delta(C_x)$ is equal to the number of zeros preceding the leading 1 in x'. And the latter number does not increases as x' increases.

For x equal to the last string with l' ones we have that $|\Delta(C_x)|/|C_x| = \binom{n}{l'+1}/\binom{n}{l'} = (n-l')/(l'+1)$ so we are done.

As a result we obtain the following triplets of k, m, a for which condition (*) and hence Theorem 1 hold.

Theorem 3. *There is a constant c_3, such that for every $k \leq n$, m and a string x of complexity at least $C(x|n) \geq a + 2C(m|n, a) + c_3$, there is a string y obtained from x by flipping at most m bits such that $C(y|n) \geq k$. Here $a = k - \lfloor m \log((n-l)/l) \rfloor$ where l is the least number such that $2^k \leq b(n, l)$.*

Proof. Let l be as above and let c_1 be the constant from Theorem 1. We first note that the conditions of Theorem 1 hold for a, k, m. Indeed, assume that $|N^m(A)| < 2^k$, then by the definition of l, $|N^m(A)| < \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{l}$ and by Corollary 1 we have $|A| < |N^m(A)|((n-l)/l)^{-m} < 2^k((n-l)/l)^{-m} \leq 2^a$. Hence, Theorem 1 asserts that for every string x with $C(x|n) \geq a + 2C(k, m|n, a) + c_1$ there is a string y obtained from x by flipping at most m bits such that $C(y|n) \geq k$.

It suffices to prove that $C(k, m|n, a) \leq C(m|n, a) + O(1) \leq \log m + O(1)$. To this end we will prove that k can be retrieved from m, n, a. By definition l is a function of n, k and a is a function of n, k, m. The function $l(n, k)$ is non-decreasing in k hence the function $a(n, k, m) = k - \lfloor (m+1)\log((n-l)/l) \rfloor$ is also non-decreasing in k, as the sum of two non-decreasing functions. Moreover, the first term increases by 1 as k increments by 1. This implies that k can be retrieved from m, n, a hence $C(k, m|n, a) \leq C(m|n, a) + O(1)$.

For $p \in (0, 1)$ let $H(p) = -p \log p - (1-p)\log(1-p)$ be the Shannon Entropy function. Note that for every $\alpha \in [0; 1)$ there are two different β_1, β_2 such that $h(\beta_1) = h(\beta_2) = \alpha$; they are related by the equality $\beta_1 + \beta_2 = 1$. Let $H^{-1}(\alpha)$ stand for the least of them. The function $H^{-1}(\alpha)$ increases in the range $(0, 0.5)$ as so does H.

Theorem 4. *For all $\alpha < 1$ and $i > 0$ there is $m(\alpha, i)$ (depending also on the reference computer) such that for all large enough n the following holds: For all x of length n with $C(x|n) \leq \alpha n$ there is y obtained from x by flipping at most $m(\alpha, i)$ bits such that $C(y|n) \geq C(x|n) + i$. For any fixed i there is a positive α such that $m(\alpha, i) = 1$.*

Proof. Fix α and i and let x be such that $C(x|n) \leq \alpha n$ and let $k = C(x|n) + i$. Let l be the least number such that $b(n, l) \geq 2^k$. We first prove that $l \leq \beta n$ for some constant $\beta < 1/2$, for large enough n. This means that $b(n, \beta n) \geq 2^k$ for some constant $\beta < 1$, for large enough n. Let β be any number in the interval $(H^{-1}(\alpha); 1/2)$ As $\alpha < 1$, the interval is not empty. Then, $b(n, \beta n) \geq \binom{n}{\beta n} \geq 2^{nH(\beta)(1+o(1))}$ (where the last inequality is standard, see e.g. [7]). Plugging in the definition of β can continue the inequality: $b(n, \beta n) \geq 2^{nH(\beta)(1+o(1))} \geq 2^{n\alpha+i} \geq 2^k$ for large enough n.

Define now $a = k - \lfloor m \log((n-l)/l) \rfloor$. Applying Theorem 3, with a, k, l as above, we get that for every x there is y obtained from x by flipping at most m bits such that $C(y|n) \geq k$, as needed, provided that

$$C(x|n) \geq a + 2C(m|n, a) + c_3. \tag{1}$$

To show that (1) holds, note that $C(m|n, a) \leq \log m$. Plugging this, along with the definition of a, k, in (1) we get that it is enough to show that $C(x|n) \geq C(x|n) + i - \lfloor m \log((n-l)/l) \rfloor + 2\log m + c_3$.

Using that $l \leq \beta n$ and the appropriate bound on β we get that it is enough to have $\lfloor m \log((1-\beta)/\beta) \rfloor > i + 2\log m + c_3$. Note that the definition of β implies that $\beta < 1/2$ hence $\frac{1-\beta}{\beta} > 1$. Therefore for large enough m we will have $\lfloor m \log((1-\beta)/\beta) \rfloor > i + 2\log m + c_3$.

Finally, let $m = 1$. Note that $\log((1-\beta)/\beta)$ tends to infinity as β tends to 0. Therefore for any fixed i there is a positive β such that $\lfloor m \log((1-\beta)/\beta) \rfloor > i + 2\log m + c_3$. Let α be equal to any positive real such that $H(\alpha) < \beta$.

Remark 1. We note that Theorem 4 works for fixed i, with respect to n, while m depends on i and α for fixed α or could be fixed when α gets small enough. One could ask whether it might be true that i could be a function of n, e.g, could the following strengthening of Theorem 4 be true: For any α (or even for some α) the complexity of a string x that is bounded by αn could be increased to $\alpha n + i(n)$ by changing only one bit. It is obvious that we cannot expect such a strengthening for $i(n) > \log n$, as given x the complexity of any y that differs form it in one place is at most $C(x|n) + \log n$. Other lower bounds on m vs. the amount of increase in complexity, and the relation to α are developed in Theorem 6 and Theorem 7.

Let us estimate how many bits we need to flip to increase complexity from $k-1$ to k when k is close to n, say for $k = n$.

Theorem 5. *For every x with $C(x|n) < n$ by flipping at most $c_3\sqrt{n}$ bits of x we can increase its complexity (by at least 1).*

Proof. Assume first that $C(x|n) \leq n-3$. Let $k = C(x|n)+1 \leq n-2$ and $m = c_4\sqrt{n}$ for a constant c_4 to be defined later. Apply Theorem 3. As $2^k \leq 2^n/4$ we have $l \leq n/2 - d_2\sqrt{n}$ and $(n-l)/l \geq (n/2+d_2\sqrt{n})/(n/2-d_2\sqrt{n}) \geq 1+2d_3/\sqrt{n} \geq 2^{d_4/\sqrt{n}}$ for large enough n. This implies that $a \leq k - c_4 d_4$. By Theorem 3 for every x with $C(x|n) \geq k - c_4 d_4 + 2C(m|a,n) + c_3$ there is y obtained from x by flipping at most m bits with $C(y|n) \geq k$. Obviously $C(m|a,n) \leq \log c_4 + c_5$. Therefore if c_4 is large enough we have $k - c_4 d_4 + 2C(m|a,n) + c_1 \leq k-1$ and we are done.

Assume now that $C(x|n) \geq n-2$. Let us prove that by flipping $O(\sqrt{n})$ bits we can increase the complexity of x up to n. This time we will apply Theorem 1 and Corollary 1 directly. For some c_3 for $l = n/2 + c_3\sqrt{n}$ we have $b(n,l) \geq 2^n(1-1/c_2)$, where c_2 is the constant from Theorem 1. Let $m = c_3\sqrt{n} + c_4\sqrt{n}$, where c_4 is chosen so that $b(n, l-m) \leq 2^{n-c_5}$, and c_5 will be chosen later. Let $a = n - c_5$ and $k = n$. By Corollary 1 the conditions of Theorem 1 are fulfilled. As $a + 2C(k,m|n) + c_1 \leq n - c_5 + 2\log c_5 + c_6 \leq n-2$ if c_5 was chosen appropriately, we are done.

Now we proceed to the lower bounds of the number of flipped bits. We will show that for every m there is α such that the complexity of some strings of complexity αn cannot be increased by flipping at most m bits. And there are strings for which we need to flip $\Omega(\sqrt{n})$ bits.

Theorem 6. *For every $m, k \geq 1$ there is a $\theta(k,m) < 1$ such that for every $\alpha > \theta(k,m)$, for almost all n there is a string x of length n such that $C(x|n) \leq \alpha n$ and $C(y|n) < C(x|n) + k$ for every string y obtained from x by flipping at most m bits.*

Proof. Let $\theta(k,m) = H(1/(1+2^{k/m}))$, and let $\theta(k,m) < \alpha$. As $k > 0$ we note that $1/(1+2^{k/m}) < 1/2$. Hence $\theta(k,m) < 1$. Without loss of generality assume that $\alpha < 1$.

Pick any β in the interval $(1/(1+2^{k/m}); H^{-1}(\alpha))$. Again by the bound above, and using the fact that H is monotone in the interval $(0; 0.5)$, the interval for β is non empty. Let $l = \beta n + c_2 m$ for a constant c_2 to be defined later.

We first prove that every string x having at most l ones satisfies the inequality $C(x|n) < \alpha n$, for large enough n. Indeed, the number of such strings is equal to $b(n,l)$ and hence is at most $2^{nH(l/n)(1+o(1))}$ [7] (as $l < n/2$). Therefore $C(x|n) < nH(\beta)(1+o(1)) + O(1) < n\alpha$ for large enough n, where the constant $O(1)$ depends on β, c_2, m and the reference computer.

So we need to show that there is a string x having at most l ones and satisfying the second statement of the theorem. Assume that this is not the case. Let then x_0 be a random string having at most βn ones, that is, $C(x_0|n) \geq \log(b(n,\beta n))$. If x_0 satisfies the statement then we are done. Otherwise there is x_1 having at most $\beta n + m$ ones such that $C(x_1|n) \geq C(x_0|n) + k$. Repeating this argument c_2 times we either find a string satisfying the statement or obtain a string x_{c_2} with $C(x_{c_2}|n) \geq C(x_0|n) + c_2 k$ having at most $\beta n + c_2 m = l$ ones. Hence $C(x_{c_2}|n) \geq \log(b(n,\beta n)) + c_2 k$. On the other hand, $C(x_{c_2}|n) \leq \log(b(n,l)) + 2C(l|n) + c_1 \leq \log(b(n,l)) + 2\log c_2 + c_3$, where c_3 depends on k, m, α and the reference computer. To obtain the contradiction we have to show that the upper bound of $C(x_{c_2}|n)$ is less than the lower bound. The ratio of $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{l}$ and $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{\beta n}$ can be is bounded using the following

Lemma 1. *If $j \geq s \geq 0$ and $j + s \leq n/2$ then*

$$\frac{b(n, j+s)}{b(n,j)} \leq 1 + \Big(\frac{n-j+s}{j-s+1}\Big)^s.$$

Proof.

$$\begin{aligned}\frac{b(n,j+s)}{b(n,j)} &\leq 1 + \max_{i=1}^{s} \binom{n}{j+i} \Big/ \binom{n}{j+i-s} \\ &\leq 1 + \max_{i=1}^{s} \Big(\frac{n-j-i+s}{j+i-s+1}\Big)^s \leq 1 + \Big(\frac{n-j+s}{j-s+1}\Big)^s.\end{aligned}$$

By Lemma 1 we have $\frac{b(n,l)}{b(n,\beta n)} \leq 2\Big(\frac{1-\beta}{\beta}\Big)^{c_2 m}$. Thus, to achieve contradiction it is enough to choose c_2 so that

$$1 + c_2 m \log((1-\beta)/\beta) + 2\log c_2 + c_3 < c_2 k. \tag{2}$$

Indeed, by the choice of β we have $m\log((1-\beta)/\beta) < k$. Hence the left hand side of (2) as a function of c_2 grows slowly than the right hand side and for large enough c_2 the inequality holds.

We will show now that sometimes we need to flip $\Omega(\sqrt{n})$ bits of x to increase its complexity even by 1.

Theorem 7. *There is a constant c such that for almost all n there is a string x of length n and complexity at most $n-1$, and such that the following holds: For every string y obtained from x by flipping at most $\sqrt{n}/c$ bits, $C(y|n) \leq C(x|n)$.*

Proof. For every c_1 there is c_2 such the set of strings with at most $n/2 - c_2\sqrt{n}$ ones has cardinality less than 2^{n-c_1} and therefore the complexity of every such string is less than $n - c_1 + 2\log c_1 + c_3$. Pick c_1 so that $n - c_1 + 2\log c_1 + c_3 \leq n-1$.

Let x_0 be a random string with at most $l = n/2 - (c_2+1)\sqrt{n}$ ones. Assume that for some x_1 we have $C(y|n) \geq C(x|n) + 1$ and x_1 differs from x_0 in at most $\sqrt{n}/c$ bits. In this case apply the same argument to x_1 and so on, c times. Either we will obtain x_i differing from x_0 in at most $i\sqrt{n}/c$ bits satisfying the statement of the theorem, or x_c such that $C(x_c|n) \geq C(x|n) + c$. In the first case x_i has at most $n/2 - (c_2+1)\sqrt{n} + \sqrt{n} = n/2 - c_2\sqrt{n}$ ones hence $C(x_i|n) \leq n-1$ and we are done.

Let us show now that the second case is impossible. We have $C(x_c|n) \geq \log \sum_i^l \binom{n}{i} + c$ and $C(x_c|n) \leq \log \sum_i^{l+\sqrt{n}} \binom{n}{i} + 2\log c + c_4$. By Lemma 1 we can upper bound the ratio $\sum_i^{l+\sqrt{n}} \binom{n}{i} / \sum_i^l \binom{n}{i}$ by

$$1 + \Big(\frac{n - l + \sqrt{n}}{l - \sqrt{n}}\Big)^{\sqrt{n}} = 1 + \Big(\frac{n/2 + (c_2+2)\sqrt{n}}{n/2 - (c_2+2)\sqrt{n}}\Big)^{\sqrt{n}} \leq c_5$$

for some constant c_5 for large enough n. Therefore we will have a contradiction if $\log c_5 + 2\log c + c_4 < c$.

3 Increasing Kolmogorov Complexity via Expanders

In this section we will use, in place of Boolean cubes, graphs that have stronger expansion properties. Recall the theorem of Margulis [6] on explicit expanders.

Theorem 8 (Margulis). *Let k be an integer and $G = (V, E)$ be the graph with vertices $V = \{0, \ldots, k-1\}^2$ where a vertex (x, y) is adjacent to vertices (x, y), $(x+1, y)$, $(x, y+1)$, $(x, x+y)$, and $(-y, x)$ (all operations are mod k). There is a positive ε such that for every $A \subset V$ the set $N(A)$ of all neighbors of vertices in A has at least $(1 + \varepsilon(1 - |A|/|V|))|A|$ elements.*

Let $k = 2^l$. We will identify strings of length $n = 2l$ and nodes of the Margulis' expander G. Let $N^d(u)$ denote the set of all nodes at the distance at most d from u in the graph G. Let $N^d(A)$ stand for the union of $N^d(u)$ over $u \in A$.

Theorem 9. *There is a constant c_2 such that for every node u in G with $C(u|n) < n$ there is a node $v \in N^{c_2}(u)$ with $C(u|n) > C(v|n)$.*

Proof. Let c be a constant to be specified later. Let c_1 be the constant such that for every n the number of strings y of length n with $C(y|n) \geq n$ is more than $2^n/c_1$. Let c_2 be a constant such that $(1 + \varepsilon c_1)^{c_2} \geq 2^c$.

Assume that the statement of the theorem is false for some node u. Let us exhibit a small set containing u. Let

$$A_i = \{u' \in V \mid \forall v \in N^i(u')\ C(v|n) \leq C(u|n)\}$$

where $i = 0, \ldots, c_2$. Obviously, $A_{i-1} = N(A_i)$ and therefore we have $A_0 \supset A_1 \supset \cdots \supset A_{c_2}$. By definition, all strings in A_{c_2} have Kolmogorov complexity at most $C(u|n) < n$. Therefore we can upper bound $|A_0|$ in two ways: $|A_0| \leq 2^{C(u|n)+1}$ and $|A_0| \leq 2^n - 2^n/c_1$. By expansion property we have

$$|A_0| \geq (1 + \varepsilon c_1)|A_1| \geq \cdots \geq (1 + \varepsilon c_1)^{c_2}|A_{c_2}| \geq 2^c|A_{c_2}|.$$

Hence A_{c_2} is small, $|A_{c_2}| \leq 2^{-c}|A_0| \leq 2^{C(u|n)+1-c}$. Since u is in A_{c_2} and A_{c_2} can be enumerated given l and $C(u|n)$, we can describe u by its index in the enumeration of A_{c_2} of length $C(u|n) + 1 - c$ and by c (and $C(u|n)$ can be computed from the length of the index and c). Hence $C(u|n) \leq (C(u|n) + 1 - c) + 2\log c + O(1)$. If c is large then this is a contradiction.

Using Theorem 9 we can design a polynomial time algorithm that having access to the oracle $\tilde{R} = \{x \mid C(x \mid |x|) \geq |x|\}$ for every even length $2l$ finds a string in $\tilde{R}$ of length $2l$.

Theorem 10. *There is an algorithm that having access to the oracle $\tilde{R} = \{x \mid C(x \mid |x|) \geq |x|\}$ for every even length $2l$ in time poly(l) finds a string in $\tilde{R}$ of length $2l$.*

Proof. We will find strings $u_0, \ldots, u_l$ such that $|u_i| = 2i$ and $u_i \in \tilde{R}$. Let u_0 be the empty string. Certainly $u_0 \in \tilde{R}$.

To find u_i given u_{i-1} append first 00 to u_{i-1} and let u be the resulting string. As $C(u_{i-1}|2(i-1)) \geq 2(i-1)$ we have $C(u_i|2i) \geq 2i - c_3$ for some constant c_3. By Theorem 9 there is a string v at in $N^{c_3c_2}(u)$ such that $v \in \tilde{R}$. Making at most $5^{c_3c_2}$ queries to the oracle $\tilde{R}$ we find the first such v and let $u_i = v$.

Remark 2. The same argument applies as well to every set of the form $\{x \mid C(x \mid |x|) \geq f(|x|)\}$ where $f(n) \leq n$ and $f(n+1) \leq f(n) + O(\log n)$ for all n. In this case we search for v in $N^{(c_3+O(\log n))c_2}(u)$ in place of $N^{c_3c_2}(u)$. As $N^{(c_3+O(\log n))c_2}(u)$ still has polynomial size the algorithm runs in polynomial time. Note that the algorithm need no information about f other than the constant hidden in $O(\log n)$.

Remark 3. The argument applies also to find random strings of odd lengths, but that requires more technical details. Given a string u of even length $n = 2l$ with $C(u|n) \geq n$ we need to find a string v of odd length $n = 2l + 1$ with $C(v|n) \geq n$. To this and we can use Margulis' expander for the largest k such that $k^2 \leq 2^{2l+1}$. Obviously $k^2 \geq 2^{2l}$ and we may identify strings of length $2l + 1$ ending with 0 with the first 2^{2l} nodes of the graph, and the other nodes with the first remaining strings of length $2l+1$. Again we have $C(u0|2l+1) \geq 2l+1-c_3$ for a constant c_3. For large enough l the difference between 2^{2l+1} and k^2 is less than $2^{2l+1}/(2c_1)$

where c_1 is a constant such that the number of random strings of length $2l+1$ is at least $2^{2l+1}/c_1$. Therefore at least $k^2/(2c_1)$ nodes in the graph are random and we can apply the arguments from the proof of Theorem 9 with $2c_1$ in place of c_1.

Corollary 2. *$BPP \subset P^{\tilde{R}}$*

Proof. Let M be a probabilistic machine recognizing a language A. Let n be the length of input to M. We can assume that the probability that M errs on at least one string of length n is at most 2^{-n}. Let n^d be the length of random strings used by M on inputs of length n.

Here is the deterministic algorithm with oracle $\tilde{R}$ to recognize A: Find a string $r \in \tilde{R}$ of length n^d and run M on the input x using r as the sequence of random bits for M (we use the same string r for all inputs x). Then output the result of M.

If for some string of length n the answer is incorrect then the string r falls into a set of cardinality 2^{n^d-n} that is identified by n and M and hence $C(r|n^d) \leq n^d - n + O(1) < n^d$ for n large enough, which is a contradiction. Thus our polynomial time algorithm with oracle $\tilde{R}$ is correct for almost all inputs. Hardwiring the table of answers for small inputs we obtain a polynomial time algorithm with oracle $\tilde{R}$ that recognizes A (on all inputs).

Let us turn to the unconditional Kolmogorov complexity $C(x)$. Let $R = \{x \mid C(x) \geq |x|\}$. We will show that Theorem 10, the next two remarks and Corollary 2 generalize to R. As to Theorem 9, we can prove only a weaker version:

Theorem 11. *There is a constant c_2 such that for every node u in G with $C(u) < n$ there is a node $v \in N^{c_2 \log n + c_2}(u)$ with $C(u) > C(v)$.*

Proof. Essentially the same proof, as for Theorem 9 but this time we need to choose c_2 so that $(1+\varepsilon c_1)^{c_2 \log n + c_2} \geq 2^{c+2\log n}$. In place of inequality $C(u|n) \leq C(u|n) + 1 - c + 2\log c + O(1)$ we have the inequality $C(u) \leq C(u) + 1 - c - 2\log n + 2\log c + 2\log l + O(1)$. The term $2\log n$ is needed as this time we have to identify the length of u.

However, to prove the analog of Theorem 10 we need only to increase Kolmogorov complexity of strings u with $C(u) \geq |u| - O(1)$. For that special case we have

Theorem 12. *For every constant c_3 there is a constant c_4 such that for every node u in G with $n > C(u) \geq n - c_3$ there is a node $v \in N^{c_4}(u)$ with $C(u) > C(v)$.*

Proof. Again the same proof but in place of inequality $C(u|n) \leq C(u|n) + 1 - c + 2\log c + O(1)$ we have the inequality $C(u) \leq C(u) + 1 - c + 2\log c + O(1)$. This time we can find the length of u from the length $C(u) + 1 - c$ of the index of u in A_{c_4} and from c, as $C(u)$ and $|u|$ are close to each other.

Therefore Theorem 10, the next two remarks and Corollary 2 generalize to the unconditional Kolmogorov complexity.

References

1. Ahlswede, Gács, Körner. Bounds on conditional probabilities with applications in multi-user communication. Z. Wahrscheinlichkeitstheorie verw. Gebiete 34 (1976) 157–177.
2. Allender, Buhrman, Koucký, van Melkbeek and Ronneberger. Power from Random Strings. 43rd IEEE Symposium on the Foundations of Computer Science (2002) 669–678.
3. L.H. Harper. Optimal numberings and isoperimetric problems on graphs. J. Combinatorial Theory 1 (1966) 385–393.
4. G.O.H. Katona. The Hamming-sphere has minimum boundary. Studia Scientarium Mathematicarum Hungarica 10 (1975) 131–140.
5. M. Li, P.M.B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. New York, Springer-Verlag, 1997.
6. G.A. Margulis. Explicit constructions of concentrators. Explicit construction of concentrators. Probab. Info. Trans., 9 (1975), 325–332. (Translated into English from "Problemy peredachi informatsii" 9(2) (1973) 71–80.)
7. Rosen (ed.), Handbook of Discrete Combinatorial Mathematics, CRC Press, 2000.

Kolmogorov-Loveland Randomness and Stochasticity

Wolfgang Merkle[1], Joseph Miller[2], André Nies[3], Jan Reimann[1], and Frank Stephan[4]

[1] Universität Heidelberg, Heidelberg, Germany
[2] Indiana University, Bloomington, Indiana, USA
[3] University of Auckland, Auckland, New Zealand
[4] National University of Singapore, Singapore

Abstract. One of the major open problems in the field of effective randomness is whether Martin-Löf randomness is the same as Kolmogorov-Loveland (or KL) randomness, where an infinite binary sequence is KL-random if there is no computable non-monotonic betting strategy that succeeds on the sequence in the sense of having an unbounded gain in the limit while betting successively on bits of the sequence. Our first main result states that every KL-random sequence has arbitrarily dense, easily extractable subsequences that are Martin-Löf random. A key lemma in the proof of this result is that for every effective split of a KL-random sequence at least one of the halves is Martin-Löf random. We show that this splitting property does not characterize KL-randomness by constructing a sequence that is not even computably random such that every effective split yields subsequences that are 2-random, hence are in particular Martin-Löf random.

A sequence X is KL-stochastic if there is no computable non-monotonic selection rule that selects from X an infinite, biased sequence. Our second main result asserts that every KL-stochastic sequence has constructive dimension 1, or equivalently, a sequence cannot be KL-stochastic if it has infinitely many prefixes that can be compressed by a factor of $\alpha < 1$ with respect to prefix-free Kolmogorov complexity. This improves on a result by Muchnik, who has shown a similar implication where the premise requires that such compressible prefixes can be found effectively.

1 Introduction

In 1998, Muchnik, Semenov, and Uspensky [11] combined non-monotonic selection rules in the sense of Kolmogorov and Loveland with the concept of computable betting strategies. The resulting concept of *non-monotonic betting strategies* is a generalization of the concept of monotonic betting strategies, used by Schnorr to define a randomness notion nowadays known as *computable randomness*. Schnorr's motivation behind this randomness concept was his criticism of Martin-Löf randomness [7] as not being a completely effective notion of

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 422–433, 2005.

randomness, since the sets used in Martin-Löf tests only have to be uniformly *enumerable*.

An infinite binary sequence against which no *computable* non-monotonic betting strategy succeeds is called *Kolmogorov-Loveland random*, or KL-random, for short. Muchnik, Semenov, and Uspensky [11] showed that Martin-Löf randomness implies KL-randomness. Muchnik et al. [11] and others [1] raised the question whether the two concepts are different. This is now a major open problem in the area. A proof that both concepts are the same would give a striking argument against Schnorr's criticism of Martin-Löf randomness.

Most researchers conjecture the notions are different. However, a result of Muchnik [11] indicates that KL-randomness is rather close to Martin-Löf randomness.

Recall that it is possible to characterize Martin-Löf randomness as incompressibility with respect to prefix-free Kolmogorov complexity K: A sequence A is random if and only if there is a constant c such that the K-complexity of the length n prefix $A \upharpoonright n$ of A is at least $n - c$. It follows that a sequence A cannot be Martin-Löf random if there is a function h such that

$$K(A \upharpoonright h(c)) \leq h(c) - c \qquad \text{for every } c. \tag{1}$$

On the other hand, by the result of Muchnik [11] a sequence A cannot be KL-random if (1) holds for a *computable* function h. So, the difference between Martin-Löf randomness and KL-randomness appears, from this viewpoint, rather small. Not being Martin-Löf random means that for any given constant bound there are infinitely many initial segments for which the compressibility exceeds this bound. If, moreover, we are able to detect such initial segments effectively (by means of a computable function), then the sequence cannot even be KL-random.

In this paper we continue the investigations by Muchnik, Semenov, and Uspensky, and give additional evidence that KL-randomness behaves similar to Martin-Löf randomness.

In Section 3 we refine the splitting technique that Muchnik used in order to obtain the result mentioned above. We show that if A is KL-random and Z is a computable, infinite and co-infinite set of natural numbers, either the bits of A whose positions are in Z or the remaining bits form a Martin-Löf random sequence. In fact both do if A is Δ^0_2. Moreover, in that case, for each computable, nondecreasing, and unbounded function g and almost all n, $K(A \upharpoonright n) \geq n - g(n)$.

We construct counterexamples that show that two of the implications mentioned in the preceding paragraph cannot be extended to equivalences. First, there is a sequence that is not computably random all whose "parts" in the sense above (i.e., which can be obtained through a computable splitting) are Martin-Löf random. Second, there is a sequence A that is not even stochastic such that for all g as above and almost all n, $K(A \upharpoonright n) \geq n - g(n)$; moreover, the sequence A can be chosen to be left-c.e. if viewed as the binary expansion of a real.

In the last two sections we consider KL-stochasticity. A sequence is KL-stochastic if there is no computable non-monotonic selection rule that selects

from the given sequence a sequence that is biased in the sense that the frequencies of 0's and 1's do not converge to 1/2. First we give a more direct construction of a KL-stochastic sequence that is not even weakly 1-random. Next we consider constructive dimension. Muchnik [11] demonstrates, by an argument similar to his proof that a sequence A cannot be KL-random if there is a computable function that satisfies (1), that a sequence A cannot be KL-stochastic if there is a computable, unbounded function h and a rational $\alpha < 1$ such that

$$K(A \upharpoonright h(i)) \leq \alpha h(i) \qquad \text{for every } i, \tag{2}$$

i.e., if we can effectively find arbitrarily long prefixes of A that can be compressed by a factor of α in the sense that the prefix-free Kolmogorov complexity of the prefix is at most α times the length of the prefix. Theorem 22 below states that KL-stochastic sequences have constructive dimension 1. This is equivalent to the assertion that in the second mentioned result of Muchnik it is not necessary to require that the function h be computable, i.e., it suffices to require the mere existence of arbitrarily long prefixes of A that can be compressed by a factor of α.

In the remainder of the introduction we gather some notation that will be used throughout the text. Unless explicitly stated otherwise, the term *sequence* refers to an infinite binary sequence and a *class* is a set of sequences. Sequences are denoted by capital letters like $A, B, \ldots, R, S, \ldots$.

We will often deal with generalized joins and splittings. Assume that Z is an infinite and co-infinite set of natural numbers. The *Z-join* $A_0 \oplus_Z A_1$ of sequences A_0 and A_1 is the result of merging the sequences using Z as a guide. Formally,

$$A_0 \oplus_Z A_1(n) = \begin{cases} A_0(|\overline{Z} \cap \{0, \ldots, n-1\}|) & \text{if } Z(n) = 0, \\ A_1(|Z \cap \{0, \ldots, n-1\}|) & \text{if } Z(n) = 1. \end{cases}$$

On the other hand, given a sequence A and a set $Z \subseteq \omega$ one can obtain a new sequence (string) $A \upharpoonright_Z$ by picking the positions that are in Z. Let p_Z denote the principal function of Z, i.e. $\mathrm{p}_Z(n)$ is the $(n+1)$st element of Z (where this is undefined if no such element exists). Formally,

$$A \upharpoonright_Z (n) = A(\mathrm{p}_Z(n)), \quad \text{where } \mathrm{p}_Z(n) = \mu x[|Z \cap \{0, \ldots, x\}| \geq n+1].$$

If Z is infinite, $A \upharpoonright_Z$ will yield a new infinite sequence, otherwise we define $A \upharpoonright_Z$ to be the string of length $|Z|$ extracted from A via Z. Note that this notation is consistent with the usual notation of initial segments in the sense that $A \upharpoonright_n = A \upharpoonright_{\{0,\ldots,n-1\}}$. Observe that $A = A_0 \oplus_Z A_1$ if and only if $A \upharpoonright_Z = A_1$ and $A \upharpoonright_{\overline{Z}} = A_0$.

Due to space considerations, several proofs are omitted. These proofs can be found in the full version of this paper [10].

2 Random and Stochastic Sequences

In this section, we give a brief and informal review of the concepts of effective randomness and stochasticity that are used in the following, for further details

and formal definitions we refer to the surveys and monographs cited in the bibliography [11, 1, 5, 9, 19].

Intuitively speaking, a non-monotonic betting strategy defines a process that place bets on bits of a given sequence X. More precisely, the betting strategy determines a sequence of mutually distinct places $n_0, n_1, \ldots$ at which it bets a certain portion of the current capital on the value of the respective bit of X being 0 or 1. (Note that, by betting none of the capital, the betting strategy may always choose to "inspect" the next bit only.) The place n_{i+1} and the bet which is to be placed depends solely on the previously scanned bits $X(n_0)$ through $X(n_i)$. Payoff is fair in the sense that the stake is double in case the guess on the next bit was correct and is lost otherwise. For a betting strategy b that is applied with a certain initial capital c, we write $d_b^A(n)$ for the capital that has been accumulated after the first n bets on the bits of a sequence A while betting according to b; the function d_b is called the corresponding payoff function or martingale.

A non-monotonic betting strategy b *succeeds on* a sequence A if

$$\limsup_{n\to\infty} d_b^A(n) = \infty\,.$$

A sequence A is KL-*random* if there is no partial computable non-monotonic betting strategy that succeeds on A. The concept of KL-randomness remains the same if one uses in its definition computable instead of partial computable non-monotonic betting strategies [9].

One can modify the concept of a betting strategy in that, instead of specifying a bet on every next bit to be scanned, the strategy simply determines whether the next bit should be selected or not. Such a strategy is called a *selection rule*. The sequence selected from X is then the sequence of all bits that are selected, in the order of selection. A sequence X is called stochastic with respect to a given class of admissible selection rules if no selection rule in the class selects from X an infinite sequence that is biased in the sense that the frequencies of 0's and 1's do not converge to 1/2. A sequence is *Kolmogorov-Loveland stochastic* or *KL-stochastic*, for short, if the sequence is stochastic with respect to the class of partial computable non-monotonic selection rules; again, this concept remains the same if one replaces "partial computable" by "computable". A sequence is *Mises-Wald-Church stochastic* or *MWC-stochastic*, for short, if the sequence is stochastic with respect to the class of partial computable monotonic selection rules.

Furthermore, we consider Martin-Löf random sequences [7]. Let $W_0, W_1, \ldots$ be a standard enumeration of the computably enumerable sets.

Definition 1. *A* Martin-Löf test *is a uniformly computably enumerable sequence* $(A_n : n \in \omega)$ *of sets of strings such that for every* n*,*

$$\lambda\,(\{Y : Y \text{ has a prefix in } A_n\}) \leq 2^{-(n+1)},$$

where λ *denotes Lebesgue measure on Cantor space.*

A sequence $\mathcal{X}$ is covered *by a Martin-Löf test $(A_n : n \in \omega)$ if for every n the set A_n contains a prefix of X. A sequence is* Martin-Löf random *if it cannot be covered by any Martin-Löf test.*

A Martin-Löf test is called a *Schnorr test* if the Lebesgue measure of the set $\{Y : Y$ has a prefix in $A_n\}$ is computable in n (in the usual sense that the measure can be approximated effectively to any given precision strictly larger than 0); a sequence is called *Schnorr-random* if it cannot be covered by a Schnorr test.

Remark 2. Let b be a computable non-monotonic betting strategy that on every sequence scans all places of the sequence. Then there is a monotonic betting strategy that succeeds on every sequence on which b succeeds. This follows from results of Buhrman, Melkebeek, Regan, Sivakumar and Strauss [2].

One can use this fact to infer the following proposition.

Proposition 3. *The class of computably random sequences is closed under computable permutations of the natural numbers.*

3 Splitting Properties of KL-Random Sequences

KL-random sequences bear some properties which make them appear quite "close" to Martin-Löf random sequences. One of them is a splitting property, which stresses the importance of non-monotonicity in betting strategies.

Proposition 4. *Let Z be a computable, infinite and co-infinite set of natural numbers, and let $A = A_0 \oplus_Z A_1$. Then A is KL-random if and only if*

$$A_0 \text{ is } KL^{A_1}\text{-random} \quad \text{and} \quad A_1 \text{ is } KL^{A_0}\text{-random}. \tag{3}$$

Theorem 5. *Let Z be a computable, infinite and co-infinite set of natural numbers. If the sequence $A = A_0 \oplus_Z A_1$ is KL-random, then at least one of A_0 and A_1 is Martin-Löf random.*

Proof. Suppose neither A_0 nor A_1 is Martin-Löf random. Then there are Martin-Löf tests $(U_n^0 : n \in \omega)$ and $(U_n^1 : n \in \omega)$ with $U_n^i = \{\sigma_{n,0}^i, \sigma_{n,1}^i, \dots\}$, that cover A_0 and A_1, respectively.

Define functions f_0, f_1 by $f_i(n) = \mu k\, \sigma_{n,k}^i \sqsubset A_i$. Obviously there must be an $i \in \{0,1\}$ such that there are infinitely many m for which $f_i(m) \geq f_{1-i}(m)$. We define a new Martin-Löf test $\{V_n\}$ by $V_n = \bigcup_{m>n} \bigcup_{k=0}^{f_i(m)} [\sigma_{n,k}^{1-i}]$. Then $\{V_n\}$ is a Schnorr test relative to the oracle A_i (a Schnorr^{A_i}-test) and covers A_{1-i}, so A_{1-i} is not Schnorr^{A_i}-random. Since KL-randomness implies Schnorr-randomness (for relativized versions, too), it follows that A_{1-i} is not KL^{A_i}-random, contradicting Theorem 4. □

An interesting consequence of (the relativized form of) Theorem 5 is stated in Theorem 8; in the proof of this theorem we will use Remark 6, due to van Lambalgen [20] (also see [4] for a proof). For KL-randomness, the closest one

can presently come to van Lambalgen's result is Proposition 4. Note the subtle difference: in the case of Martin-Löf randomness, one merely needs A_0 to be random, not random relative to A_1.

Remark 6. Let Z be a computable, infinite and co-infinite set of natural numbers. The sequence $A = A_0 \oplus_Z A_1$ is Martin-Löf random if and only if A_0 is Martin-Löf random and A_1 is Martin-Löf random relative to A_0. (Furthermore, this equivalence remains true if we replace Martin-Löf randomness by Martin-Löf randomness relative to some oracle.)

Definition 7. *A set Z has* density α *if* $\lim_{m\to\infty} \frac{|Z\cap\{0,\ldots,m-1\}|}{m} = \alpha$.

Theorem 8. *Let R be a KL-random sequence and let $\alpha < 1$ be a rational. Then there is a computable set Z of density at least α such that $R\upharpoonright_Z$ is Martin-Löf random.*

Proof. For a start, we fix some notation for successive splits of the natural numbers. Let $\{\mathrm{N}_w\}_{w\in\{0,1\}^*}$ be a uniformly computable family of sets of natural numbers such that for all w,

$$(i)\ \mathrm{N}_\varepsilon = \omega\ , \qquad (ii)\ \mathrm{N}_w = \mathrm{N}_{w0}\dot\cup\mathrm{N}_{w1}\ , \qquad (iii)\ \mathrm{N}_w \text{ has density } \frac{1}{2^{|w|}}\ ,$$

where $\dot\cup$ denotes disjoint union.

By (iii), for any word w the complement $\overline{\mathrm{N}_w}$ of N_w has density $1 - 1/2^{|w|}$, thus it suffices to show that there are words $w_1 \sqsubseteq w_2 \sqsubseteq \ldots$ such that for all i,

$$(iv)\ |w_i| = i \ \text{ and } \ (v)\ R_i = R\upharpoonright_{\overline{\mathrm{N}_{w_i}}} \text{ is Martin-Löf random .}$$

The w_i are defined inductively. For a start, observe that by Theorem 5 for $r_1 = 0$ or for $r_1 = 1$ the sequence $R\upharpoonright_{\mathrm{N}_{r_1}}$ is Martin-Löf random; pick r_1 such that the latter is true and let $w_1 = 1 - r_1$. For $i > 1$, let w_i be defined as follows. By Proposition 4 the sequence $R\upharpoonright_{\mathrm{N}_{w_i}}$ is KL-random relative to R_{i-1}, hence by (ii) and by a relativized version of Theorem 5, for $r_i = 0$ or for $r_i = 1$ the sequence $R\upharpoonright_{\mathrm{N}_{wr_i}}$ is Martin-Löf random relative to R_i; pick r_i such that the latter is true and let $w_i = w(1 - r_i)$.

Now (iv) follows for all i by an easy induction argument, using van Lambalgen's result from Remark 6. □

The functions f_i in the proof of Theorem 5 can be viewed as a modulus for a certain type of approximation to the sequences under consideration. The technique of comparing two given moduli can also be applied to other types of moduli, e.g., to a modulus of convergence of an effectively approximable sequence.

Theorem 9. *Let Z be a computable, infinite and co-infinite set of natural numbers and let $A = A_0 \oplus_Z A_1$ be KL-random where A_1 is in Δ^0_2. Then A_0 is Martin-Löf random.*

By applying Theorem 9 to the set Z and its complement, the following Corollary is immediate.

Corollary 10. *Let Z be a computable, infinite and co-infinite set of natural numbers and let $A = A_0 \oplus_Z A_1$ be KL-random and Δ^0_2. Then A_0 and A_1 are both Martin-Löf random.*

The next example shows that splitting properties like the one considered in Corollary 10 do not necessarily imply Martin-Löf randomness.

Theorem 11. *There is a sequence A which is not computably random such that for each computable infinite and co-infinite set V, $A\restriction_V$ is 2-random, i.e. is Martin-Löf random relative to $\emptyset'$.*

A function g is an *order* if g is computable, nondecreasing, and unbounded.

Corollary 12. *Suppose A is in Δ^0_2 and is KL-random. Then for each order g and almost all n, $K(A \restriction n) \geq n - g(n)$.*

Remark 13. In the full version of this article it will be shown that there is a left-c.e. real A which is not MWC-stochastic, but satisfies $K(A \restriction n) \geq^+ n - g(n)$ for each order g and almost all n. Thus even for left-c.e. reals, the conclusion of Corollary 12 is not equivalent to Martin-Löf randomness.

4 Kolmogorov-Loveland Stochasticity

There are two standard techniques for constructing KL-random sequences. The first one is a probabilistic construction due to van Lambalgen [20]. The second one is to construct directly a Martin-Löf random sequence, e.g., by diagonalizing against a universal left-computable martingale. Theorem 14 is demonstrated by a further technique that allows to construct KL-stochastic sequences with certain additional properties that could not be achieved by the mentioned standard methods.

A sequence X is *weakly 1-random* (also called *Kurtz-random*) if X is contained in every c.e. open class of uniform measure 1. Note that Schnorr randomness implies weak 1-randomness, but not conversely.

Theorem 14. *There is a non-empty Π^0_1 class $\mathcal{P}$ of KL-stochastic sequences such that no $X \in \mathcal{P}$ is weakly 1-random.*

The proof of Theorem 14 is omitted due to space considerations. By the usual basis theorems [12], the following corollary is immediate.

Corollary 15. *There is a left-c.e., not weakly 1-random KL-stochastic sequence. There is a low, not weakly 1-random, KL-stochastic sequence. There is a not weakly 1-random KL-stochastic sequence that is of hyperimmune-free degree.*

5 The Dimension of KL-Stochastic Sequences

There exists an interesting connection between the asymptotic complexity of sequences and Hausdorff dimension. Hausdorff dimension is defined via Hausdorff measures, and similar to Lebesgue measure, one can define effective versions of them. This leads to the concept of *constructive dimension*, first introduced by Lutz [6], which can equivalently defined in terms of prefix-free Kolmogorov complexity K.

Theorem 16. *The constructive dimension* $\dim_1 A$ *of a sequence* A *is given by*

$$\dim_1 A = \liminf_{n\to\infty} \frac{K(A\upharpoonright_n)}{n}. \tag{4}$$

Note that C, plain Kolmogorov complexity, and K differ by at most $\log(|x|)$, so in Theorem 16 one can replace K by C. Theorem 16 was proven in the presented form by Mayordomo [8], but much of it was already implicit in earlier work by Ryabko [14, 15], Staiger [17, 18], and Cai and Hartmanis [3]. For more on constructive dimension see Reimann [13].

Muchnik [11] refuted a conjecture by Kolmogorov (who asserted that there exists a KL-stochastic sequence A such that $K(A\upharpoonright_n) = \mathrm{O}(\log n)$) by showing that, if A is KL-stochastic, then $\limsup_{n\to\infty} K(A\upharpoonright_n)/n = 1$. In the following, we are going to strengthen this result by showing that $\dim_1 A = 1$ for any KL-stochastic sequence A.

This relates to a result of Ryabko [16], who observed that the probabilistic argument for the construction of KL-stochastic sequences yields with probability 1 a sequence that has constructive dimension 1.

The proof of Theorem 22 bears some similarities to the proof of Theorem 8, where it has been shown that any KL-random sequence has arbitrarily dense subsequences that are Martin-Löf random. We will need the following Proposition, which is a slightly generalized version of a corresponding result by Muchnik et al. [11]. The proof of the proposition is omitted.

Proposition 17. *For any rational* $\alpha < 1$ *there is a natural numbers* k_α *and a rational* $\varepsilon_\alpha > 0$ *such that the following holds. Given an index for a computable martingale* d *with initial capital* 1*, we can effectively find indices for computable monotonic selection rules* $s_1, \ldots, s_{2k_\alpha}$ *such that for all words* w *where*

$$d(w) \geq 2^{(1-\alpha)|w|} \tag{5}$$

there is an index i *such that the selection rule* s_i *selects from* w *a finite sequence of length at least* $\varepsilon_\alpha|w|$ *such that the ratio of* 0*'s and the ratio of* 1*'s in this finite sequence differ by at least* ε_α*.*

Definition 18. *Let* α *be a rational. A word* w *is called* α-compressible *if* $\mathrm{K}(w) \leq \alpha|w|$*.*

Remark 19. Given a rational $\alpha < 1$ and a finite set D of α-compressible words, we can effectively find an index for a computable martingale d with initial capital 1 such that for all $w \in D$ we have $d(w) \geq 2^{(1-\alpha)|w|}$.

For a proof, let d_w be the martingale that starts with initial capital $2^{-\alpha|w|}$ and plays a doubling strategy along w, i.e., always bets all its capital on the next bit being the same as the corresponding bit of w; then we have in particular $d_w(w) = 2^{(1-\alpha)|w|}$.

Let d be the sum of the martingales d_w over all words $w \in D$, i.e., betting according to d amounts to playing in parallel all martingales d_w where $w \in D$. Obviously $d(v) \geq d_w(v)$ for all words v and all $w \in D$, so it remains to show that the initial capital of d does not exceed 1. The latter follows because every $w \in D$ is α-compressible, i.e., can be coded by a prefix-free code of length at most $\alpha|w|$, hence the sum of $2^{-\alpha|w|}$ over all $w \in D$ is at most 1.

Lemma 20. *Let $A = A_1 \oplus A_2$ be KL-stochastic. Then one of the sequences A_1 and A_2 has constructive dimension* 1.

Proof. For a proof by contradiction, assume that the consequence of the lemma is false, i.e., that there is some rational number $\alpha_0 < 1$ such that A_1 and A_2 both have constructive dimension of at most α_0. Pick rational numbers α_1 and α such that $\alpha_0 < \alpha_1 < \alpha < 1$. By Theorem 16, for $r = 1, 2$, there are arbitrarily large prefixes w of A_r that are α_1-compressible, i.e., $\mathrm{K}(w) \leq \alpha_1|w|$. We argue next that for any m there are arbitrarily large intervals I with $\min I = m$ such that the restriction w of A_r to I is α-compressible.

Let $w_0, w_1, \ldots$ be an effective enumeration of all α-compressible words w. For the scope of this proof, say a word w is a *subword of X at m* if

$$w = X(m)X(m+1)\ldots X(m+|w|-1)\ .$$

Let ε_α be the constant from Proposition 17.

Claim 1. For $r = 1, 2$, the function g_r defined by

$$g_r(m) = \min\{i\colon w_i \text{ is a subword of } A_r \text{ at } m \text{ and } |w_i| > \frac{2}{\varepsilon_\alpha^2} m\}$$

is total.

Proof. There are infinitely many α_1-compressible prefixes v of A_r. Given any such prefix of length at least m, let u and w be the words such that $v = uw$ and $|u| = m$. Then we have

$$\mathrm{K}(v) \leq^+ \mathrm{K}(w) + 2\log m \leq \alpha_1|v| + 2\log m = \alpha|w|\left(\frac{\alpha_1}{\alpha}\frac{|v|}{|w|} + \frac{2\log m}{\alpha|w|}\right)\ ,$$

where the expression in brackets goes to $\alpha_1/\alpha < 1$ when the length of w goes to infinity. As a consequence, we have $\mathrm{K}(w) \leq \alpha|w|$ for all such words w that are long enough, hence by assumption on A for any m and t there is a word w_i and an index i as required in the definition of $g_r(m)$. □

Let $m_0 = 0$ and for all $t > 0$, let

$$m_{t+1} = m_t + \max\{|w_i| \colon i \leq \max\{g_1(m_t), g_2(m_t)\}\} \, .$$

In the following, we assume that there are infinitely many t where

$$g_1(m_t) \leq g_2(m_t) \, ; \tag{6}$$

we omit the essentially identical considerations for the symmetric case where there are infinitely many t such that $g_1(m_t) \geq g_2(m_t)$. Let

$$D_t = \{w_0, w_1, \ldots, w_{g_2(m_t)}\}$$

Claim 2. There are infinitely many t such that some word in D_t is a subword of A_1 at m_t.

Proof. By definition of $g_1(m_t)$, the word $w_{g_1(m_t)}$ is a subword of A_1 at m_t, where this word is in D_t for each of the infinitely many t such that $g_1(m_t)$ is less than or equal to $g_2(m_t)$. □

Claim 3. Given D_t and m_t, we can compute an index for a monotonic computable selection rules $s(t)$ that scans only bits of the form

$$A_1(m_t), A_1(m_t + 1), \ldots, A_1(m_{t+1} - 1)$$

of A such that for infinitely many t the selection rule $s(t)$ selects from these bits a finite sequence of length at least $2m_t/\varepsilon_\alpha$ where the ratios of 0's and of 1's in this finite sequence differ by at least ε_α.

Proof. By Proposition 17 and Remark 19, from the set D_t we can compute indices for monotonic computable selection rules $s_1, \ldots, s_{2k_\alpha}$ such that for each $w \in D_t$ there is an index i such that the selection rule s_i selects from w a finite sequence of length at least $\varepsilon_\alpha |w|$ such that the ratio of 0's and 1's in this finite sequence differ by at least ε_α. Any word $w \in D_t$ has length of at least $2m_t/\varepsilon_t^2$, hence the selected finite sequence has length of at least $2m_t/\varepsilon_\alpha$. Furthermore, by Claim 2, there are infinitely many t such that some $w \in D_t$ is a subword of A_1 at m_t, and among the corresponding indices i some index i_0 between 1 and $2k_\alpha$ must appear infinitely often. So it suffices to let for any t the selection rule $s(t)$ be equal to the i_0th selection rule from the list of selection rules computed from D_t. □

Now we construct an non-monotonic computable selection rule s that witnesses that A is not KL-stochastic. The selection rule s works in stages $t = 0, 1, \ldots$ and scans during stage t the bits of A that correspond to bits of the form

$$A_1(y) \quad \text{and} \quad A_2(y), \qquad \text{where } m_t \leq y < m_{t+1} \, .$$

At the beginning of stage t, the value of $g_2(m_t)$ and the set D_t is computed as follows. Successively for $i = 0, 1, \ldots$, check whether w_i is a subword of A_2 at m_t by scanning all the bits

$$A_2(m_t), \ldots, A_2(m_t + |w_i| - 1)$$

of A that have not been scanned so far, until eventually the index i equal to $g_2(m_t)$ is found, i.e., until we find some minimum i such that w_i is a subword of A_2 at m_t. Observe that by definition of m_{t+1}, the index i is found while scanning only bits of the form $A_2(y)$ where $y < m_{t+1}$. Next the selection rule s scans and selects the bits $A_1(m_t), A_1(m_t + 1), \ldots$ according to the selection rule s_{i_0} as in Claim 3; recall that this selection rule can be computed from D_t. Finally, stage t is concluded by computing m_{t+1} from $g_1(t)$ and $g_2(t)$, where $g_1(t)$ is obtained like $g_2(t)$, i.e., in particular, the computation of m_{t+1} only requires to scan bits of the form $A_r(y)$ where $y < m_{t+1}$.

By Claim 2 there are infinitely many t such that some $w \in D_t$ is a subword of A_1 at m_t. By choice of $s(t)$ and definition of s, for each such t the selection rule s selects during stage t a finite sequence of length at least $2m_t/\varepsilon_\alpha$ where the ratios of 0's and 1's in this finite sequence differ by at least ε_α. Consequently, the at most m_t bits of A that might have been selected by s before stage t are at most a fraction of $\varepsilon_\alpha/2$ of the bits selected during stage t, hence with respect to all the bits selected up to stage t the ratios of 0's and 1's differ by at least $\varepsilon_\alpha/2$. This contradicts the fact that A is KL-stochastic, hence our assumption that A_1 and A_2 both have constructive dimension strictly less than 1 is wrong. □

Lemma 21. *If $Z \subseteq \omega$ is computable, infinite, co-infinite, with density $\delta = \delta_Z$. Then it holds for any sequences A, B,*

$$\dim_1 B \oplus_Z A \geq \delta \dim_1 A + (1 - \delta) \dim_1^A B. \tag{7}$$

The proof of Lemma 21 is omitted due to space considerations.

Theorem 22. *If R is KL-stochastic, then* $\dim_1 R = 1$.

Proof. The proof is rather similar to the proof of Theorem 8, in particular, we use the notation N_w from there. It suffices to show that there are words $w_1 \sqsubseteq w_2 \sqsubseteq \ldots$ such that for all i, we have $|w_i| = i$ and

$$\dim_1 R_i = 1\,, \qquad \text{where } R_i = R \restriction_{\overline{\mathrm{N}_{w_i}}}\,;$$

the theorem then follows by Lemma 21 and because for any word w, the set $\overline{\mathrm{N}_w}$ has density $1 - 1/2^{|w|}$.

The w_i are defined inductively. For a start, observe that by Lemma 20 for $r_1 = 0$ or for $r_1 = 1$ the sequence $R \restriction_{\mathrm{N}_r}$ has constructive dimension 1; pick r_1 such that the latter is true and let $w_1 = 1 - r_1$. For $i > 1$, let w_i be defined as follows. By an argument similar to the proof of Proposition 4, the sequence $R \restriction_{\mathrm{N}_w}$ is KL-stochastic relative to R_{i-1}, hence by a relativized version of Lemma 20, for $r_i = 0$ or for $r_i = 1$ the sequence $R \restriction_{\mathrm{N}_{wr}}$ has constructive dimension 1 relative to R_w; pick r_i such the latter is true and let $w_i = w(1 - r_i)$.

It remains to show by induction on i that all the sequences R_i have constructive dimension 1. For $i = 1$, this is true by construction, while the induction step follows according to the choice of the w_i and due to Lemma 21 by an argument similar to the corresponding part of the proof of Theorem 8; details are left to the reader. □

Acknowledgments. We are grateful to Klaus Ambos-Spies, Rod Downey, Antonín Kučera, Steffen Lempp, Jack Lutz, Boris Ryabko, and Ted Slaman for helpful discussion.

We would also like to thank the anonymous referees for many helpful comments and suggestions.

References

1. Ambos-Spies, K., Kučera, A.: Randomness in computability theory. In: Computability theory and its applications (Boulder, CO, 1999). Volume 257 of Contemp. Math. Amer. Math. Soc., Providence, RI (2000) 1–14
2. Buhrman, H., van Melkebeek, D., Regan, K.W., Sivakumar, D., Strauss, M.: A generalization of resource-bounded measure, with application to the BPP vs. EXP problem. SIAM J. Comput. **30** (2000) 576–601
3. Cai, J.Y., Hartmanis, J.: On Hausdorff and topological dimensions of the Kolmogorov complexity of the real line. J. Comput. System Sci. **49** (1994) 605–619
4. Downey, R., Hirschfeldt, D., Nies, A., Terwijn, S.: Calibrating randomness. Bulletin of Symbolic Logic, to appear
5. Li, M., Vitányi, P.: An introduction to Kolmogorov complexity and its applications. Graduate Texts in Computer Science. Springer, New York (1997)
6. Lutz, J.H.: Gales and the constructive dimension of individual sequences. In: International Colloquium on Automata, Languages and Programming (Geneva, 2000). Springer, Berlin (2000) 902–913
7. Martin-Löf, P.: The definition of random sequences. Information and Control **9** (1966) 602–619
8. Mayordomo, E.: A Kolmogorov complexity characterization of constructive Hausdorff dimension. Inform. Process. Lett. **84** (2002) 1–3
9. Merkle, W.: The Kolmogorov-Loveland stochastic sequences are not closed under selecting subsequences. J. Symbolic Logic **68** (2003) 1362–1376
10. Merkle, W., Miller, J., Nies, A., Reimann, J., Stephan, F.: Kolmogorov-Loveland randomness and stochasticity. Annals of Pure and Applied Logic, to appear.
11. Muchnik, A.A., Semenov, A.L., Uspensky, V.A.: Mathematical metaphysics of randomness. Theoret. Comput. Sci. **207** (1998) 263–317
12. Odifreddi, P.: Classical recursion theory. North-Holland, Amsterdam (1989)
13. Reimann, J.: Computability and fractal dimension. Doctoral Dissertation, Universität Heidelberg, Heidelberg, Germany (2004)
14. Ryabko, B.Y.: Coding of combinatorial sources and Hausdorff dimension. Sov. Math. Dokl. **30** (1984) 219–222
15. Ryabko, B.Y.: Noiseless coding of combinatorial sources, Hausdorff dimension and Kolmogorov complexity. Probl. Information Transmission **22** (1986) 170–179
16. Ryabko, B.Y.: Private communication (April 2003)
17. Staiger, L.: Kolmogorov complexity and Hausdorff dimension. Inform. and Comput. **103** (1993) 159–194
18. Staiger, L.: A tight upper bound on Kolmogorov complexity and uniformly optimal prediction. Theory of Computing Systems **31** (1998) 215–229
19. Uspensky, V.A., Semenov, A.L., Shen′, A.K.: Can an (individual) sequence of zeros and ones be random? Russian Math. Surveys **45** (1990) 121–189
20. Van Lambalgen, M.: Random sequences. Doctoral Dissertation, Universiteit van Amsterdam, Amsterdam, Netherlands (1987)

Information Theory in Property Testing and Monotonicity Testing in Higher Dimension*

Nir Ailon and Bernard Chazelle

Department of Computer Science, Princeton University, Princeton NJ, USA
{nailon, chazelle}@cs.princeton.edu

Abstract. In general property testing, we are given oracle access to a function f, and we wish to randomly test if the function satisfies a given property P, or it is ε-far from having that property. In a more general setting, the domain on which the function is defined is equipped with a probability distribution, which assigns different weight to different elements in the distance function. This paper relates the complexity of testing the monotonicity of a function over the d-dimensional cube to the Shannon entropy of the underlying distribution. We provide an improved upper bound on the property tester query complexity and we finetune the exponential dependence on the dimension d.

1 Introduction

In general property testing [4,7,9,13], we are given oracle access to a function f, and we wish to randomly test if the function satisfies a given property P, or it is ε-far from having that property. By ε-far we mean, that any function g that has the property P differs from f in at least ε-fraction places. We allow the property tester to err with at most constant probability, say $1/3$ (in this paper we assume only one-sided error). In many interesting cases, this relaxation allows the tester to query only a sublinear portion of the input f, which is crucial when the input is a giant dataset.

The *query complexity* of the property is the minimal number of f-queries performed by a tester for that property (although the classical "number of operations" quantity can be considered too). A query to a function can be viewed as a quantity of information, which gives rise to the relation between property testing and information complexity [4], which will be made more precise in what follows.

An interesting ramification of property testing problems [4,5,10] generalizes the definition of distance between two functions: Instead of defining the distance between f and g as the fractional size of the set $\{x \mid f(x) \neq g(x)\}$, we attach a probability distribution $\mathcal{D}$ to the function domain, and define

$$\mathrm{dist}(f,g) = \Pr(\{x \mid f(x) \neq g(x)\}).$$

* This work was supported in part by NSF grant CCR-0306283 and ARO Grant DAAH04-96-1-0181.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 434–447, 2005.

The "old" definition reduces to the case $\mathcal{D} = \mathcal{U}$ (the uniform distribution). This definition allows assignment of importance weights to domain points. It also allows property testers to deal with functions defined on infinite domains, though it may be necessary to assume additional structure (for example, measurability of f). Such functions arise when dealing with natural phenomena, like the temperature as a function of location and time. Of course in these cases we couldn't read the entire input even if we had unlimited resources.

The distribution should not be considered as part of the problem, but rather as a parameter of the problem. Fischer [4] distinguishes between the case where $\mathcal{D}$ is known to the tester, and the case where it is not known. The latter is known as the "distribution-free" case [10]. In the distribution-free case, the property tester is allowed to sample from the distribution (but it does not know the probabilities). The main techniques developed in this work will be used for the distribution-known case, but we will also show an application to the distribution-free case.

The following question motivated the results in this paper: what happens when the distribution $\mathcal{D}$ is uniform on a strict subset S of the domain, and zero outside S? Intuitively, the "effective" domain is smaller, and therefore testing the property should be simpler. For general distributions, a natural measure of the "size" of the effective domain is the Shannon entropy H of $\mathcal{D}$. In this paper we show a connection between the quantity H and the query complexity, which further supports the connection between property testing and information theory.

One interesting, well-studied property is monotonicity [2, 3, 4, 6, 8, 10, 11, 12]. A real function f over a poset $\mathcal{P}$ is monotone if any $x, y \in \mathcal{P}$ such that $x \leq y$ satisfy $f(x) \leq f(y)$. In this paper we assume that $\mathcal{P}$ is the d-dimensional cube $[n]^d$, with the order: $(x_1, \ldots, x_d) \leq (y_1, \ldots, y_d)$ if $x_i \leq y_i$ for all $i = 1, \ldots, d$.

Halevy and Kushilevitz [10] describe a property tester with query complexity $O(\frac{2^d \log^d n}{\varepsilon})$ in the distribution-free case. In [11] they show a property tester with query complexity $O(\frac{d4^d \log n}{\varepsilon})$, for the special case of known uniform distribution ($\mathcal{D} = \mathcal{U}$). If d is fixed, this result improves a result by Dodis et al. [2], who describe a property tester with query complexity $O(\frac{d^2 \log^2 n}{\varepsilon})$ (For large d, n must be doubly-exponential in d for Halevy-Kushilevitz's result to be better than that of Dodis et al.). The main result of our paper is as follows:

Theorem 1. *Let $\mathcal{D}$ be a (known) distribution on $[n]^d$ with independent marginal distributions (in other words, $\mathcal{D}$ is a product $\mathcal{D}_1 \times \cdots \times \mathcal{D}_d$ of distributions $\mathcal{D}_i$ on $[n]$). Let H be the Shannon entropy of $\mathcal{D}$. Then there exists a property tester for functions over $([n]^d, \mathcal{D})$ with expected query complexity $O(\frac{2^d H}{\varepsilon})$.*

In the special case $\mathcal{D} = \mathcal{U}$, this theorem improves Halevy and Kushilevitz's result by replacing the 4^d with 2^d (because then $H = d \log n$). It also generalizes previous work to any product distribution and gives a first evidence of the connection between property testing and the Shannon entropy of the underlying distribution.

Although this paper discusses mainly the *known* distribution case, the techniques developed here can be used to show the following:

Theorem 2. *Let $\mathcal{D}$ be an (unknown) distribution on $[n]^d$ with independent marginal distributions. Then there exists a property tester for functions over $([n]^d, \mathcal{D})$ with query complexity $O(\frac{d2^d \log n}{\varepsilon})$.*

Note that although Theorem 2 assumes that the distribution $\mathcal{D}$ is unknown, it will in fact be implicitly assumed by the property tester that $\mathcal{D}$ is a product of d marginal distributions. This is a relaxation of the notion of distribution-free property testing: the distribution is assumed to belong to some big family of distributions. This improves Halevy and Kushilevitz's $O(\frac{\log^d n 2^d}{\varepsilon})$ property tester [10] for this relaxed version (in their result, however, nothing is assumed about the distribution $\mathcal{D}$).

The rest of the paper is organized as follows: Section 2 starts with preliminaries and definitions, Section 3 proves Theorem 1 for the case $([n], \mathcal{D})$, Section 4 proves Theorem 1 for the case $([n]^d, \mathcal{U})$, and Section 5 completes the proof of Theorem 1. In Section 6 we prove Theorem 2. In Sections 7 and 8 we prove two important technical lemmas. Section 9 discusses future work and open problems.

2 Preliminaries

Let f be a real valued function on the domain $[n]^d$, with a probability distribution $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_d$. Assume that D_i assigns probability p_j^i to $j \in [n]$, and therefore $\mathcal{D}$ assigns probability $\prod_{k=1}^d p_{i_k}^k$ to $(i_1, i_2, \ldots, i_d)$.

Definition 1. *The distance of f from monotonicity, denoted by ε, is defined as $\min \Pr_{\mathcal{D}}(\{f \neq g\})$, where the minimum is over all monotone functions g.*

Definition 2. *The i-th axis-parallel order $\leq_i$ on $[n]^d$ is defined as $(x_1, \ldots, x_d) \leq_i (y_1, \ldots, y_d)$ if $x_i \leq y_i$ and $x_j = y_j$ for $j \neq i$.*

Definition 3. *The i-th axis-parallel distance of f to monotonicity, denoted by ε_i, is $\min \Pr_{\mathcal{D}}(\{f \neq g\})$, where the minimum is over all functions g that are monotone with respect to $\leq_i$.*

It is a simple observation that f is monotone on $[n]^d$ if and only if it is monotone with respect to $\leq_i$ for each $i = 1, \ldots, d$.

Definition 4. *An integer pair $\langle i, j \rangle$ (for $i, j \in [n]^d$, $i \leq j$) is a* violating pair *if $f(i) > f(j)$. We say that "j is in violation with i" or "i is in violation with j" in this case.*

Although this work deals with the finite domain case, it will be useful in what follows to consider the continuous cube I^d, where $I = \{x \in \mathbb{R} \mid 0 \leq x < 1\}$. The probability distribution is the Lebesgue measure, denoted by μ. The distance between two measurable functions $\alpha, \beta : I^d \to \mathbb{R}$ is $\mu(\{\alpha \neq \beta\})$ (the set $\{\alpha \neq \beta\}$ is measurable). The distance of α from monotonicity is $\inf \operatorname{dist}(\alpha, \beta)$ where the infimum is over all monotone functions β.

For $i = 1, \ldots, d$, consider the following sequence of subintervals covering I:

$$\Delta_1^i = [0, p_1^i), \Delta_2^i = [p_1^i, p_1^i + p_2^i), \ldots, \Delta_n^i = [1 - p_n^i, 1).$$

For a number $x \in I$, define $\mathrm{int}_i(x) = j$ if $x \in \Delta_j^i$, that is, x belongs to the j-th interval induced by $\mathcal{D}_i$. If $d = 1$ we omit the superscript and simply write Δ_j and $\mathrm{int}(x)$. It is obvious that if x is distributed uniformly in I, then $\mathrm{int}_i(x)$ is distributed according to $\mathcal{D}_i$.

For a given $f : [n]^d \to \mathbb{R}$, denote by $\tilde{f} : I^d \to \mathbb{R}$ the function

$$\tilde{f}(x_1, \ldots, x_d) = f(\mathrm{int}_1(x_1), \mathrm{int}_2(x_2), \ldots, \mathrm{int}_d(x_d)).$$

The function $\tilde{f}$ is constant on rectangles of the form $\Delta_{i_1}^1 \times \cdots \times \Delta_{i_d}^d$, for any $i_1, \ldots, i_d \in [n]$. Moreover, any function $\alpha : I^d \to \mathbb{R}$ which is constant on these rectangles can be viewed as a function over $[n]^d$. The following lemma formalizes an intuitive connection between $([n]^d, \mathcal{D})$ and $(I^d, \mathcal{U})$. The proof is postponed to Section 7.

Lemma 1. *The distance $\tilde{\varepsilon}$ of $\tilde{f}$ from monotonicity in I^d (with respect to the Lebesgue measure) equals the distance ε of f from monotonicity in $[n]^d$ (with respect to $\mathcal{D}$). This is also true with respect to the axis-parallel orders $\leq_i$.*

Finally, we give a precise definition of a property tester:

Definition 5. *An ε-property tester for monotonicity (or, ε-monotonicity tester) is a randomized algorithm that, given $f : [n]^d \to \mathbb{R}$, outputs "ACCEPT" with probability* 1 *if f is monotone, and "REJECT" with probability at least* 2/3 *if f is ε-far from being monotone w.r.t. a fixed distribution $\mathcal{D}$. In the distribution-known case, the probabilities of $\mathcal{D}$ are known. In the distribution-free case they are unknown, but the property tester can sample from $\mathcal{D}$.*

3 A Property Tester for $([n], \mathcal{D})$

The algorithm is a generalization of an algorithm presented in [10]. Let $f : [n] \to \mathbb{R}$ be the input function. We need a few definitions and lemmas.

Definition 6. *For a violating pair $\langle i, j \rangle$ we say that i is* active *if*

$$\Pr(\textit{ in violation with } i \mid [i+1, j]) \geq 1/2.$$

Similarly, j is active *if* $\Pr(\textit{ in violation with } j \mid [i, j-1]) \geq 1/2$.

In other words, an active integer in a violating pair $\langle i, j \rangle$ is also in violation with an abundance of elements in the interval $[i, j]$.

Definition 7. *For a violating pair $\langle i, j \rangle$, we say that i is* strongly *active if it is active and $p_i \leq Pr([i+1, j])$. Similarly, j is* strongly *active if it is active and $p_j \leq Pr([i, j-1])$.*

Lemma 2. *If $\langle i, j\rangle$ is a violating pair, then either i is strongly active or j is strongly active.*

Proof. It is immediate that for any $i < k < j$, either $\langle i, k\rangle$ or $\langle k, j\rangle$ is a violating pair. So either i or j is in violation with at least half the weight of the integers $[i + 1, j - 1]$. This proves that either i or j is active. So assume i is active but *not* strongly active. This means that $p_i > Pr([i + 1, j])$. But this would imply that j is strongly active. Indeed, p_i is greater than half of $Pr([i, j - 1])$, and i is in violation with j, so j is active. But $p_j < p_i$ so j is strongly active. □

Lemma 3. *Let J be the collection of strongly active integers from all violating pairs of f. Then $Pr(J) \geq \varepsilon$.*

Proof. Actually, any collection J of at least one integer from each violating pair has this property. Proof of this simple fact can be found in [10]. □

To describe the algorithm, we need another piece of notation. For $x \in I$, let $\mathrm{left}(x)$ denote the left endpoint of the interval $\Delta_{\mathrm{int}(x)}$, and similarly let $\mathrm{right}(x)$ denote its right endpoint.

The following algorithm is an ε-property tester for monotonicity of f, with expected query complexity $O(\frac{H+1}{\varepsilon})$. We show how to eliminate the added $1/\varepsilon$ shortly.

monotonicity-test $(f, \mathcal{D}, \varepsilon)$

```
1 repeat O(ε^{-1}) times
2     choose random x ∈ I
        set δ ← p_{int(x)}
3       set r ← right(x)
4       while r + δ ≤ 2
5           choose random y ∈_U [r, min{r + δ, 1}]
6           if f(int(x)) > f(int(y))
7             then output REJECT
            δ ← 2δ
        set δ ← p_{int(x)}
        set l ← left(x)
8       while l − δ ≥ −1
            choose random y ∈_U [max{l − δ, 0}, x]
            if f(int(y)) > f(int(x))
              then output REJECT
            set δ ← 2δ
  output ACCEPT
```

We first calculate the expected running time of **monotonicity-test**. The number of iterations of the internal *while* loops (lines 4,8) is clearly at most

$\log(2/p_{\mathrm{int}(x)})$ (all the logarithms are taken in base 2 in this paper). Clearly, $\mathbf{E}_{x\in_{\mathcal{U}} I}[\log(2/p_{\mathrm{int}(x)})] = \mathbf{E}_{i\in\mathcal{D}}[\log(2/p_i)] = H + 1$. We prove correctness of the algorithm. Obviously, if f is monotone then the algorithm returns "ACCEPT". Assume that f is ε-far from being monotone. By lemma 3, with probability at least ε, the random variable x chosen in line 2 satisfies $\mathrm{int}(x) \in J$. This means that $i = \mathrm{int}(x)$ is strongly active with respect to a violating pair $\langle i, j\rangle$ or $\langle j, i\rangle$. Assume the former case (a similar analysis can be done for the latter). So i is in violation with at least half the weight of $[i + 1, j]$, and also $p_i \leq Pr([i + 1, j])$. Consider the intervals $[r, r + p_i 2^t]$ for $t = 0, 1, 2, \ldots$ with r as in line 3. For some t, this interval "contains" the corresponding interval $[i + 1, j]$ (i.e. $\Delta_{i+1} \cup \cdots \cup \Delta_j$), but $p_i 2^t$ is at most twice $\Pr([i + 1, j])$. The latter by virtue of i being *strongly* active. For this t, with probability at least $1/2$ the y chosen in line 5 is in $[i + 1, j]$. In such a case, the probability of y being a witness of nonmonotonicity in lines 6-7 is at least $1/2$, by virtue of i being *active*. Summing up, we get that the probability of outputting "REJECT" in a single iteration of the loop in line 1 is at least $\varepsilon/4$. Repeating $O(\varepsilon^{-1})$ times gives a constant probability.

We note that the additive constant 1 in the query complexity can be eliminated using a simple technical observation. Indeed, notice that, for x chosen in line 2, if $p_{\mathrm{int}(x)} > 1/2$ then x cannot be strongly active by definition, and therefore that iteration can be aborted without any query. If $p_{\mathrm{int}(x)} \leq 1/2$ then we can eliminate one iteration from the while loops by initializing $\delta = 2p_{\mathrm{int}(x)}$ instead of $\delta = p_{\mathrm{int}(x)}$ and by slightly decreasing the probability of success in each iteration of the *repeat* loop. This gets rid of the additive constant, and concludes the proof of Theorem 1 in the $([n], \mathcal{D})$ case.

4 A Property Tester for $([n]^d, \mathcal{U})$

Let $f : [n]^d \to \mathcal{U}$ denote the input function. For a dimension $j \in [d]$ and integers $i_1, \ldots, \hat{i}_j, \ldots, i_d \in [n]$, let $f^j_{i_1,\ldots,\hat{i_j},\ldots,i_d}$ denote the one-dimensional function obtained by restricting f to the line $\{i_1\} \times \cdots \times \{i_{j-1}\} \times [n] \times \{i_{j+1}\} \times \cdots \times \{i_d\}$.

highdim-mon-uniform-test (f, ε)

```
  repeat O(ε^{-1} d 2^d) times
1     choose random dimension j ∈ [d]
2     choose random i_1, ..., î_j, ..., i_d ∈ [n]
3     run one iteration of repeat loop of
        monotonicity-test(f^j_{i_1,...,î_j,...,i_d}, U, *)
  output ACCEPT
```

To prove that the above algorithm is an ε-monotonicity tester for f, we will need the following lemma. It is an improved version of a theorem from [11], with 2^d replacing the 4^d on the right hand side. Recall Definition 2 of ε_i.

Lemma 4. $\sum_{i=1}^{d} \varepsilon_i \geq \varepsilon/2^{d+1}$.

The correctness of **highdim-mon-uniform-test** is a simple consequence of Lemma 4. If f is monotone, then the algorithm returns "ACCEPT" with probability 1. So assume f is ε-far from monotonicity. By Lemma 4, the restricted one-dimensional function $f^j_{i_1,\ldots,\hat{i}_j,\ldots,i_d}$ chosen in line 3 has expected distance of at least $\gamma = \frac{1}{d}\sum \varepsilon_i \geq \frac{1}{d}\varepsilon/2^{d+1}$ from monotonicity, in each iteration of the *repeat* loop. A single iteration of **monotonicity-test** has an expected success probability of $\Omega(\gamma)$ by the analysis of the previous section. Repeating $O(\varepsilon^{-1}d2^d)$ times amplifies the probability of success to any fixed constant. As for the query complexity, line 3 makes $O(\log n)$ queries, which is the entropy of the uniform distribution on $[n]$. So the entire query complexity is $O(\varepsilon^{-1}2^d d \log n) = O(\varepsilon^{-1}2^d H)$, as required. It remains to prove Lemma 4:

Proof. For $i = 1, \ldots, d$, let B_i denote a minimal subset of $[n]^d$ such that f can be changed on B_i to get a monotone function with respect to $\leq_i$. So $|B_i| = n^d\varepsilon_i$. Let $B = \cup_{i=1}^{d} B_i$. So $|B| \leq \sum \varepsilon_i [n]^d$. Let $\chi_B : [n]^d \to \{0,1\}$ denote the characteristic function of B: $\chi_B(x) = 1$ if $x \in B$, otherwise 0. We define operators Ψ_L and Ψ_R on $\{0,1\}$ functions over $[n]$ as follows:

$$(\Psi_L v)(i) = \begin{cases} 1 & \text{if there exists } j \in [1,i] \text{ s.t. } \sum_{k=j}^{i} v(k) \geq (i-j+1)/2 \\ 0 & \text{otherwise} \end{cases}$$

$$(\Psi_R v)(i) = \begin{cases} 1 & \text{if there exists } j \in [i,n] \text{ s.t. } \sum_{k=i}^{j} v(k) \geq (j-i+1)/2 \\ 0 & \text{otherwise} \end{cases}$$

Given a $\{0,1\}$-function over $[n]^d$, we define operators $\Psi_L^{(i)}$ (resp. $\Psi_R^{(i)}$) for $i = 1, \ldots, d$ by applying Ψ_R (resp. Ψ_L) independently on one-dimensional lines of the form $\{x_1\} \times \cdots \times \{x_{i-1}\} \times [n] \times \{x_{i+1}\} \times \cdots \times \{x_d\}$. Finally, for $i = 1, \ldots, d$ we define the functions $\varphi_L^{(i)}, \varphi_R^{(i)} : [n]^d \to \{0,1\}$ as follows:

$$\varphi_L^{(i)} = \left(\Psi_L^{(i)} \circ \Psi_L^{(i+1)} \circ \cdots \circ \Psi_L^{(d)}\right)\chi_B, \quad \varphi_R^{(i)} = \left(\Psi_R^{(i)} \circ \Psi_R^{(i+1)} \circ \cdots \circ \Psi_R^{(d)}\right)\chi_B \tag{1}$$

Note that $\varphi_L^{(i)} = \Psi_L^{(i)}\varphi_L^{(i+1)}$ and $\varphi_R^{(i)} = \Psi_R^{(i)}\varphi_R^{(i+1)}$. We claim that outside the set $\{\varphi_L^{(1)} = 1\} \cup \{\varphi_R^{(1)} = 1\} \subseteq [n]^d$ the function f is monotone. Indeed, choose $x, y \in [n]^d$ such that $x \leq y$ and $\varphi_L^{(1)}(y) = \varphi_R^{(1)}(x) = 0$. We want to show that $f(x) \leq f(y)$.

Claim 3. *Any $b \in B$ satisfies $\varphi_L^{(i)}(b) = \varphi_R^{(i)}(b) = 1$ for $i = 1, \ldots, d$.*

By the above Claim, $x, y \notin B$. Now consider the two line segments: $S_R = [x_1, y_1] \times \{x_2\} \times \cdots \times \{x_d\}$, $S_L = [x_1, y_1] \times \{y_2\} \times \cdots \times \{y_d\}$. By definition of $\Psi_R^{(1)}$ (resp. $\Psi_L^{(1)}$), the average value of $\varphi_R^{(2)}$ (resp. $\varphi_L^{(2)}$) on S_R (resp. S_L) is less than $1/2$. Therefore, there exists $z_1 \in [x_1, y_1]$ such that $\varphi_R^{(2)}(z_1, x_2, \ldots, x_d) + \varphi_L^{(2)}(z_1, y_2, \ldots, y_d) < 1$. Since these values are in $\{0, 1\}$, we get that

$$\varphi_R^{(2)}(z_1, x_2, \ldots, x_d) = \varphi_L^{(2)}(z_1, y_2, \ldots, y_d) = 0. \tag{2}$$

Denote $x^{(1)} = (z_1, x_2, \ldots, x_d)$ and $y^{(1)} = (z_2, y_2, \ldots, y_d)$. By Claim 3 and (2), both $x^{(1)}$ and $y^{(1)}$ are outside B. Since $x \leq_1 x^{(1)}$ we get that $f(x) \leq f(x^{(1)})$. A similar argument shows that $f(y^{(1)}) \leq f(y)$. We use an inductive argument, using the functions $\varphi_L^{(2)}$ and $\varphi_R^{(2)}$ to show that $f(x^{(1)}) \leq f(y^{(1)})$. The general inductive step generates points $x^{(i)} \leq y^{(i)}$ that agree in the first i coordinates, and such that $\varphi_R^{(i+1)}(x^{(i)}) = \varphi_L^{(i+1)}(y^{(i)}) = 0$ (consequently, $x^{(i)}, y^{(i)} \notin B$). In the base step we will end up with $x^{(d-1)}$ and $y^{(d-1)}$ that differ in their last coordinate only. Therefore, they are $\leq_d$-comparable and $f(x^{(d-1)}) \leq f(y^{(d-1)})$ because $x^{(d-1)}, y^{(d-1)} \notin B$. It remains to bound the size of the set $\{\varphi_L^{(1)} = 1\}$. A similar analysis can be applied to $\{\varphi_R^{(1)} = 1\}$. We claim that $|\{\varphi_L^{(1)} = 1\}| \leq |B|2^d$. This is a simple consequence of the following lemma.

Lemma 5. *Let $v \in \{0,1\}^n$. Then the number of 1's in $\Psi_L v$ is at most twice the number of 1's in v. A similar result holds for Ψ_R.*

To prove this, imagine walking on the domain $[n]$ from 1 to n, and marking integers according to the following rule (assume on initialization that all domain points are unmarked and a counter is set to 0):

If the value of v on the current integer i is 1, then mark i. Also, in this case increase the counter by 1. If $v(i) = 0$ and the counter is > 0, then mark integer i and decrease the counter by 1. Otherwise do nothing.

It is obvious that the number of marked integers is at most twice the number of 1's in v. It is also not hard to show that $(\Psi_L v)(i) = 1$ only if i is marked. Indeed, if $(\Psi_L v)(i) = 1$, then for some $j \leq i$, vector v on integer segment $[j, i]$ has at least as many 1's as 0's. This implies that either $v(i) = 1$ or the counter at i is positive, therefore i is marked. This proves the lemma.

We conclude that the combined size of $\{\varphi_L^{(1)} = 1\}$ and $\{\varphi_R^{(1)} = 1\}$ is at most $|B|2^{d+1}$. This means that f is monotone on a subset of $[n]^d$ of size at least $n^d - |B|2^{d+1}$. It is a simple fact that any monotone function on a subset of $[n]^d$ can be completed to a monotone function on the entire domain (see Lemma 1 [6]). So the distance ε of f from monotonicity is at most $2^{d+1} \sum \varepsilon_i$, as required. □

5 A Property Tester for $([n]^d, \mathcal{D})$

Let $f : [n]^d \to \mathbb{R}$ be the input function, where $[n]^d$ is equipped with a (known) distribution $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_d$. The following algorithm is a monotonicity tester for f.

highdim-monotonicity-test $(f, \mathcal{D}, \varepsilon)$

```
1 repeat O(ε^{-1} d 2^d) times
2     choose random dimension j ∈ [d]
3     choose random (i_1, ..., i_d) ∈_D [n]^d
4     run one iteration of repeat loop of
        monotonicity-test(f^j_{i_1,...,î_j,...,i_d}, D_j, *)
  output ACCEPT
```

Clearly, for $\mathcal{D} = \mathcal{U}$ **highdim-monotonicity-test** is equivalent to **highdim-mon-uniform-test**.

We start with the query complexity analysis. The call to **monotonicity-test** in line 4 has query complexity $O(H_j)$ (the entropy of $\mathcal{D}_j$). Therefore, the expected query complexity in each iteration of the *repeat* loop is $\frac{1}{d}\sum_{j=1}^{d} O(H_j) = \frac{1}{d}O(H)$ (we use the well known identity that the entropy of a product of independent variables is the sum of the individual entropies). Therefore the total running time is $O(\varepsilon^{-1}2^d H)$, as claimed.

We prove correctness. Clearly, if f is monotone then **highdim-monotonicity-test** outputs "ACCEPT" with probability 1. Assume f is ε-far from monotone. In order to lower bound the success probability (outputting "REJECT") of line 4, we want to lower bound the average axis-parallel distances to monotonicity of f, similarly to Lemma 4. In order to do that, we consider the continuous case. Recall the definition of the function $\tilde{f} : I^d \to \mathbb{R}$ from Section 2. Let $\tilde{\varepsilon}$ be its distance from monotonicity w.r.t. the Lebesgue measure, and $\tilde{\varepsilon}_i$ its corresponding axis-parallel distances. We need the following lemma, which is a continuous version of Lemma 4.

Lemma 6. $\sum_{i=1}^{d} \tilde{\varepsilon}_i \geq \tilde{\varepsilon}/2^{d+1}$.

Proof. The proof is basically as that of Lemma 4, with a redefinition of $B_i, B, \chi_B, \Psi_L, \Psi_R, \Psi_L^i, \Psi_R^i, \varphi_L^{(i)}, \varphi_R^{(i)}$. We pick an arbitrarily small $\delta > 0$, and define the set $B_i \subseteq I^d$ as the set $\{f \neq g\}$ for some $\leq_i$-monotone g with distance at most $\tilde{\varepsilon}_i + \delta$ from f (so $\tilde{\varepsilon}_i \leq \mu(B_i) \leq \tilde{\varepsilon}_i + \delta$). Let χ_B be the characteristic function of $B = \cup B_i$. Obviously, $\mu(B) \leq \sum \tilde{\varepsilon}_i + \delta d$. We then define the following continuous versions of Ψ_L, Ψ_R, which are now operators on measurable $\{0,1\}$ functions over I:

$$(\Psi_L v)(x) = \begin{cases} 1 & v(x) = 1 \text{ or there exists } y \in [0,x) \text{ s.t. } \int_y^x v(t)dt \geq \frac{1}{2}(x-y) \\ 0 & \text{otherwise} \end{cases}$$

$$(\Psi_R v)(x) = \begin{cases} 1 & v(x) = 1 \text{ or there exists } y \in (x,1] \text{ s.t. } \int_x^y v(t)dt \geq \frac{1}{2}(y-x) \\ 0 & \text{otherwise} \end{cases}$$

The operator Ψ_L^i (resp. Ψ_R^i) on functions of I^d applies Ψ_L (resp. Ψ_R) on all lines of the form $\{x_1\} \times \cdots \times \{x_{i-1}\} \times I \times \{x_{i+1}\} \times \cdots \times \{x_d\}$. The functions $\varphi_L^{(i)}$ and $\varphi_R^{(i)}$ are defined as in (1). The main observation is that $\mu(\{\varphi_L^{(1)} = 1\}) \leq 2^d \mu(B)$ (similarly, for $\varphi_R^{(1)}$). This is a simple consequence of the following lemma, which is a continuous version of Lemma 5.

Lemma 7. *Let v be a measurable $\{0,1\}$ function defined on I. Then $\int_0^1 (\Psi_L v)(t)dt \leq 2\int_0^1 v(t)dt$. A similar result holds for Ψ_R.*

The mostly technical proof of Lemma 7 can be found in Section 8. The rest of the proof of Lemma 6 continues very similar to that of Lemma 4 and by taking $\delta \to 0$. □

As a result of Lemmas 6 and 1, we have: $\sum \varepsilon_i \geq \varepsilon/2^{d+1}$. This means that the expected one-dimensional distance from monotonicity of $f^j_{i_1,\ldots,\hat{i_j},\ldots,i_d}$ in line 4 (w.r.t. the marginal distribution $\mathcal{D}_j$) is at least $\gamma = \frac{1}{d}\varepsilon/2^{d+1}$. By the analysis of **monotonicity-test**, we know that the probability of outputting "REJECT" in a single iteration of the repeat loop is $\Omega(\gamma)$. Therefore, by repeating $O(1/\gamma)$ times we get constant probability of success. This completes the proof of Theorem 1. □

6 The Distribution-(Almost) Free Case

We prove Theorem 2. Let $f : [n]^d \to \mathbb{R}$ be the input function, where $[n]^d$ is equipped with a distribution $\mathcal{D} = \mathcal{D}_i \times \cdots \times \mathcal{D}_d$, and the marginal distributions $\mathcal{D}_i$ are unknown.

We cannot simply run **highdim-monotonicity-test** on f, because that algorithm expects the argument $\mathcal{D}$ to be the actual probabilities of the distribution. In the distribution-free case, we can only pass an oracle$[\mathcal{D}]$, which is a distribution sampling function. Therefore our new algorithm, **highdim-monotonicity-test-distfree** will take f, oracle$[\mathcal{D}]$ and ε as input.

highdim-monotonicity-test1 $(f, \texttt{oracle}[\mathcal{D}], \varepsilon)$

```
1 repeat O(ε^{-1} d 2^d) times
2     choose random dimension j ∈ [d]
3     choose random (i_1, ..., î_j, ..., i_d) ∈_D [n]^{d-1}
4     run one iteration of repeat loop of
        monotonicity-test1(f^j_{i_1,...,î_j,...,i_d}, oracle[D_j], *)
  output ACCEPT
```

Note that oracle$[\mathcal{D}_j]$ in line 4 is obtained by projecting the output of oracle$[\mathcal{D}]$. Algorithm **monotonicity-test1** is defined to be exactly Halevy-Kushilevitz's 1-dimensional distribution-free monotonicity tester[1] [10]. We omit its description here and refer the reader to [10]. The running time of a single iteration of the repeat loop of **monotonicity-test1** is $O(\log n)$, and the total running time is $O(\varepsilon^{-1}d2^d \log n)$, as required.

Let f' denote the one dimensional function $f^j_{i_1,\ldots,\hat{i_j},\ldots,i_d}$, as chosen in line 4 of **highdim-monotonicity-test1**, and let ε' be its distance from monotonicity w.r.t. $\mathcal{D}_j$. In [10] it is proven that a single repeat-loop iteration of **monotonicity-test1** $(f, \text{oracle}[\mathcal{D}_j], *)$ outputs "REJECT" with probability $\Omega(\varepsilon')$. But we showed in Section 5 that $\mathbf{E}[\varepsilon'] \geq \frac{1}{d}\varepsilon/2^{d+1}$. Repeating lines 2-4 $O(\varepsilon^{-1}d2^d)$ times amplifies this to a constant probability. This concludes the proof of Theorem 2. □

7 Proof of Lemma 1

The direction $\tilde{\varepsilon} \leq \varepsilon$ is clear. It remains to show that $\varepsilon \leq \tilde{\varepsilon}$. Pick an arbitrarily small $\delta > 0$, and let $\tilde{g}$ be some monotone function on I^d with distance at most $\tilde{\varepsilon} + \delta$ to $\tilde{f}$. We are going to replace $\tilde{g}$ with a monotone function g over $[n]^d$ with distance at most $\tilde{\varepsilon} + 2\delta$ to f. To do this, we will make it constant on tiles of the form $\Delta^1_{i_1} \times \Delta^2_{i_2} \times \cdots \times \Delta^d_{i_d}$, paying a price of at most one extra δ. We will do this one dimension at a time.

We show how to do this for the first dimension, and the rest is done similarly. Our goal is to replace $\tilde{g}$ with a monotone function $\tilde{g}^{(1)}$ that has distance at most $\tilde{\varepsilon} + \delta(1 + 1/d)$ from $\tilde{f}$, with the property that it is constant on any line segment of the form $\Delta^1_i \times \{x_2\} \times \cdots \times \{x_d\}$, for any $i \in [n]$ and $x_2, \ldots, x_d \in I$. For every $i \in [n]$, do the following: For every $x_1 \in \Delta^1_i$, consider the restriction of the function $\tilde{g}$ to the $d-1$ dimensional cube $\{x_1\} \times I^{d-1}$. Denote this function by $\tilde{g}_{x_1}(x_2, \ldots, x_d)$. Let $\tilde{\varepsilon}_{x_1}$ denote the distance between $\tilde{g}_{x_1}$ and $\tilde{f}_{x_1}$ (where $\tilde{f}_{x_1}$ is defined similarly to $\tilde{g}_{x_1}$). Let $\gamma = \inf_{x_1 \in \Delta^1_i} \tilde{\varepsilon}_{x_1}$. Pick x_1 such that $\tilde{\varepsilon}_{x_1}$ is at most $\gamma + \delta/d$. We now "smear" the value of $\tilde{g}$ at $(x_1, x_2, \ldots., x_d)$ to $\Delta^1_i \times \{x_2\} \times \cdots \times \{x_d\}$, for all $x_2, \ldots, x_d$. Doing this for all $i = 1, \ldots, n$ produces the function $\tilde{g}^{(1)}$. It is not hard to see that the distance between $\tilde{g}^{(1)}$ and f is at most $\tilde{\varepsilon} + \delta(1 + 1/d)$, and the function $\tilde{g}^{(1)}$ is monotone.

After obtaining $\tilde{g}^{(j)}$, we obtain $\tilde{g}^{(j+1)}$ by repeating the above process for the $(j+1)$-th dimension. It is easy to verify that for $j < d$: (1) If $\tilde{g}^{(j)}$ is monotone then so is $\tilde{g}^{(j+1)}$. (2) If $\tilde{g}^{(j)}$ is constant on $\Delta^1_{i_1} \times \Delta^2_{i_2} \times \cdots \times \Delta^j_{i_j} \times \{x_{j+1}\} \times \cdots \times \{x_d\}$ for all $i_1, \ldots, i_j$ and $x_{j+1}, \ldots, x_d$, then $\tilde{g}^{(j+1)}$ is constant on $\Delta^1_{i_1} \times \Delta^2_{i_2} \times \cdots \times \Delta^{j+1}_{i_{j+1}} \times \{x_{j+2}\} \times \cdots \times \{x_d\}$ for all $i_1, \ldots, i_{j+1}$ and $x_{j+2}, \ldots, x_d$. (3) If the distance between $\tilde{g}^{(j)}$ and $\tilde{f}$ is at most $\tilde{\varepsilon} + j\delta/d$, then the distance between $\tilde{g}^{(j+1)}$ and $\tilde{f}$ is at most $\tilde{\varepsilon} + (j+1)\delta/d$.

[1] It is called **Algorithm-monotone-1-dim**$_{\mathcal{D}}(f, \varepsilon)$ there.

Therefore, $\tilde{g}^{(d)}$ is monotone, and it is defined over $[n]^d$ (because it is constant over $\Delta^1_{i_1} \times \cdots \times \Delta^d_{i_d}$). Denote the equivalent function over $([n]^d, \mathcal{D})$ by g. The monotone function g has distance at most $\tilde{\varepsilon} + 2\delta$ from f. The set of possible distances between functions over $([n]^d, \mathcal{D})$ is finite, therefore by choosing δ small enough we obtain a function g which has distance exactly $\tilde{\varepsilon}$ from f. This concludes the proof. □

8 Proof of Lemma 7

Let B denote the set $\{x|v(x) = 1\}$, and C denote $\{x|(\Psi_L v)(x) = 1\}$. We want to show that $\mu(C) \leq 2\mu(B)$. It suffices to show that for any $\varepsilon > 0$, $\mu(C) \leq (2+\varepsilon)\mu(B)$.

For $y < x$, define $\rho(y,x) = \frac{\int_y^x v(t)dt}{y-x} = \frac{\mu(B \cap [y,x])}{\mu([y,x])}$. That is, $\rho(y,x)$ is the measure of the set $\{v = 1\}$ conditioned on $[y,x]$.

Pick an arbitrary small $\varepsilon > 0$. Let C_ε be the set of points $x \in I$ such that there exists $y < x$ with $\rho(y,x) > 1/2 - \varepsilon$. For $x \in C_\varepsilon$, we say that y is an *ε-witness* for x if $\rho(y,x) > 1/2 - \varepsilon$. We say that y is a *strong ε-witness* for x if for all $z : y < z \leq x$, $\rho(y,z) > 1/2 - \varepsilon$.

We claim that if $x \in C_\varepsilon$, then there exists a strong ε-witness y for x. Assume otherwise. Let y be any ε-witness for x. Since y is not a strong ε-witness for x, there exists $z : y < z < x$ such that $\rho(y,z) \leq 1/2 - \varepsilon$. Let z_0 be the supremum of all such z. Clearly, $y < z_0 < x$ (z_0 cannot be x because then by continuity of ρ we would get $\rho(y,x) \leq 1/2 - \varepsilon$). We claim that z_0 is a strong witness for x. Indeed, if for some $z' : z_0 < z' < x$ we had $\rho(z_0, z') \leq 1/2 - \varepsilon$, then it would imply $\rho(y,z') \leq 1/2 - \varepsilon$, contradicting our choice of the supremum.

For all $x \in C_\varepsilon$, let $y(x)$ be the infimum among all strong ε-witnesses of x. We claim that for $x \neq x'$, the intervals $[y(x), x)$ and $[y(x'), x')$ are either disjoint, or $y(x) = y(x')$. Otherwise, we would have, without loss of generality, $y(x) < y(x')$ with both $x, x' > y(x')$. But then any strong ε-witness for x that is strictly between $y(x)$ and $y(x')$ (which exists) is a strong ε-witness for x', contradicting the choice of $y(x')$. Therefore, the set $Y = y(C_\varepsilon)$ (the image of $y(\cdot)$) is countable, and for any $y_0 \in Y$ there exists an $x(y_0) > y_0$ which is the supremum over all $x : x > y_0$ such that $y(x) = y_0$. For two distinct $y_1, y_2 \in Y$, the intervals $[y_1, x(y_1))$ and $[y_2, x(y_2))$ are disjoint. Let $D = \cup_{y \in Y} [y, x(y))$. Clearly, by continuity of ρ, for all $y \in Y$, $\mu([y, x(y))) \leq \frac{\mu([y,x(y)) \cap B)}{1/2-\varepsilon}$. Therefore $\mu(D) \leq \frac{\mu(D \cap B)}{1/2-\varepsilon}$. We also have that $\mu(\bar{D}) = \mu(D)$ (where $\bar{D}$ is the closure of D), because D is a union of countably many intervals. Therefore, $\mu(\bar{D}) \leq \frac{\mu(\bar{D} \cap B)}{1/2-\varepsilon}$. By our previous claim $C_\varepsilon \subseteq \bar{D}$, therefore $\mu(C_\varepsilon) \leq \frac{\mu(\bar{D} \cap B)}{1/2-\varepsilon}$, and thus $\mu(C_\varepsilon \cup (B \backslash \bar{D})) \leq \frac{\mu(B)}{1/2-\varepsilon}$. We claim that up to a set of measure zero, C is contained in $C_\varepsilon \cup (B \backslash \bar{D})$. We omit the proof of this simple fact. We conclude that $\mu(C) \leq \frac{\mu(B)}{1/2-\varepsilon}$. □

9 Future Work

1. *Lower bounds:* The best known lower bound for the one-dimensional uniform distribution property tester [3] is $\Omega(\varepsilon^{-1}\log n)$. For arbitrary distribution it is possible, using Yao's minimax principal, to show a lower bound of $\Omega(\varepsilon^{-1}\log(\varepsilon/p_{max}))$, where p_{max} is the maximal probability in the distribution. Note that $\log(1/p_{max})$ can be arbitrarily smaller than H. It would be interesting to close the gap, as well as generalize for higher dimension.
2. *High-dimensional monotonicity:* It is not known if Lemma 4 is tight. Namely, is there a high dimensional function that has axis-parallel distances from monotonicity exponentially (in d) smaller than the global distance to monotonicity? We note that even if the exponential dependence is tight in the inequality, it would not necessarily mean that the property testing query complexity should be exponential in d (other algorithms that are not based on axis-parallel comparisons might do a better job).
3. *Other posets and distributions:* It would be interesting to generalize the results here to functions over general posets [6] as well as arbitrary distributions (not necessarily product distributions).
4. *More information theory in property testing:* It would be interesting to see how the entropy or other complexity measures of $\mathcal{D}$ affect the query complexity of other interesting property testing problems.

Acknowledgements. We would like to thank Shirley Halevy and Eyal Kushilevitz for enlightening discussions.

References

1. Batu, T., Rubinfeld, R., White, P. *Fast approximate PCPs for multidimensional bin-packing problems*, Proc. RANDOM (1999), 245–256.
2. Dodis, Y., Goldreich, O., Lehman, E., Raskhodnikova, S., Ron, D., Samorodnitsky, A. *Improved testing algorithms for monotonicity*, Proc. RANDOM (1999), 97–108.
3. Ergun, F., Kannan, S., Kumar, S. Ravi, Rubinfeld, R., Viswanathan, M. *Spot-checkers*, Proc. STOC (1998), 259–268.
4. Fischer, E. *The art of uninformed decisions: A primer to property testing*, Bulletin of EATCS, 75: 97-126, 2001.
5. Fischer, E., Kindler, G., Ron, D., Safra, S., Samorodnitsky, A., *Testing Juntas* Proc. FOCS (2002), 103–112.
6. Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., Samorodnitsky, A. *Monotonicity testing over general poset domains*, Proc. STOC (2002), 474–483.
7. Goldreich, O. *Combinatorial property testing - A survey*, in "Randomization Methods in Algorithm Design," 45-60, 1998.
8. Goldreich, O., Goldwasser, S., Lehman, E., Ron, D., Samordinsky, A. *Testing monotonicity*, Combinatorica, 20 (2000), 301–337.
9. Goldreich, O., Goldwasser, S., Ron, D. *Property testing and its connection to learning and approximation*, J. ACM 45 (1998), 653–750.

10. Halevy, S., Kushilevitz, E. *Distribution-free property testing*, Proc. RANDOM (2003), 302–317.
11. Halevy, S., Kushilevitz, E. *Testing Monotonicity over Graph Products*, ICALP (2004).
12. Parnas, M., Ron, D., Rubinfeld, R. *Tolerant property testing and distance approximation*, ECCC 2004.
13. Ron, D. *Property testing*, in "Handbook on Randomization," Volume II, 597-649, 2001.
14. Rubinfeld, R., Sudan, M. *Robust characterization of polynomials with applications to program testing*, SIAM J. Comput. 25 (1996), 647–668.

On Nash Equilibria in Non-cooperative All-Optical Networks

Vittorio Bilò[1,2], Michele Flammini[1], and Luca Moscardelli[1]

[1] Dipartimento di Informatica, Università di L'Aquila,
Via Vetoio, Coppito 67100 L'Aquila, Italy
{bilo, flammini, moscardelli}@di.univaq.it
[2] Dipartimento di Matematica "Ennio De Giorgi", Università di Lecce,
Provinciale Lecce-Arnesano, P.O. Box 193, I-73100 Lecce, Italy

Abstract. In this paper we investigate the problem in which an all-optical network provider must determine suitable payment functions for non-cooperative agents wishing to communicate so as to induce routings in Nash equilibrium using a low number of wavelengths. We assume three different information levels specifying the local knowledge that agents may exploit to compute their payments. While under complete information of all the agents and their routing strategies we show that functions can be determined that perform how centralized algorithms preserving their time complexity, knowing only the used wavelengths along connecting paths (minimal level) or along the edges (intermediate level) the most reasonable functions either do not admit equilibria or equilibria with a different color assigned to each agent, that is with the worst possible ratio between the Nash versus optimum performance, also called price of anarchy. However, by suitably restricting the network topology, a price of anarchy 25.72 has been obtained for chains and 51.44 for rings under the minimal level, and further reduced respectively to 3 and 6 under the intermediate level, up to additive factors converging to 0 as the load increases. Finally, again under the minimal level, a price of anarchy logarithmic in the number of agents has been determined also for trees.

1 Introduction

All-optical networks are widely considered to be the future of the state of the art communication networks due to the possibility of managing thousand of users, covering wide areas and providing a bandwidth which is orders of magnitude faster than traditional networks. Such high performances elect optical as the leading technology in many applications such as video conferencing, scientific visualization and high-speed distributed computing. The high bandwidth provided by all-optical networks can be partitioned by means of the *wavelength-division multiplexing* (WDM) [4] in order to obtain a large number of parallel high speed channels along a same optical fiber (see [2, 8] for a survey of the main related results).

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 448–459, 2005.

The problem of investigating the existence and performance of Nash equilibria in all-optical networks has been first considered in [3]. In such a setting, a service provider has to satisfy a given set of point-to-point communication requests, charging each of them a cost depending on its wavelength and on the wavelengths of the other requests met along its path in the network. Each request is issued by a non-cooperative agent who is interested only in the minimization of his own cost. Under this assumption any request is willing to be rerouted each time it may be served by a cheaper path in the network and the evolution of the network can be modelled as a multi-player game. A routing solution, that is an assignment of paths and colors to the requests, in which no request can lower its cost by choosing a different strategy is said to be a pure Nash equilibrium. In [3] four different reasonable payment functions for the agents have been presented and shown either to induce equilibria with the worst possible global performance, that is with a different color assigned to each request, or not yielding the convergence of the agents to an equilibrium. However, the existence of Nash equilibria for such non-convergent functions has been left open.

Several other games [5, 6, 7, 12, 13, 14, 15, 18] have been shown to possess pure Nash equilibria or to converge to a pure Nash equilibrium independently from their starting state. In [10] the loss of the global performance of Nash equilibria due to the lack of cooperation among the agents has been expressed as the ratio between the cost of the worst Nash equilibrium and that of an optimal centralized solution, also called *price of anarchy* or *coordination ratio*. Bounding the price of anarchy of selfish routing in different models is now arising as one of the most interesting research areas lying on the boundary between Computer Science and Game Theory, see for example [11, 16].

In this paper we are interested in the problem of determining suitable payment functions for the non-cooperative agents of an all-optical network that induce Nash equilibria using a low number of wavelengths. We assume different information levels specifying the local knowledge that agents may exploit to compute their payments. In particular, each agent has knowledge of all the other agents and their routing strategies under the complete information level, of the wavelengths used along any given edge under the intermediate level, and finally of the wavelengths used along any given path connecting its source and destination under the minimal level. We first show that under the complete level, functions can be determined that perform how centralized algorithms for the problem, that is preserving both their approximation ratio and their time complexity. We then prove that the most reasonable functions under the remaining two levels either do not admit equilibria or have the worst possible price of anarchy, that is with a different color assigned to each agent. However, by suitably restricting the network topology, a price of anarchy 25.72 has been obtained for chains and 51.44 for rings under the minimal level, and further reduced respectively to 3 and 6 under the intermediate level, up to additive factor converging to 0 as the load increases. Finally, again under the minimal level, a price of anarchy logarithmic in the number of agents has been determined for trees.

The paper is organized as follows. In the next section we give the basic definitions and notation. In Section 3 we present the above mentioned results concerning the complete information level and we introduce suitable classes of payment functions useful for characterizing convergent games with the worst possible price of anarchy. In Section 4 we present the general results concerning the reasonable payment functions for the minimal and intermediate levels. In Section 5, we present the results for specific networks, that is chains, rings and trees, and finally, in Section 6, we give some conclusive remarks and discuss some open questions.

2 The Model

We model an all-optical network as an undirected graph $G = (V, E)$ where nodes in V represent sites and undirected edges in E bidirectional optical fiber links between the sites.

Given any two nodes $x, y \in V$, we denote a communication request between x and y as $\{x, y\}$. A communication instance in G is a multiset of requests I eventually containing multiple requests between the same pairs of nodes. A path system P for an instance I in G is a set of paths containing a distinguished simple connecting path in G for each request in I. A solution $\mathcal{R}(G, I)$ for an instance I in G, $\mathcal{R}$ for short, is a pair $(P_{\mathcal{R}}, c_{\mathcal{R}})$ in which $P_{\mathcal{R}}$ is a path system for I and $c_{\mathcal{R}} : I \rightarrow W$ (with $W = \mathbb{N}^+$ being the set of wavelengths) is a function associating a wavelength or color to each request in I. Let $p_{\mathcal{R}}(\{x, y\}) \in P_{\mathcal{R}}$ denote the path connecting $\{x, y\}$ in $\mathcal{R}$ and $|p_{\mathcal{R}}(\{x, y\})|$ be its length in terms of number of edges. A solution $\mathcal{R}$ is feasible or is a routing for I in G if $c_{\mathcal{R}}(\{x_1, y_1\}) \neq c_{\mathcal{R}}(\{x_2, y_2\})$ for any two requests $\{x_1, y_1\} \in I$ and $\{x_2, y_2\} \in I$ whose connecting paths $p_{\mathcal{R}}(\{x_1, y_1\})$ and $p_{\mathcal{R}}(\{x_2, y_2\})$ share an edge in G, i.e., the associated colors are different.

Let $\omega_{\mathcal{R}}(G, I)$ be the number of colors used by the routing $\mathcal{R}$ for I in G and $\omega(G, I) = \min_{\mathcal{R}} \omega_{\mathcal{R}}(G, I)$ be the minimum number of colors used by any routing for I. Analogously, let $\pi_{\mathcal{R}}(G, I) = max_{e \in E} |\{p \in P_{\mathcal{R}} | e \in p\}|$ be the maximum load of an edge in G induced by the routing $\mathcal{R}$ and $\pi(G, I) = \min_{\mathcal{R}} \pi_{\mathcal{R}}(G, I)$ be the minimum maximum load of the routings for I in G, also called load of I in G. Clearly, since all the requests sharing an edge must have different colors, $\omega_{\mathcal{R}}(G, I) \geq \pi_{\mathcal{R}}(G, I)$ for every $\mathcal{R}$ and thus $\omega(G, I) \geq \pi(G, I)$.

Since the optical spectrum is a scarce resource, an interesting optimization all-optical routing problem arises consisting in the determination of routings using a number of wavelengths close to the minimum one, i.e., $\omega(G, I)$.

In order to model our non-cooperative environment, we assume that each communication request $\{x, y\} \in I$ is issued and handled by an agent α that for the sake of simplicity we consider as coincident with the request, that is $\alpha = \{x, y\}$. A payment function $price_{\mathcal{R}} : I \rightarrow \mathbb{R}^+$ is a function associating to each agent $\alpha \in I$ the price he has to pay to the network provider in order to obtain the asked service if the routing $\mathcal{R}$ is adopted. Let *price* denote the collection of the functions $price_{\mathcal{R}}$ for all the possible routings $\mathcal{R}$. With abuse

of notation, in the following we often identify the functions $price_{\mathcal{R}}$ with the collection $price$.

In order to represent the increasing cost incurred by the network provider to implement a routing using up to a given wavelength and to give to our payment functions a higher degree of generality, we assume the existence of a non-decreasing function $f : W \to I\!R^+$ associating a (positive) cost to every color.

A routing $\mathcal{R}$ is at Nash equilibrium if and only if for any agent α and routing $\mathcal{R}'$ differing from $\mathcal{R}$ only for the path and/or the color associated to α, it holds $price_{\mathcal{R}}(\alpha) \leq price_{\mathcal{R}'}(\alpha)$.

A game $\mathcal{G} = (G, I, price)$ among the $|I|$ agents belonging to I on the network G induced by the collection of pricing functions $price$ is defined as the set of agents' strategies $P^\alpha \times W$, where P^α is the set of connecting paths for α and the utility function $u(\alpha) = -p(\alpha)$. Denoted as $\mathcal{N}$ the set of the routings at Nash equilibrium, the coordination ratio or price of anarchy of the game $\mathcal{G}$ is defined as $\rho(G) = sup_{\mathcal{R}\in\mathcal{N}} \frac{\omega_{\mathcal{R}}(G,I)}{\omega(G,I)}$.

The evolution of the game $\mathcal{G}$ is a sequence of moves $(\alpha, p_{old}, color_{old}, p_{new}, color_{new})$, where α is an agent and $(p_{old}, color_{old}) \in P^\alpha \times W$ and $(p_{new}, color_{new}) \in P^\alpha \times W$ are the old and the new strategy of α, respectively. The Nash dynamics of the game $\mathcal{G}$ is the directed graph (Φ, M) where Φ is the set of the possible routings for G and I and there exists an arc $(\mathcal{R}_1, \mathcal{R}_2) \in M$ if there exists a selfish move from $\mathcal{R}_1$ to $\mathcal{R}_2$. If the Nash dynamics is acyclic for every G and I, the game $\mathcal{G}$ is said to be convergent. Analogously, any payment function $price$ inducing only convergent games is said to be convergent.

Proposition 1. *Any convergent game $\mathcal{G}$ admits at least one Nash equilibrium.*

It is worth noting that even if the Nash dynamics is not acyclic the game might admit a Nash equilibrium.

Before concluding the section, we finally observe that the payment function $price_{\mathcal{R}}$, in a strongly distributed non-cooperative environment, must be computed by each single agent requiring the communication service. However, the level of global information they have can be limited by technological constraints as well as privacy policies carried out by the service provider or simply enforced by the law. Therefore, in general $price_{\mathcal{R}}$ is not computed starting from the instance I and the routing $\mathcal{R}$, but on a more restricted set of information induced by them. In order to better specify this aspect, we introduce the concepts of states and levels of information.

The edge state (resp. path state) of the network G induced by a routing $\mathcal{R}$ for I is a function $\sigma_{\mathcal{R}} : E \to 2^W$ (resp. $\bar{\sigma}_{\mathcal{R}} : \mathcal{P} \to 2^W$ where $\mathcal{P}$ is the set of all the simple paths in G) associating to every edge $e \in E$ (resp. path $p \in \mathcal{P}$) the set of the wavelengths used along e (resp. p).

It is then possible to distinguish among three basic levels of information:

Minimal. Each agent $\alpha = \{x, y\}$ knows the available wavelengths along any given path connecting x to y, that is the function $price_{\mathcal{R}}$ can be computed

even knowing only the restriction of the path state $\bar{\sigma}_{\mathcal{R}}(\alpha)$ on the set of the paths from x to y.

Intermediate. Each agent information knows the wavelengths available along any given edge, that is the function $price_{\mathcal{R}}$ can be computed even knowing only the edge state $\sigma_{\mathcal{R}}$.

Complete. Each agent $\alpha = \{x, y\}$ knows the instance I and the routing $\mathcal{R}$, that is the function *price* is not restricted.

Clearly, even if not explicitly mentioned, in any level of information, for each agent α, $price_{\mathcal{R}}(\alpha)$ can depend also on the wavelength $c(\alpha)$ and the path $p(\alpha)$ assigned to α in $\mathcal{R}$. Notice also that even with a minimal information any agent is always able to compute a valid wavelength for the chosen path so as to not interfere with any other agent and thus not compromising the feasibility of the solution.

For ease of notation, when clear from the context, in the following we will drop the index $\mathcal{R}$ from the notation, thus for instance writing $price(\alpha)$, $p(\alpha)$, $c(\alpha)$, $\sigma(e)$ and $\bar{\sigma}(p)$ in place of $price_{\mathcal{R}}(\alpha)$, $p_{\mathcal{R}}(\alpha)$, $c_{\mathcal{R}}(\alpha)$, $\sigma_{\mathcal{R}}(e)$ and $\bar{\sigma}_{\mathcal{R}}(p)$.

3 Preliminaries

In this section we first prove that, under complete information, any centralized algorithm $\mathcal{A}$ for the all-optical routing problem can be suitably used in a non-cooperative environment for achieving a price of anarchy either equal to 1 or k according to whether $\mathcal{A}$ is optimal or k-approximating, respectively.

Theorem 1. *Let $\mathcal{A}$ be any k-approximation algorithm for the all-optical routing problem ($k = 1$ if $\mathcal{A}$ is optimal). Then there exists a payment function yielding a game $\mathcal{G}$ converging in at most $3|I|$ steps and having price of anarchy equal to k. Moreover, the time complexity of the computation of the function is at most the same of $\mathcal{A}$.*

Starting from the above theorem, in the sequel we will only focus on the minimal and intermediate information levels. Before proceeding with the presentation of our results, let us first introduce two families of payment functions that provide a useful characterization of the class of convergent games with the worst possible price of anarchy.

Given a generic move μ performed by an agent α from $\mathcal{R}_{old}$ to $\mathcal{R}_{new}$, let A be the set of agents sharing at least one edge with α in $\mathcal{R}_{old}$ and no edges with α in $\mathcal{R}_{new}$ and B be the set of agents sharing at least one edge with α in $\mathcal{R}_{new}$. We denote with Π the set of payment functions satisfying the following conditions:

$$\forall \beta \in A, price_{\mathcal{R}_{new}}(\beta) \leq price_{\mathcal{R}_{old}}(\beta) \tag{1}$$

$$\forall \beta \in B, price_{\mathcal{R}_{new}}(\beta) > price_{\mathcal{R}_{old}}(\beta) \Rightarrow price_{\mathcal{R}_{new}}(\beta) \leq price_{\mathcal{R}_{new}}(\alpha) \tag{2}$$

Theorem 2. *All the payment functions belonging to Π are convergent.*

As it can be easily seen, the number of steps needed to converge to an equilibrium may not be polynomial in the dimensions of the instance.

We now define a subclass of Π of payment functions inducing games $\mathcal{G}$ having a price of anarchy $\rho = \frac{|I|}{\omega(G,I)}$, which is clearly the worst possible since any feasible solution for the problem uses at most $|I|$ colors while $\omega(G, I)$ is the optimal solution.

We denote as $\Xi = \Xi' \cup \Xi''$ the class of payment functions satisfying at least one of the following conditions:

Subclass Ξ': $\forall \alpha \in I, price(\alpha)$ depends only on the color $c(\alpha)$ and $c(\alpha) \geq c'(\alpha) \Rightarrow price_{\mathcal{R}}(\alpha) \geq price_{\mathcal{R}'}(\alpha)$ for any $\mathcal{R}$ and $\mathcal{R}'$, that is the cost for each agent depends only on his own color in the routing and any other agent α never gets a benefit in performing a move $(\alpha, p_{old}, color_{old}, p_{new}, color_{new})$ where $color_{new} \geq color_{old}$.

Subclass Ξ'': $\forall \alpha \in I$, $price(\alpha) = max_{e \in p(\alpha)} g(\sigma(e))$ where $g : 2^W \rightarrow I\!R$ is such that for any sets of colors A and B, $A \setminus \{d\} \subseteq B \Rightarrow g(A) \leq g(B \cup \{d'\})$ with $d \in A$, $d' \geq d$ and $d' \notin B$. In other words, the image $g(B \cup \{d'\})$ according to g of a set $B \cup \{d'\}$ containing all the elements of another set A except for at most one element (d) which has to be replaced in the first set by a strictly greater element d', cannot be smaller than the image $g(A)$ of the second set.

Theorem 3. *$\Xi \subseteq \Pi$, that is any function price $\in \Xi$ is convergent. Moreover, all the payment functions belonging to Ξ induce games $\mathcal{G} = (G, I, price)$ having a price of anarchy $\rho = \frac{|I|}{w(G,I)}$.*

4 Minimal and Intermediate Payment Functions

Even though the results obtained in the case of complete information are fully satisfactory, it must be stressed that the assumption that each agent has the full knowledge of the instance and the routing is very strong and in real world networks might not be true either due to technological or privacy policies reasons. Therefore, in the sequel we will focus on payment functions based on a more restricted level of information.

Let us first define a complete set of payment functions that can be computed under a minimal or intermediate information level.

Given a routing $\mathcal{R}$, we first propose suitable cost functions defined on the edges that will be used as building blocks for the definition of the mentioned payment functions:

- $col(e, \alpha) = f(c(\alpha))$: the amount charged to α on the edge e is the cost, according to f, of the color he uses.
- $max(e, \alpha) = \max_{k \in \sigma_{\mathcal{R}}(e)} f(k)$: the amount charged to α on the edge e is the cost of the highest color used along e (considering also the other agents).
- $sum(e, \alpha) = \sum_{k \in \sigma_{\mathcal{R}}(e)} f(k)$: the amount charged to α on the edge e is the sum of the costs of all the colors used along e.

- $avmax(e,\alpha) = \max_{k\in\sigma_{\mathcal{R}}(e)} \frac{f(k)}{|\sigma_{\mathcal{R}}(e)|}$: the amount charged to α on the edge e is the cost of the highest color used along e, averaged or shared among all the agents traversing e.
- $avsum(e,\alpha) = \sum_{k\in\sigma_{\mathcal{R}}(e)} \frac{f(k)}{|\sigma_{\mathcal{R}}(e)|}$: the amount charged to α on the edge e is the sum of the costs of all the colors used along e, averaged on all the agents traversing e.

Starting from any edge cost function *cost*, it is possible to define the following payment functions:

- $max - cost(\alpha) = \max_{e\in p(\alpha)} cost(e,\alpha)$: the price asked to α is the maximum cost, according to *cost*, of an edge used by α.
- $sum - cost(\alpha) = \sum_{e\in p(\alpha)} cost(e,\alpha)$: the price asked to α is the sum of the costs of the edges used by α.

The combination of the introduced edge cost functions with the above two strategies, that is maximization or summation, gives rise to ten possible payment functions. In all the cases, since the function f is non decreasing, agents have an incentive to choose small colors so as to possibly minimize the overall number of used colors.

The functions $max - col$, $max - max$, $sum - max$ and $sum - avmax$ have been considered in [3], where $max - col$ and $max - max$ have been shown to be convergent even if inducing games $\mathcal{G} = (G, I, price)$ with the worst possible price of anarchy $\rho = \frac{|I|}{w(G,I)}$, while $sum - max$ and $sum - avmax$ have been shown to be non convergent, even if the existence of Nash equilibria in the corresponding games was left open.

Notice that all the ten introduced payment functions are computable under an intermediate information level. Moreover, $max - col$, $sum - col$ and $max - max$ require only the minimal level, as $max - col(\alpha) = \max_{e\in p(\alpha)} col(e,\alpha) = f(c(\alpha))$, $sum - col(\alpha) = \sum_{e\in p(\alpha)} col(e,\alpha) = |p(\alpha)| \cdot f(c(\alpha))$ and $max - max(\alpha) = \max_{e\in p(\alpha)} \max_{k\in\sigma(e)} f(k) = \max_{k\in\bar{\sigma}(p(\alpha))} f(k)$.

In particular, the following lemma, leading to the same consequences of [3], shows how the families of functions defined in the previous section nicely characterize their class of games.

Lemma 1. *The payment functions $max - col$ and $max - max$ belong to the class Ξ.*

Moreover, a similar result holds for $sum - col$.

Theorem 4. *The payment function $sum - col$ belongs to Π, that is it is convergent, but induces games $\mathcal{G} = (G, I, price)$ having a price of anarchy $\rho = \frac{|I|}{w(G,I)}$.*

Therefore, even if convergent, all the three minimal level functions yield the worst possible price of anarchy. Unfortunately, the following results show that also the remaining seven payment functions for the intermediate information level either are not convergent or yield the worst possible price of anarchy.

Theorem 5. *The payment function $max - sum$ belongs to the class Ξ.*

Proof. We show that $max - sum$ belongs to the subclass Ξ'' of Ξ. To this aim, it suffices to show that $g(\sigma(e)) = sum(e, \alpha) = \sum_{k \in \sigma(e)} f(k)$ satisfies, for any sets of colors A and B, the property $A \setminus \{d\} \subseteq B \Rightarrow g(A) \leq g(B \cup \{d'\})$ with $d \in A$, $d' \geq d$ and $d' \notin B$. Since $A \setminus \{d\} \subseteq B$ and f is positive and non decreasing, we have $g(B \cup \{d'\}) = f(d') + \sum_{k \in B} f(k) \geq f(d) + \sum_{k \in A \setminus \{d\}} f(k) = g(A)$. □

Theorem 6. *No Nash equilibria exist for the games induced by the payment functions*

1. *$sum - max$ when the pricing function f is unbounded;*
2. *$max - avmax$ and $sum - avmax$ when f is such that $\exists k : f(k) > 2f(1)$;*
3. *$max - avsum$ and $sum - avsum$ when the f is such that $\exists k : f(k) > f(1)$, that is f is non constant.*

Finally, we have the following theorem concerning $sum - sum$.

Theorem 7. *The payment function $sum - sum$ is not convergent when the pricing function f is such that $\exists k : f(k) > f(1) + f(2)$.*

According to the above results, an interesting left open question for the minimal and intermediate information levels is the determination of suitable payment functions able to induce convergent games with a price of anarchy better than the worst possible one.

5 Results for Specific Topologies

Since the results obtained for generic networks are not fully satisfactory when the level of information is not complete, in this section we focus on networks having specific topologies, like chains (nodes connected along a line), rings (cycles of nodes) and trees.

Let us first consider the minimal information level.

Any strictly increasing payment function computable under this level, like $price(\alpha) = c(\alpha)$, induces games in which a routing $\mathcal{R}$ at Nash equilibrium can be seen as a solution of the classical *First-Fit* algorithm for the all-optical routing problem that assigns to each request the smallest available color. In particular, such a solution is the one returned by *First-Fit* when requests are considered in non decreasing order of color in $\mathcal{R}$. Therefore, the induced price of anarchy is bounded by the approximation ratio of *First-Fit*.

Concerning the above mentioned topologies, for any instance I, *First-Fit* uses a number of colors that is at most $25.72\pi(G, I)$ in chains [9] and at most $\mathcal{O}((\log |I|)\pi(G, I))$ in trees [1].

Recalling that $\omega(G, I) \geq \pi(G, I)$, the following theorem holds.

Theorem 8. *The payment function $price(\alpha) = c(\alpha)$ induces games with a price of anarchy 25.72 in chains and $\rho = \mathcal{O}(\log |I|)$ in trees, both converging in $\omega_{\mathcal{R}}(G, I)^2$ steps from any initial routing $\mathcal{R}$.*

For rings, it is possible to use the result of [9] by routing requests on the chain obtained deleting an edge $e \in E$ of the ring. In fact, denoted as P the path system containing all such connecting paths, for any routing $\mathcal{R}$ with $P_{\mathcal{R}} = P$ it results $\pi_{\mathcal{R}}(G, I) \leq 2\pi(G, I)$. In other words, this at most doubles the induced load.

The following payment function forces Nash equilibria using the set of paths P:

$$price(\alpha) = \begin{cases} 1 & \text{if } e \in p(\alpha) \\ 1 - \frac{1}{c(\alpha)} & \text{otherwise} \end{cases}$$

Since the payment function belongs to Π and restricted to the routings $\mathcal{R}$ with $P_{\mathcal{R}} = P$ is strictly increasing, that is each agent has an incentive in choosing the smallest available color, by the doubling of the load the following theorem holds.

Theorem 9. *The above payment function in rings induces games with price of anarchy* 51.44 *converging in* $\omega_{\mathcal{R}}(G, I)^2$ *steps from any initial routing* $\mathcal{R}$.

In the remaining part of the section we show how, raising the level of information to the intermediate one, it is possible to further reduce the price of anarchy for chains and rings. We first focus on ring topologies, as the corresponding results can be directly extended to chains.

Our purpose is to force the agents to simulate the behavior of the online algorithm proposed by Slusarek [17]. In such an algorithm, the path system P is fixed and the optical spectrum is divided in shelves, numbered consecutively starting from 1. The color assigned to an arriving agent α is the lowest available one in the minimal shelf i such that the load induced on the edges of α by all the agents up to shelf i is at most i. More precisely, denoted as $sh(w)$ the shelf of a given wavelength w, the load $l(\alpha, \mathcal{R})$ of an agent α according to a routing $\mathcal{R}$ is

$$l(\alpha, \mathcal{R}) = \max_{e \in p(\alpha)} |\{w | w \in \sigma(e) \wedge sh(w) \leq sh(c(\alpha))\}|.$$

Clearly, any routing $\mathcal{R}$ such that $P_{\mathcal{R}} = P$ has the same load $\pi_{\mathcal{R}}(G, I)$ and in the routing $\mathcal{R}$ returned by the algorithm at most $\pi_{\mathcal{R}}(G, I)$ shelves are used to allocate all the agents. Moreover, as shown in [17], during all the execution of the algorithm the first shelf contains only one color and the other ones no more than three colors, thus yielding $\omega_{\mathcal{R}}(G, I) \leq 3 \cdot \pi_{\mathcal{R}}(G, I) - 2$ colors, that is at most 3 times above the optimum.

In devising a payment function that mimic Slusarek's algorithm it is necessary to cope with several problems. First of all, it is necessary to fix a path system with a low induced load. This is obtained by incentiving agents to use shortest paths, as in any routing $\mathcal{R}$ satisfying such a property $\pi_{\mathcal{R}}(G, I) \leq 2\pi(G, I)$, that is the induced load is at most doubled. Moreover, the convergence of the game is an issue since the move of an agent might compromise the minimality of the payments of the agents in his shelf and in the above ones. Finally, in order for an agent to always find a proper shelf during the evolution of the game, we have to allow an unlimited number of colors to be contained in each shelf. Since however Nash equilibria will use only one wavelength of the first shelf and at most three

in the other ones, we have to choose the function sh associating colors to shelves trying to minimize, for each i, the maximum third color of all the shelves up to i. To this aim, we partition the set of wavelengths W in two subsets W_1 and W_2 in which $W_2 = \{2^i | i = 0, 1, 2, \ldots\}$ is the set of the slack wavelengths. Each shelf has an infinite number of wavelengths and the first one of shelf 1 and the first three ones of all the other shelves correspond consecutively to small colors in W_1, while the other ones to slack colors in W_2, assigned to the different shelves according to a dove-tail technique in such a way that, for every $i \geq 1$ and $j > 3$, there exist a finite wavelength $w \in W_2$ that corresponds to the j-th color of shelf i.

As it can be easily checked, while the first color of shelf 1 is 3, the third color of each shelf $i > 1$ is at most $3i + \lfloor \log_2 3i \rfloor + 3$.

Assuming that the function sh realizes the above mapping of colors and that n is the number of nodes in the ring, the payment function $price_{\mathcal{R}}$ charges to agent α

$$\min \left\{ \left\lfloor \frac{n}{2} \right\rfloor, |p(\alpha)| + n \max\{0, l(\alpha, \mathcal{R}) - sh(c(\alpha))\} - \frac{1}{2 + sh(c(\alpha)) - \frac{1}{2c(\alpha)}} \right\}.$$

Intuitively, a routing $\mathcal{R}$ is at Nash equilibrium if and only if each request uses a shortest path and the color that Slusarek's algorithm assigns to him when agents arrive in non decreasing order of color in $\mathcal{R}$, clearly not using colors in W_2.

Theorem 10. *The game induced by price in rings converges and has a price of anarchy* $\rho = 6 + \mathcal{O}(\frac{log(\pi(G,I))}{\pi(G,I)})$.

Proof. Let us consider any evolution of the game starting from any given configuration and let us show that it converges to a Nash equilibrium in a finite number of agent moves. At any intermediate configuration, let us partition I into two sets A and B such that A contains all the agents that in the sequel of the game will never perform a move to a higher shelf. We show that, after a finite number of moves, a new agent will enter in A, so that in a finite number of steps it will finally result $A = I$. This proves the convergence of the game, since starting from such final configuration the agents can not perform an infinite number of moves without raising their shelf.

First of all we note that, since by construction the number of wavelengths in each shelf is infinite, each agent α can always perform a move leading him in a feasible shelf i, that is such that $i \geq l(\alpha, \mathcal{R})$. Moreover, the moves leading α on a shortest path and in a feasible shelf are the only ones which can make α's cost strictly smaller than $\lfloor \frac{n}{2} \rfloor$. Finally, an agent routing a shortest path and lying in a feasible shelf, maintaining his connection path decreases his cost if he moves to a lower feasible shelf or to a smaller color of its shelf.

Consider a move of agent α_1 to a shelf i_1. The proof proceeds by cases: after the move

1. α_1 will never perform a move increasing his shelf.
   In this case α_1 is entered in A.
2. At a certain point of the game, α_1 performs a move increasing his shelf.
   This is due to another agent α_2 who has performed a move to a shelf $i_2 \leq i_1$. However, it cannot be $i_2 = i_1$, as α_2 increases α_1's load to $i_1 + 1$ and would have the same load $i_1 + 1$ in shelf i_1 (i.e., i_1 is not feasible for α_2), thus not decreasing his cost. We can then continue applying the same analysis to α_2. Since the number of shelves in which agents can move is bounded by $\pi(G, I)$, we must finally arrive to an agent α_j with $j \leq \pi(G, I)$ for which the first case holds.

As already observed, any routing $\mathcal{R}$ at Nash equilibrium corresponds to the output of Slusarek's algorithm without using colors in W_2 when the agents arrive in non decreasing order of color in $\mathcal{R}$. Thus, the maximum used color in $\mathcal{R}$ is at most the third one of shelf $\pi(G, I)$ if $\pi(G, I) > 1$ while it is equal to the first one of shelf 1 otherwise. This shows that the maximum used color is at most $3\pi_{\mathcal{R}}(G, I) + \lfloor \log_2 3\pi_{\mathcal{R}}(G, I) \rfloor + 3$. Since $\mathcal{R}$ uses shortest paths, $\pi_{\mathcal{R}}(G, I) \leq 2\pi_{\mathcal{R}}(G, I)$ and the resulting price of anarchy is $\rho = 6 + \mathcal{O}(\frac{log(\pi_{\mathcal{R}}(G,I))}{\pi_{\mathcal{R}}(G,I)})$. □

Clearly, a similar function *price*′ defined as

$$\min \left\{ n, 1 + 2n \max\{0, load(\alpha, \mathcal{R}) - sh(c(\alpha))\} - \frac{1}{2 + sh(c(\alpha)) - \frac{1}{2c(\alpha)}} \right\}$$

can be used also in chains, for which connection paths are unique and thus no doubling of the load occurs.

Theorem 11. *The game induced by price*′ *in chains converges and has a price of anarchy* $\rho = 3 + \mathcal{O}(\frac{log(\pi(G,I))}{\pi(G,I)})$.

6 Conclusion

We have considered the problem of determining suitable payment functions for the non-cooperative agents of an all-optical network to induce Nash equilibria using a low number of wavelengths.

While the complete information level has been fully understood, under the lower levels the main left open question is the determination of functions that on every topology yield Nash equilibria with a performance better than the worst possible one assigning a different color to each agent. Moreover, still under incomplete information, it would be also nice to improve and extend our results on specific topologies.

Finally, our results outline nice connections between payment functions and online algorithms, that allow to cope with the arbitrary order of the moves of the agents. It would be nice to understand the conditions and eventual systematic methods allowing to get converging payment functions preserving online algorithms performances under uncomplete information.

References

1. Y. Bartal and S. Leonardi. On-line routing in all-optical networks. *Theoretical Computer Science - special issue for ICALP 1997*, 221(1-2):19–39, 1999.
2. B. Beauquier, J. C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro. Graph problems arising from wavelength-routing in all- optical networks. In *Proceedings of the 2nd Workshop on Optics and Computer Science, part of IPPS'97*, 1997.
3. V. Bilò and L. Moscardelli. The price of anarchy in all-optical networks. In *Proceedings of the 11th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 3104 of *LNCS*, pages 13–22. Springer, 2004.
4. N. K. Chung, K. Nosu, and G. Winzer. Special issue on dense WDM networks. *IEEE Journal on Selected Areas in Communications*, 8, 1990.
5. A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–351, 2003.
6. A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure equilibria. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, pages 604–612. ACM Press, 2004.
7. D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP)*, LNCS, pages 123–134. Springer, 2002.
8. L. Gargano and U. Vaccaro. *"Routing in All–Optical Networks: Algorithmic and Graph–Theoretic Problems"in: Numbers, Information and Complexity.* Kluwer Academic, 2000.
9. H.A. Kierstead and J. Qin. Coloring interval graphs with first-fit. *Discrete Mathematics*, 144:47–57, 1995.
10. E. Koutsoupias and C.H. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *LNCS*, pages 387–396, 1999.
11. M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC)*, pages 510–519, 2001.
12. I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13:111–124, 1996.
13. G. Owen. *Game theory.* Academic Press, 3rd edition, 1995.
14. C. H. Papadimitriou and M. Yannakakis. On complexity as bounded rationality. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 726–733, 1994.
15. R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.
16. T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of ACM*, 49(2):236–259, 2002.
17. M. Slusarek. Optimal on-line coloring of circular arc graphs. *Informatique Théorique et Applications*, 29(5):423–429, 1995.
18. A. Vetta. Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 416–425, 2002.

Speed Scaling to Manage Temperature

Nikhil Bansal[1] and Kirk Pruhs[2,⋆]

[1] IBM T. J. Watson Research Center
nikhil@us.ibm.com
[2] Computer Science Department, University of Pittsburgh
kirk@cs.pitt.edu

Abstract. We consider speed scaling algorithms to minimize device temperature subject to the constraint that every task finishes by its deadline. We assume that the device cools according to Fourier's law. We show that the optimal offline algorithm proposed in [18] for minimizing total energy (that we call YDS) is an $O(1)$-approximation with respect to temperature. Tangentially, we observe that the energy optimality of YDS is an elegant consequence of the well known KKT optimality conditions. Two online algorithms, AVR and Optimal Available, were proposed in [18] in the context of energy management. It was shown that these algorithms were $O(1)$-competitive with respect to energy in [18] and [2]. Here we show these algorithms are not $O(1)$-competitive with respect to temperature. This demonstratively illustrates the observation from practice that power management techniques that are effective for managing energy may not be effective for managing temperature. We show that the most intuitive temperature management algorithm, running at such a speed so that the temperature is constant, is surprisingly not $O(1)$-competitive with respect to temperature. In contrast, we show that the online algorithm BKP, proposed in [2], is $O(1)$-competitive with respect to temperature. This is the first $O(1)$-competitiveness analysis with respect to temperature for an online algorithm.

1 Introduction

In May Intel abruptly announced that it had scrapped the development of two new computer chips (code named Tejas and Jayhawk) for desktops/servers in order to rush to the marketplace a more efficient chip technology more than a year ahead of schedule. Analysts said the move showed how eager the world's largest chip maker was to cut back on the heat its chips generate. Intel's method of cranking up chip speed was beginning to require expensive and noisy cooling systems for computers [17]. (Current estimates are that cooling solutions are rising at $1 to $3 per watt of heat dissipated [15].) This demonstrates that exponentially growing device power consumption, the power densities in microprocessors have doubled every three years [15], has reached the critical point

⋆ Supported in part by NSF grants CCR-0098752, ANI-0123705, CNS-0325353, and CCF-0448196.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 460–471, 2005.

where power management, and in particular temperature management, is one of the main criteria driving device design.

There is an extensive literature on power management in computing devices, see for example [4, 12, 16]. However, the vast majority of this literature focuses on power management to conserve energy, and not on power management to reduce temperature. Temperature and energy are different physical entities with quite different properties. Bluntly, if the processor in your mobile device exceeds its energy bound, your battery is exhausted; If your processor exceeds it thermal threshold, the processor dies. Power management schemes for conserving energy focus on reducing cumulative power, while power management schemes for reducing temperature must focus more on instantaneous power. Therefore power management schemes designed to conserve energy may not perform as well when the goal is to reduce temperature. In fact, many low-power techniques are reported to have little or no effect on temperature [15]. Temperature aware design is therefore a distinct, albeit related, area of study to energy aware design [15].

Both in academic research and practice, dynamic voltage/frequency/speed scaling is the dominant technique for power management. Speed scaling involves dynamically changing the speed of the processor. Current microprocessors from AMD, Intel and Transmeta allow the speed of the microprocessor to be set dynamically. Some modern processors already are able to sense their own temperature, and thus can be slowed down or shut down so the processor temperature will stay below its thermal threshold [15]. In this paper we study speed scaling strategies to manage temperature.

1.1 Problem Formulation

We need to model the cooling behavior of a device. Cooling is a complex phenomenon that can not be modeled completely accurately by any simple model [14]. Still we require some first order simple approximation. We assume that the rate of cooling follows Fourier's Law, which states that the rate of cooling is proportional to the difference in temperature between the object and the ambient environment temperature. We assume that the environment has a fixed temperature, and that temperature is scaled so that the ambient temperature is zero. A first order approximation for rate of change T' of the temperature T is then $T' = aP - bT$, where P is the supplied power, and a, b are constants [14].

We make the standard assumption that if the processor is run at speed s, then the power consumption is $P(s) = s^\alpha$ for some constant $\alpha > 1$ [4, 2, 11, 1]. For CMOS based devices, which will likely remain the dominant technology for the near term future, the well known cube-root rule is that the speed s is roughly proportional to the cube-root of the power P, or equivalently, $P(s) = s^3$, i.e., the power is proportional to the cube of the speed [4].

The input is a collection of tasks, where each task i has a release time r_i when it arrives into the system, an amount of work w_i that must be performed to complete the task, and a deadline d_i for completing this work. In the online version of the problem, the scheduler learns about a task only at its release time; at this time, the scheduler also learns the exact work requirement and

the deadline of the task. In some settings, for example, the playing of a video or other multimedia presentation, there may be natural deadlines for the various tasks imposed by the application. In other settings, the system may impose deadlines to better manage tasks [5]. A schedule specifies which task to run at each time, and at what speed that task should be run. The work performed on a job is the integral over time of the speed that this job is run at that time. The schedule must be feasible, that is, w_i units of work must be performed on task i during the time interval $[r_i, d_i]$. This is always possible since the processor can run at any speed. Preemption is allowed, that is, the scheduler may suspend a task and then later restart the task from the point of suspension.

Energy is power integrated over time. In the energy problem, the goal is to find a feasible schedule that minimizes the total energy used. In the temperature problem, the goal is to find a feasible schedule that minimizes the the maximum temperature that the device reaches over the whole history of the schedule. That is, we want to determine the least thermal threshold that will allow us to complete these tasks. We assume that the initial temperature of the device is the ambient temperature, although this assumption is not crucial.

A schedule is c-competitive for a particular objective function if the value of that objective function on the schedule is at most c times the value of the objective function on the optimal schedule. A scheduling algorithm is c-competitive if its output is c-competitive for all instances.

1.2 Results

Theoretical investigations into speed scaling techniques to manage energy were initiated in [18]. They showed that there is a simple greedy offline algorithm, which we call YDS, that generates the feasible schedule (for all α) that uses minimum energy. We describe the algorithm YDS in section 2. [18] also proposed two online algorithms Average Rate (AVR), and Optimal Available (OA). The algorithm Average Rate (AVR) runs each task i at a rate of $w_i/(d_i - r_i)$. The algorithm Optimal Available (OA) at any point of time schedules the unfinished work optimally (say using YDS) under the assumption that no more tasks will arrive. AVR and OA were proved $O(1)$-competitive with respect to energy in [18] and [2], respectively. In [2] another online algorithm, which we call BKP, was proposed, and shown to be $O(1)$-competitive with respect to energy. The motivation for introducing BKP was that it has a lower competitive ratio, with respect to energy, than do AVR or OA when α is large. We postpone a description of the BKP algorithm until section 4.

Theoretical investigations into speed scaling techniques to manage temperature were initiated in [2]. In [2] it was shown that in principle the offline problem can be solved in polynomial time (modulo numerical stability issues) using the Ellipsoid algorithm. The Ellipsoid algorithm is useful for establishing theoretical results, but it is practically inefficient for moderate sized problems. Due to the complicated nature of Fourier's law, it seems unlikely that one can easily compute the optimal temperature schedule.

In section 2 we show that while the YDS schedule may not be optimal for temperature, it is an $O(1)$-approximate schedule with respect to temperature. Thus, this constructively shows that there are schedules that are $O(1)$-approximate with respect to both of the dual criteria: temperature and energy. In some sense, this is the best result possible, as even for some 1 job instances it is not to difficult to see that there does not exist a schedule that is $(1 + \epsilon)$-approximate for both energy and temperature (see Lemma 5 for details).

In section 3 we show that that online algorithms OA and AVR, proposed in [18] in the context of energy management, are not $O(1)$-competitive with respect to temperature. Recall that there algorithms are $O(1)$-competitive with respect to energy. This demonstratively illustrates the observation from practice that power management techniques that are effective for managing energy may not be effective for temperature. We also show that an intuitive temperature management algorithm, running at such a speed so that the temperature is constant, is surprisingly not $O(1)$-competitive with respect to temperature.

In section 4 we give our main result. We show that the online speed scaling algorithm, that we call BKP, proposed in [2] in the context of energy management, is $O(1)$-competitive with respect to temperature. This is the first $O(1)$-competitiveness analysis with respect to temperature for an online algorithm. Another way to interpret this result is that if BKP exceeds the thermal threshold T_{max} of the device then every other feasible schedule causes the device to reach a temperature of $\Omega(T_{max})$. Our temperature analysis of the online algorithm BKP compares BKP to YDS, and thus our temperature analysis of YDS is also necessary component to our temperature analysis of BKP.

At first inspection it seems difficult to reason about temperature because of the exponential decay nature of Fourier's law. All of our temperature analyses use the observation that the maximum temperature can be approximated within a constant by a times the maximum energy used over an interval of length $\Theta(1/b)$. We believe that this observation nicely explains the different natures of energy and temperature management, greatly simplifies the task of reasoning about temperature, and will surely be useful in future investigations in algorithmic issues in temperature management.

The paper [18] did not contain a proof that the YDS algorithm produces the most energy efficient feasible schedule. And to the best of our knowledge, no such proof has appeared in the literature. We show in section 5 that the correctness of YDS is an elegant consequence of the well-known KKT optimality conditions. This illustrates the utility of the KKT optimality conditions in power management problems. As another example, KKT optimality conditions can be used to simply some proofs in [13].

1.3 Further Related Results

The YDS schedule also minimizes the maximum speed, over all times in the schedule, that the processor runs at that time [18]. The BKP online algorithm is optimally (among deterministic online algorithms) e-competitive for this objective function [2].

[10] studies online speed scaling algorithms to minimize energy usage in a setting where the device also has a sleep state, and gives an offline polynomial-time 2-approximation algorithm. [10] also gives an $O(1)$-competitive online algorithm, which uses as a subroutine an algorithm for pure dynamic speed scaling. These results have been extended to the case of multiple slow-down states in [1].

[13] give an efficient algorithm for the problem of minimizing the average flow time of a collection of dynamically released equi-work processes subject to the constraint that a fixed amount of energy is available.

2 YDS Temperature

Our goal is to show that the energy optimal YDS schedule produced by the YDS algorithm is 20-competitive with respect to temperature. We first start with a description of the YDS algorithm. We then show in Lemma 1 that the optimal maximum temperature of a schedule is within a factor of 4 of the a times the maximum energy expended of an interval of length $c = \frac{\ln 2}{b}$. We call such an interval a c-interval. Thus it is sufficient to show that YDS is 5-competitive with respect to the maximum energy expended over any c-interval.

YDS Algorithm: Let $w(t_1, t_2)$ denote the work that has release time $\geq t_1$ and has deadline $\leq t_2$. The *intensity* of the interval $[t_1, t_2]$ is then $w(t_1, t_2)/(t_2 - t_1)$. The YDS algorithm repeats the following steps: Let $[t_1, t_2]$ be the maximum intensity interval. The processor will run at speed $w(t_1, t_2)/(t_2 - t_1)$ during $[t_1, t_2]$. Then the instance is modified as if the times $[t_1, t_2]$ didn't exist. That is, all deadlines $d_i > t_1$ are reduced to $\max(t_1, d_i - (t_2 - t_1))$, and all release times $r_i > t_1$ are reduced to $\max(t_1, r_i - (t_2 - t_1))$, and the process is repeated. Given the speed as a function of time as determined by this procedure, YDS always runs the released, unfinished task with the earliest deadline.

Note that the YDS schedule has the property that each task is run at a fixed rate when the task is run. This rate is fixed with respect to time, but may be different for different tasks.

Lemma 1. *For any schedule, if E denotes a times the maximum energy in a c-interval, then $E/2 \leq T_{max} \leq 2E$.*

Proof. Let $P(t)$ be the power at time t. We rewrite Fourier's Law as

$$d(e^{bt}T) = ae^{bt}P(t)dt \tag{1}$$

Let $k = cb = \ln 2$. Define T_{max} to be the maximum temperature reached for the schedule, and E be defined to be a times the maximum energy used any c-interval. We first show that T_{max} is at most twice E. Suppose that temperature T_{max} is achieved at time t_0. Then integrating equation 1 from $t_0 - k/b = t_0 - c$ to t_0, we get $T(t_0)e^{bt_0} - T(t_0 - k/b)e^{bt_0 - k} = a\int_{t_0-k/b}^{t_0} e^{bt}P(t)dt$. As e^{bt} is increasing in t, the term $\int_{t_0-k/b}^{t_0} e^{bt}P(t)dt$ is at most $e^{bt_0}\int_{t_0-k/b}^{t_0} P(t)dt$. Thus, $T(t_0) \leq T(t_0 - k/b)e^{-k} + a\int_{t_0-k/b}^{t_0} P(t)dt$. As $T(t_0 - k/b) \leq T(t_0) = T_{max}$, it follows

that $T_{max}(1-e^{-k}) \leq a\int_{t_0-k/b}^{t_0} P(t)dt \leq E$, and hence $T_{max} \leq \frac{E}{1-e^{-k}} = 2E$, as $k = \ln 2$.

We now want to show T_{max} is at least half of E. Let $[t_0 - k/b, t_0]$ be a c-interval where the maximum amount of energy is used. Integrating equation 1 gives $T(t_0)e^{bt_0} - T(t_0 - k/b)e^{bt_0-k} = a\int_{t_0-k/b}^{t_0} e^{bt}P(t)dt$. Using the fact that $T(t_0 - k/b) \geq 0$, and that e^{bt} is an increasing function of t, and the definition of E, it follows that $T(t_0)e^{bt_0} \geq a\int_{t_0-k/b}^{t_0} e^{bt}P(t)dt \geq ae^{bt_0-k}\int_{t_0-k/b}^{t_0} P(t)dt = e^{bt_0-k}E$. Thus, $T_{max} \geq T(t_0) \geq e^{-k}E = E/2$.

Note that YDS is not optimal for minimizing the maximum temperature, or for minimizing the total energy in a c-interval. The fact that YDS is not optimal for temperature can be seen on 1 job instances where the in the optimal temperature schedule must run at a speed that follows some non-constant Euler curve [2] (see Lemma 5 for more details). The fact that YDS is not optimal for minimizing energy used in a c-interval can be seen from the following instance. Consider for example, an instance with 2 tasks with work 1 each, both of them arrive at time 0 and have deadlines $c/2$ and $3c/2$ respectively.

For the rest of this section we only consider the objective of minimizing the maximum energy in any c-interval. This will culminate in Lemma 3, which states that the YDS schedule is 5 approximate with respect to this objective.

We first require some preliminary definitions and observations. Let I denote a problem instance. Let $YDS(I)$ be the YDS schedule on input I. Let $Y(I)$ denote the maximum energy used in any c-interval in $YDS(I)$. Let $Opt(I)$ denote the optimum value of the maximum energy used in any c-interval, over all feasible schedules. Let $X(I)$ denote a c-interval in $YDS(I)$ that uses energy $Y(I)$. Let $r_0 = (\epsilon Y(I)/c)^{1/\alpha}$. The value of ϵ will be fixed later. Call a task in I, *slow* , if it runs at rate strictly less than r_0 in $YDS(I)$. This notion is well defined because a task runs at constant rate in the YDS schedule. The rest of the tasks are called *fast*. Define I' to consist of exactly the fast tasks in I. Let $r'(t)$ denote the rate at time t in $YDS(I')$, and $r(t)$ denote the rate at time t in $YDS(I)$.

Define an *island* to be a maximal interval of time where $r'(t) > 0$. The following two claims easily follow from the nature of the YDS algorithm.

Claim. For all times t, $r'(t) = r(t)$ if $r \geq r_0$ and $r'(t) = 0$ otherwise.

Claim. Each task in the instance I' is totally contained inside an island. That is, for each task $i \in I'$, the interval $[r_i, d_i]$ does not overlap with any time where $r'(t) = 0$.

Thus, we can view the instance I' as a collection of disjoint intervals (islands), and each task is contained in an island. For an instance $\tilde{I}$ we define its support to be $(\max_{i\in\tilde{I}} d_i - \min_{i\in\tilde{I}} r_i)$, that is, all task in an instance are contained in an interval of length equal to the support of the instance. By the energy optimality of YDS, we trivially have that,

Claim. For an instance $\tilde{I}$ that has support no more than c, $Y(\tilde{I}) = Opt(\tilde{I})$.

As most of the energy in $X(I)$ is contained is fast tasks, we have that

Lemma 2. *Let I and I' be as defined above. Then $(1-\epsilon)Y(I) \leq Y(I') \leq Y(I)$.*

We omit a formal proof of Lemma 2 due to space constraints.

We now sketch the proof that YDS is 5-competitive with respect to minimizing the maximum energy used over any c-interval.

Lemma 3. *For any instance I, $Opt(I) \geq \min(\epsilon Y(I)/2, (1-\epsilon)Y(I)/3)$. Choosing $\epsilon = 2/5$, it follows that $Opt(I) \geq Y(I)/5$.*

Proof. As it is clear that $Opt(I') \leq Opt(I)$, it suffices to show that $Opt(I') \geq \min(\epsilon Y(I)/2, (1-\epsilon)Y(I)/3)$.

Consider an island G of I' and let t be the length of G. At the YDS schedule for I' has rate at least r_0 during G, the total energy consumed by YDS and hence by any schedule for G is at least tr_0^α. If $t \geq c$, then, by an averaging argument, for any schedule for G, there is some c-interval that has energy at least $r_0^\alpha t/\lceil t/c \rceil$ which is at least $(1/2)cr_0^\alpha = \epsilon Y(I)/2$. Thus, $Opt(I') \geq \epsilon Y(I)/2$ if $t \geq c$.

If all the islands have length no more than c, then consider the islands that intersect $X(I)$. If some such island G has energy at least $(1-\epsilon)Y(I)/3$, the result follows by Claim 2. In the case that all islands that intersect $X(I)$ have energy less than $(1-\epsilon)Y/3$. By Lemma 2 we know that in $YDS(I')$ the total energy during $X(I)$ is at least $(1-\epsilon)Y(I)$. As at most two islands can lie partially in $X(I)$, at least $(1-\epsilon)Y(I)/3$ energy is in islands that are totally contained inside $X(I)$, and hence the result follows by Claim 2.

The following theorem is then a direct consequence of Lemmas 1 and 3.

Theorem 1. *The energy optimal algorithm YDS is 20-competitive with respect to maximum temperature.*

3 Online Algorithms That Are Not $O(1)$-Competitive

We show that AVR, OA and the constant temperature algorithm are not $O(1)$-competitive.

Lemma 4. *The online algorithms AVR, OA are not $O(1)$-competitive with respect to temperature. More precisely, the competitive ratio of these algorithms must depend on either the number of jobs, or the cooling rate b.*

Proof. We use a variation of an instance from [18]. Let $r_i = 1 - 1/i$, $w_i = 1/i$ and $d_i = 1$ for $1 \leq i \leq n$. For instances with a common deadline, as is the case here, AVR and OA behave identically. Let $c = 1/n$. The YDS schedule runs jobs at a constant rate of 1. Using Lemma 1 it is sufficient to show that there is some c-interval where the energy used by OA and AVR is $\omega(c)$. In particular, during the c-interval $[1-1/n, 1]$ it is the case that AVR and OA are running at a rate of $\Omega(\log n)$, and hence the energy used during this c-interval is $\Omega(c \log^\alpha n)$.

In the reasonable case that the thermal threshold T_{max} of the device is known, the most obvious temperature management strategy is to run at a rate that leaves

the temperature fixed at T_{max}. We call such a strategy $O(1)$-competitive if on any instance I on which this constant temperature algorithm missed a deadline, every feasible schedule causes a temperature of $\Omega(T_{max})$.

Lemma 5. *The speed scaling algorithm that runs at such a speed that the temperature remains constant at the thermal threshold T_{max} is not $O(1)$-competitive.*

Proof. Suppose at time 0 a task with work x (which will be specified later) and deadline ϵ arrives. We will think of ϵ as going to 0. Suppose that the temperature at time 0 is 0. We choose x such that it is equal to the maximum work that the adversary can get done by time ϵ while keeping the temperature below T_{max}/k. Using equations from [2] for the case that $\alpha = 3$, which probably are too involved to repeat here, $x = \Theta((\frac{bT_{max}}{ka})^{1/3}\epsilon^{2/3})$. The crucial fact in the term above is that the maximum work that the adversary can do depends on ϵ as $\epsilon^{2/3}$. On the other hand, the constant temperature algorithm at temperature T_{max} has power $P = bT_{max}/a$ and hence speed $(bT_{max}/a)^{1/3}$ and work $\Theta((bT_{max}/a)^{1/3}\epsilon)$, which depends linearly on ϵ. Thus, for any constant k, the ratio of the work completed the adversary over the work completed by online goes to infinity as ϵ goes to 0.

4 The Online BKP Algorithm

In this section we first introduce some notation, then state the BKP algorithm, and then show in Theorem 2 that BKP is $O(1)$-competitive with respect to temperature. Theorem 2 compares the maximum energy used in a c-interval by BKP to the maximum energy used in a c-interval by YDS. Thus our analysis of YDS is a component of our analysis of BKP. It was shown in [2] that BKP always produces a feasible schedule. We assume without loss of generality that all release times and deadlines are integers.

Let $w(t, t_1, t_2)$ denote amount of work that has arrived by time t, that has release time $\geq t_1$ and deadline $\leq t_2$. Let $k(t)$ be the maximum over all $t' > t$ of $(w(t, et-(e-1)t', t'))/(e(t'-t))$. Note that $w(t, t_1, t_2)$ and $k(t)$ may be computed by an online algorithm at time t.

BKP Algorithm Description: At time t, work at rate $ek(t)$ on the unfinished job with the earliest deadline.

Theorem 2. *The online algorithm BKP is $20e^{\alpha}2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^{\alpha}+1)$-competitive with respect to temperature.*

Proof. By Theorem 1, it is sufficient to show that BKP uses at most factor of $e^{\alpha}2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^{\alpha}+1)$ times as much energy over any c-interval as does YDS. Again let Y be the maximum energy used by YDS in any c-interval.

Let $w(t_1, t_2)$ denote the work that has release time $\geq t_1$ and has deadline $\leq t_2$. Let $y(t)$ denote the rate of work of the algorithm YDS at time t. Let $q(t)$ be the maximum over all t_1 and t_2, such that $t_1 < t < t_2$, of $w(t_1, t_2)/(t_2 - t_1)$.

For purposes of analysis, we will assume BKP always runs at a rate $q(t) \geq k(t)$, even if there is no work to do. It is obvious that $q(t) \geq k(t)$ for all t. We can then assume that in the input instance it is the case that $y(t)$ amount of work arrives with release time t and deadline $t+1$. Under this transformation, the YDS schedule remains the same, and $q(t)$ will not decrease. The fact that $q(t)$ will not decrease follows from the definition of $q(t)$.

Let X be an arbitrary c-interval. Let X_-^k (respectively X_+^k) denote the k^{th}, c-interval immediately to the left (respectively right) of X. That is, the left endpoint of X_-^k is kc units to the left of X. Let the interval Z be defined to be $X \cup X_-^1 \cup X_+^1$. We now show that the energy used by BKP during X is at most $e^\alpha 2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^\alpha+1)Y$. As X is an arbitrary c-interval, this will imply the result.

We decompose the original instance as follows: Let $w_1(t) = y(t)$ if $t \in Z$ and 0 at all other times. Let $w_2(t) = y(t) - w_1(t)$ for all t. Let

$$q_1(t) = \max_{t' < t \leq t''} \frac{\sum_{x=t'}^{t''} w_1(t)}{t'' - t'} \qquad \text{and} \qquad q_2(t) = \max_{t' < t \leq t''} \frac{\sum_{x=t'}^{t''} w_2(t)}{t'' - t'}$$

Note that $q(t) \leq q_1(t) + q_2(t)$ since $y(t) = w_1(t) + w_2(t)$ for all each t. By convexity of the speed to power function $P(s)$, it follows that $q(t)^\alpha \leq (q_1(t) + q_2(t))^\alpha \leq 2^{\alpha-1}(q_1(t)^\alpha + q_2(t)^\alpha)$ and thus $\sum_{t \in X} q(t)^\alpha \leq 2^{\alpha-1}(\sum_{t \in X} q_1(t)^\alpha + \sum_{t \in X} q_2(t)^\alpha)$.

We now wish to upper bound the sum $\sum_{t \in X} q_1(t)^\alpha$. We follow the method used in [2]. Since $X \subseteq Z$, it is the case that $\sum_{t \in X} q_1(t)^\alpha \leq \sum_{t \in Z} q_1(t)^\alpha$. Let $l(t)$ be the maximum over all t_1, such that $t_1 < t$, of $\sum_{x=t_1}^{t} w_1(x)/(t - t_1)$. Similarly, let $r(t)$ be the maximum over all t_2, such that $t \leq t_2$, of $\sum_{x=t}^{t_2} w_1(x)/(t_2 - t)$. Clearly, $q(t) \leq \max(l(t), r(t))$, and hence $q(t)^\alpha \leq l(t)^\alpha + r(t)^\alpha$. We will show that both $\sum_{t=-\infty}^{\infty} l(t)^\alpha$, and $\sum_{t=-\infty}^{\infty} r(t)^\alpha$, are at most $(\frac{\alpha}{\alpha-1})^\alpha \sum_{t>0} y(t)^\alpha = (\frac{\alpha}{\alpha-1})^\alpha \sum_{t \in Z} w_1(t)^\alpha$. This implies $\sum_{t \in Z} q_1(t)^\alpha \leq 2\left(\frac{\alpha}{\alpha-1}\right)^\alpha \sum_{t \in Z} w_1(t)^\alpha$.

The following result was first proved by Hardy and Littlewood in [8] and later simplified by Gabriel [6]. It can also be found in [9–Theorem 393 and 394].

Fact: If $y(1), \ldots, y(n)$ are arbitrary non-negative integers, let $l(t)$ be the maximum over all v, such that $1 \leq v \leq t$, of $\sum_{k=v}^{t} y(k)/(t - v + 1)$. Let $s(y)$ be a positive increasing function of y. Let $\bar{y}(1), \ldots \bar{y}(n)$ denote the sequence of $y(i)$ in decreasing sorted order. Then $\sum_{k=1}^{n} s(l(k)) \leq \sum_{k=1}^{n} s(\sum_{j=1}^{k} \bar{y}(j)/k)$.

We now apply this fact. We rescale time so that these times in Z are $1, \ldots, 3c$. We set $y(i)$ equal to $w_1(i)$ in the fact above. Further, we set $s(y) = y^\alpha$. Note that since $w_1(t) = 0$ for $t \notin Z$, no other work affects $l(t)$ other than the $y(t)$'s for $t \in Z$. Thus the definition of $l(t)$ in the fact is identical to the previous definition of $l(t)$. We can then conclude that $\sum_{t \in Z} l(t)^\alpha$ is maximized if for all i, $y(i) \geq y(i+1)$. In this case, $l(t) = \sum_{k=1}^{t} y(k)/t$. Thus, $\sum_{t \in Z} l(t)^\alpha \leq \sum_{t \in Z} \left(\sum_{i=1}^{t} y(i)/t\right)^\alpha$.

Hardy showed [7] as a special case of Hilbert's theorem (see [9–Theorem 326]), that $\sum_t \left(\sum_{i=1}^{t} y(i)/t\right)^\alpha \leq (\frac{\alpha}{\alpha-1})^\alpha \sum_t y(t)^\alpha$. Note that in these inequalities the $y(i)$'s may be arbitrary, and all that is required of α is that $\alpha > 1$. Thus

it follows that $\sum_{t\in Z} l(t)^\alpha \leq \left(\frac{\alpha}{\alpha-1}\right)^\alpha \sum_{t\in Z} w_1(t)^\alpha$. A similar analysis of $r(t)$ shows that $\sum_{t\in Z} r(t)^\alpha \leq \left(\frac{\alpha}{\alpha-1}\right)^\alpha \sum_{t\in Z} w_1(t)^\alpha$. Thus, $\sum_{t\in Z} q_1(t)^\alpha \leq 2\left(\frac{\alpha}{\alpha-1}\right)^\alpha \sum_{t\in Z} w_1(t)^\alpha$. Finally by the definition of $w_1(t)$ and $y(t)$, we have that $\sum_{t\in Z} w_1(t)^\alpha = \sum_{t\in Z} y(t)^\alpha \leq 3Y$. Thus $\sum_{t\in X} q_1(t)^\alpha \leq 6\left(\frac{\alpha}{\alpha-1}\right)^\alpha Y$.

We now bound the term $\sum_{t\in X} q_2(t)^\alpha$. Note that $w_2(t) = 0$ at all times $t \in Z$. By the definition of Y, any c-interval contains at most $c(Y/c)^{1/\alpha}$ amount of work in the YDS schedule. Thus, any c-interval X^k_+ or X^k_- for $k \geq 2$ contains at most $c(Y/c)^{1/\alpha}$ work in $w_2(t)$, it follows that $q_2(t) \leq (Y/c)^{1/\alpha}$. Thus

$$\sum_{t\in X} q_2(t)^\alpha \leq \sum_{t\in X} \left((\frac{Y}{c})^{1/\alpha}\right)^\alpha = \sum_{t\in X} \frac{Y}{c} = Y$$

Combining the above, we have that for any c-interval X

$$\sum_{t\in X} k(t)^\alpha \leq \sum_{t\in X} 2^{\alpha-1}(q_1(t)^\alpha + q_2(t)^\alpha) \leq 2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^\alpha + 1)Y$$

Since the BKP algorithm works at rate at most $ek(t)$, the energy used during the c-interval X is at most $e^\alpha 2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^\alpha + 1)Y$.

5 Using KKT to Prove YDS Correct

We show that the energy optimality of the YDS schedule follows as a direct consequence of the well known KKT optimality conditions for convex programs. We start by stating the KKT conditions. We then show how to express the energy problem as a convex program. And then show the result of applying the KKT conditions to this convex program.

Consider a convex program

$$\begin{aligned} &\min f_0(x) \\ &\quad f_i(x) \leq 0 \qquad i = 1, \ldots, n \end{aligned}$$

Assume that this program is strictly feasible, that is, there is some point x where where $f_i(x) < 0$ for $i = 1, \ldots, n$. Assume that the f_i are all differentiable. Let λ_i, $i = 1, \ldots, n$ be a variable (Lagrangian multiplier) associated with the function $f_i(x)$. Then a necessary and sufficient KKT conditions for solutions x and λ to be primal and dual feasible are [3]:

$$f_i(x) \leq 0 \qquad i = 1, \ldots, n \tag{2}$$

$$\lambda_i \geq 0 \qquad i = 1, \ldots, n \tag{3}$$

$$\lambda_i f_i(x) = 0 \tag{4}$$

$$\nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) = 0 \tag{5}$$

To state the energy minimization problem as a convex program, we break time into intervals $t_0, \ldots t_m$ at release times and deadlines of the tasks. Let $J(i)$ be the tasks that can feasibly be executed during the time interval $I_i = [t_i, t_{i+1}]$, and $J^{-1}(j)$ be intervals during which task j can be feasibly executed. We introduce a variable $w_{i,j}$, for $j \in J(i)$, that represents the work done on task j during time $[t_i, t_{i+1}]$. Our (interval indexed) mathematical program P is then:

$$\min \quad E \tag{6}$$

$$w_j \leq \sum_{i \in J^{-1}(j)} w_{i,j} \qquad j = 1, \ldots, n \tag{7}$$

$$\sum_{i=1}^{m} \left(\frac{\sum_{j \in J(i)} w_{i,j}}{t_{i+1} - t_i} \right)^{\alpha} (t_{i+1} - t_i) \leq E \tag{8}$$

$$w_{i,j} \geq 0 \qquad i = 1, \ldots, m \qquad j \in J(i) \tag{9}$$

We now apply the KKT conditions to this convex program. We associate a dual variable δ_j with equation j in line 7, a dual variable β with the equation in line 8, and a dual variable $\gamma_{i,j}$ with equation i, j in line 9. We now evaluate line 5 of the KKT conditions for our convex program. Looking at the component of equation 5 corresponding to the variable E, we get that $\beta = 1$. Looking at the component of equation 5 corresponding to the variable $w_{i,j}$, we get that

$$-\delta_j + \beta p \left(\frac{\sum_{k \in J(i)}^{n} w_{i,k}}{t_{i+1} - t_i} \right)^{\alpha - 1} - \gamma_{i,j} = 0 \tag{10}$$

Consider a $w_{i,j}$ such that $w_{i,j} > 0$. We know that by complementary slackness (equation 4) that it must be the case that $\gamma_{i,j} = 0$. Hence,

$$\delta_j = \alpha \left(\frac{\sum_{k \in J(i)} w_{i,k}}{t_{i+1} - t_i} \right)^{\alpha - 1} \tag{11}$$

Hence, the interpretation of the dual variable δ_j is α times the speed at which the processor runs during interval i raised to the power of $(\alpha - 1)$. This quantity, and hence the speed of the processor, must be the same for each interval i during which task j is run. Now consider a $w_{i,j}$ such that $w_{i,j} = 0$. Rearranging equation 10 we find that

$$\gamma_{i,j} = \alpha \left(\frac{\sum_{k \in J(i)} w_{i,k}}{t_{i+1} - t_i} \right)^{\alpha - 1} - \delta_j \tag{12}$$

Then $\gamma_{i,j}$ will be non-negative if the processor is running faster during interval i than during the intervals where task j is run.

Thus we can conclude that a sufficient condition for a primal feasible solution to be optimal is that:

- For each task j, the processor runs at the same speed, call is s_j in the intervals i in which task j is run.
- And the processor runs at speed no less than s_j during intervals i, such that $j \in J(i)$, in which task j is not run.

The solution produced the the YDS algorithm clearly has these properties and hence is optimal.

References

1. J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *IEEE Syposium on Foundations of Computer Science*, 2004.
2. N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *IEEE Syposium on Foundations of Computer Science*, 2004.
3. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
4. D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.
5. G. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer, 1997.
6. R. M. Gabriel. An additional proof of a maximal theorem of hardy and littlewood. *Journal of London Mathematical Society*, 6:163–166, 1931.
7. G. H. Hardy. Note on a theorem of hilbert. *Math. Zeitschr.*, 6:314–317, 1920.
8. G. H. Hardy and J. Littlewood. A maximal theorem with function-theoretic applications. *Acta Mathematica*, 54:81–116, 1930.
9. G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1952.
10. S. Irani, , R. K. Gupta, and S. Shukla. Algorithms for Power Savings. In *ACM/SIAM Symposium on Discrete Algorithms*, 2003.
11. S. Irani, S. Shukla, and R. Gupta. Online strategies for dynamic power management in systems with multiple power saving states. *Trans. on Embedded Computing Sys.*, 2003. Special Issue on Power Aware Embedded Computing.
12. T. Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.
13. K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. In *Scandanavian Workshop on Algorithms and Theory*, 2004.
14. J. E. Sergent and A. Krum. *Thermal Management Handbook*. McGraw-Hill, 1998.
15. K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, pages 2–13, 2003.
16. V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Design Automation Conference*, pages 732–737, 1998.
17. http://www.usatoday.com/tech/news/2004-05-07-intel-kills-tejas_x.htm.
18. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Syposium on Foundations of Computer Science*, page 374, 1995.

The Complexity of Solving Linear Equations over a Finite Ring

V. Arvind and T.C. Vijayaraghavan

Institute of Mathematical Sciences, C.I.T Campus, Chennai, India 600 113
{arvind, tcvijay}@imsc.res.in

Abstract. In this paper we first examine the computational complexity of the problem LCON defined as follows: given a matrix A and a column vector $\mathbf{b}$ over $\mathbb{Z}$, determine if $A\mathbf{x} = \mathbf{b}$ is a feasible system of linear equations over $\mathbb{Z}_q$. Here q is also given as part of the input by its prime factorization $q = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, such that each $p_i^{e_i}$ is tiny (i.e. given in unary). In [MC87] an NC^3 algorithm is given for this problem. We show that in fact the problem can be solved in randomized NC^2. More precisely, we show that LCON is in the nonuniform class $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$. Combined with the hardness of LCON for $\mathrm{L}^{\mathrm{GapL}}$, we have a fairly tight characterization of the complexity of LCON in terms of logspace counting classes. We prove the same upper bound results for the problem of testing feasibility of $A\mathbf{x} = \mathbf{b}$ over finite rings R with unity, where R is given as part of the input as a table.

1 Introduction

We study the computational complexity of the following problem LCON (for linear congruences):

Given as input a triple $(A, \mathbf{b}, q)$, where A is an integer matrix, $\mathbf{b}$ is an integer column vector, and q is a positive integer, determine if $A\mathbf{x} = \mathbf{b}$ is a feasible system of linear equations over $\mathbb{Z}_q$. Here q is given by its prime factorization $q = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, and it is assumed to satisfy the additional property that each $p_i^{e_i}$ is tiny (i.e. given in unary).

It has been a rich and fruitful line of research in the last decade to classify problems with efficient parallel algorithms using logarithmic space-bounded classes, and specifically, logspace counting classes. A well-known example is the problem of computing the determinant over integers [Tod91, Vin91] that is captured exactly by the complexity class GapL. Computing determinants over finite fields of characteristic p is captured by the class $\mathrm{Mod}_p\mathrm{L}$ [BDHM92]. Furthermore, the results of [ABO99] classify important linear algebraic problems like checking feasibility of a system linear equations over rationals, verifying if the rank of a matrix is r, and several other problems using logspace counting classes.

The problem LCON is defined[1] and studied in [MC87], and it plays a pivotal role in placing a number of abelian permutation group problems in the

[1] It is also mentioned in [Dic92] from a number-theoretic perspective.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 472–484, 2005.

complexity class NC (NC is the class of problems that can be solved in polylogarithmic time with polynomially many processors). Both LCON and the related problem of finding a solution to $A\mathbf{x} = \mathbf{b} \pmod q$ are shown to be in NC^3 in [MC87]. The basic idea used there is to solve $A\mathbf{x} = \mathbf{b} \pmod{p_i^j}$, by first solving $A\mathbf{x} = \mathbf{b} \pmod{p_i}$ and then "lifting" the solution (essentially Hensel lifting) repeatedly to solutions modulo p_i^j for increasing values of j, until a solution to $A\mathbf{x} = \mathbf{b} \pmod{p_i^{e_i}}$ is obtained. The solutions for different prime powers p^{e_i} can then be combined using the Chinese remainder theorem.

In the present paper we will describe a randomized parallel algorithm that avoids the lifting process mentioned above, and hence places the problem in randomized NC^2. Alternatively, we can use the logspace counting class GapL, which is the class of functions $f : \Sigma^* \longrightarrow \mathbb{Z}$ such that for some nondeterministic logspace-bounded Turing machine M, the function $f(x) = acc_M(x) - rej_M(x)$ for every $x \in \Sigma^*$, to argue that LCON is in $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$. We have shown in [AV04] that LCON is hard for $\mathrm{L}^{\mathrm{GapL}}$. These upper and lower complexity bounds will carry over to certain abelian permutation group problems as explained in [AV04].

We first notice that, by the Chinese remainder theorem, $A\mathbf{x} = \mathbf{b} \pmod q$ is feasible if and only if $A\mathbf{x} = \mathbf{b} \pmod{p_i^{e_i}}$ is feasible for $1 \leq i \leq k$, where $q = \prod p_i^{e_i}$ is the prime factorization of q and $p_i^{e_i}$ are tiny (i.e. given in unary).

Thus, we can focus on the problem of testing if the system $A\mathbf{x} = \mathbf{b} \pmod{p^e}$ is feasible, where p is a prime and p^e is tiny. In other words, we are testing if $A\mathbf{x} = \mathbf{b}$ has a solution in the finite ring $\mathbb{Z}_{p^e}$. We first transform the problem to solving a system of linear Diophantine equations in the following proposition.

Proposition 1. *Let A be an $m \times n$ integer matrix, $\mathbf{b}$ be an m integer column vector, and p be a prime and e a positive integer. The system of linear equations $A\mathbf{x} = \mathbf{b}$ (mod p^e) is feasible (in the finite ring $\mathbb{Z}_{p^e}$) if and only if $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution in $\mathbb{Z}$.*

Proof. Clearly, if $A\mathbf{x}+p^e\mathbf{y} = \mathbf{b}$ has a solution $\mathbf{x}', \mathbf{y}'$ in $\mathbb{Z}$, then $A\mathbf{x}' = \mathbf{b} \pmod{p^e}$. Conversely, if $\mathbf{x}'$ is a solution to $A\mathbf{x} = \mathbf{b} \pmod{p^e}$, then $A\mathbf{x}'$ must be of the form $\mathbf{b}+p^e\mathbf{y}'$ for some integral vector $\mathbf{y}'$. Consequently, $(\mathbf{x}', -\mathbf{y}')$ is an integral solution to $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$.

Remark 1. Polynomial time algorithms for solving linear Diophantine equations are well known (see e.g. [Sch98]). However, the problem is not known to be in NC. It is observed in [ABO99], that testing existence of integral solutions to $A\mathbf{x} = \mathbf{b}$ is RNC reducible to checking if $gcd(a_1, a_2, \ldots, a_n) = gcd(b_1, \ldots, b_m)$ for integers a_i and b_j. It is a long standing open problem if the latter problem is in NC (even randomized NC).

However, the system $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ of linear diophantine equations has a form whose structure we will be able to exploit and avoid computation of the gcd of integers.

We consider the following ring $\mathrm{Z}_{(\mathrm{p})}$ (contained in $\mathbb{Q}$):

$$\mathrm{Z}_{(\mathrm{p})} = \{\frac{\mathrm{a}}{\mathrm{b}} \mid \mathrm{a}, \mathrm{b}, \in \mathbb{Z} \ : \ (\mathrm{a}, \mathrm{b}) = 1 \text{ and } (\mathrm{p}, \mathrm{b}) = 1\}.$$

It is the ring of rational numbers a/b such that the denominator is not divisible by the prime p.

Lemma 1. *Let A be an $m \times n$ integer matrix, $\mathbf{b}$ be an $m \times 1$ integer column vector, p be a prime and e a positive integer. The system $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution in $\mathbb{Z}$ if and only if $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution in the ring $\mathrm{Z}_{(\mathrm{p})}$.*

Proof. If $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution in $\mathbb{Z}$ then obviously that solution lies in $\mathrm{Z}_{(\mathrm{p})}$ as well.

Conversely, suppose $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution $\mathbf{x}', \mathbf{y}'$ in $\mathrm{Z}_{(\mathrm{p})}$. Each entry of $\mathbf{x}'$ and $\mathbf{y}'$ is a rational number. Let $\alpha \in \mathbb{Z}$ be the least common multiple of the denominators of the entries in $\mathbf{x}', \mathbf{y}'$. Let $\mathbf{x}'' = \alpha\mathbf{x}'$ and $\mathbf{y}'' = \alpha\mathbf{y}'$. Both $\mathbf{x}''$ and $\mathbf{y}''$ are integral vectors and it follows that

$$A\mathbf{x}'' + p^e\mathbf{y}'' = \alpha\mathbf{b}.$$

Since $\mathbf{x}', \mathbf{y}'$ is a solution in $\mathrm{Z}_{(\mathrm{p})}$, it follows that $(\alpha, p) = 1$. Thus there are integers $s, t \in \mathbb{Z}$ such that $sp^e + t\alpha = 1$. Consequently, we have $tA\mathbf{x}'' + tp^e\mathbf{y}'' = (1 - sp^e)\mathbf{b}$. Rearranging terms, we obtain $tA\mathbf{x}'' + p^e(s\mathbf{b} + t\mathbf{y}'') = \mathbf{b}$, yielding a solution in $\mathbb{Z}$.

We observe one further property of the linear system $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$. We can rewrite it as $B\mathbf{z} = \mathbf{b}$. Notice that the matrix $B = (A; p^eI)$ is an $m \times (m + n)$ matrix of rank m and $\mathbf{z} = (\mathbf{x}, \mathbf{y})$.

Proposition 2. *$A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ is a system of linear equations with coefficient matrix $[A; p^eI]$ of full row rank.*

Let B be an $m \times n$ integer matrix of full row rank, and $\mathbf{b}$ be an integral column vector. The theory of linear diophantine equations precisely characterizes when the system of linear equations $B\mathbf{z} = \mathbf{b}$ has an *integral* solution. We state the following useful characterization from [Sch98–pp. 51] and [Dic92–pp. 82].

Theorem 1. [Sch98–pp. 51] *Let B be an $m \times n$ integer matrix of full row rank, and $\mathbf{b}$ be an integral column vector. The system of linear equations $B\mathbf{z} = \mathbf{b}$ has an* integral *solution for $\mathbf{z}$ if and only if*[2] *the gcd of all the nonzero $m \times m$ subdeterminants of B equals the gcd of all the nonzero $m \times m$ subdeterminants of the augmented matrix $[B; \mathbf{b}]$.*

Intuitively, this follows from the fact that the gcd of the $m \times m$ subdeterminants of B is the volume of fundamental parallelepiped in the integral lattice

[2] Our statement is slightly different but equivalent to that in [Sch98]. For, the gcd of the $m \times m$ subdeterminants of the augmented matrix $[B; \mathbf{b}]$ will in any case divide the gcd of all the nonzero $m \times m$ subdeterminants of B.

generated by the columns of B, and the gcd of the $m \times m$ subdeterminants of $[B; \mathbf{b}]$ is the volume of fundamental parallelepiped in the integral lattice generated by the columns of $[B; \mathbf{b}]$. $B\mathbf{z} = \mathbf{b}$ is feasible if and only if $\mathbf{b}$ lies in the lattice of B and the vector $\mathbf{b}$ will lie in this lattice if and only if the volume of the fundamental parallelepiped in the lattice generated by columns of $[B; \mathbf{b}]$ equals the volume of the fundamental parallelepiped in the lattice generated by the columns of B.

Based on the above theorem, we now give a similar characterization for the feasibility of the linear equations $B\mathbf{z} = \mathbf{b}$ over $\mathrm{Z}_{(\mathrm{p})}$. This will be useful for proving our new upper bound result. For a positive integer d, let $ord_p(d)$ be the largest nonnegative integer e such that p^e divides d.

Theorem 2. *Let B be an $m \times n$ integer matrix of full row rank, and $\mathbf{b}$ be an integral column vector. Let r denote the gcd of all the nonzero $m \times m$ subdeterminants of B, and s denote the gcd of all the nonzero $m \times m$ subdeterminants of the augmented matrix $[B; \mathbf{b}]$. The system of linear equations $B\mathbf{z} = \mathbf{b}$ has a solution in $\mathrm{Z}_{(\mathrm{p})}$ if and only if $ord_p(r) = ord_p(s)$.*

Proof. Firstly, notice that s is a factor of r for any integer matrix B of full row rank and any column vector $\mathbf{b}$ (simply because B is a submatrix of $[B; \mathbf{b}]$), where s and r are defined in the statement above. Thus, we can write $r = ds$, for some integer d.

Now, suppose $B\mathbf{z} = \mathbf{b}$ is feasible over $\mathrm{Z}_{(\mathrm{p})}$. Then, by clearing denominators of the solution, it follows that there is a positive integer $\alpha \in \mathbb{Z}$ such that $(\alpha, p) = 1$ and $B\mathbf{z} = \alpha\mathbf{b}$ is feasible over $\mathbb{Z}$. Let t denote the gcd of all nonzero $m \times m$ subdeterminants of $[B; \alpha\mathbf{b}]$. Applying Theorem 1 to the system $B\mathbf{z} = \alpha\mathbf{b}$, it follows that $r = t$. Thus, $t = r = ds$. It is clear that d must divide α, as the only difference between $[B; \mathbf{b}]$ and $[B; \alpha\mathbf{b}]$ is the factor α. Now, since α and p are relatively prime, it follows that $ord_p(r) = ord_p(s)$.

Conversely, suppose $ord_p(r) = ord_p(s)$. Since B has full row rank m, the linear system $B\mathbf{z} = \mathbf{b}$ has a rational solution $\mathbf{z}'$. Let $p^e\alpha$ be the l.c.m. of the denominators of entries in $\mathbf{z}'$. Multiplying by $p^e\alpha$ on both sides of the equation $B\mathbf{z}' = \mathbf{b}$, we get $B\mathbf{z}'' = p^e\alpha\mathbf{b}$, where $\mathbf{z}''$ has integer entries. Let t denote the gcd of all $m \times m$ subdeterminants of $[B; p^e\alpha\mathbf{b}]$. By Theorem 1, applied to the system $B\mathbf{z} = p^e\alpha\mathbf{b}$, it follows that $r = t$. Thus, $r = t = ds$. But $p \nmid d$ as $ord_p(s) = ord_p(r)$. It follows that the gcd of all $m \times m$ subdeterminants of the matrix $[B; \alpha\mathbf{b}]$ is also r. Again, by Theorem 1 applied to $B\mathbf{z} = \alpha\mathbf{b}$, it follows that $B\mathbf{z} = \alpha\mathbf{b}$ has an *integral* solution (call it $\mathbf{z}_0$). As $(\alpha, p) = 1$, it follows that $\frac{1}{\alpha}\mathbf{z}_0$ is a $\mathrm{Z}_{(\mathrm{p})}$ solution to $B\mathbf{z} = \mathbf{b}$. This completes the proof.

2 The Upper Bound Result

The algorithm that we are going to describe for LCON is based on the ideas and results of Giesbrecht [Gi95, Gi97]. Specifically, we use results from [Gi95], in which a randomized polynomial time algorithm is described to compute the

Smith Normal Form of an integer matrix. We recall some definitions. Details can be found in [Sch98–Chapter 4].

A square integer matrix M is *unimodular* if $det(M)$ is ± 1. Any $m \times n$ integer matrix A is equivalent under unimodular transformations to a unique $m \times n$ integer matrix $S = (D; 0)$, where D is an $m \times m$ integer diagonal matrix. More precisely, there are *unimodular* integer matrices P and Q of appropriate sizes such that $S = PAQ$. The matrix S is called the *Smith Normal Form* of A. If r is the rank of A, then the diagonal matrix D has diagonal $diag(s_1, \ldots, s_r, 0, \ldots, 0)$ where $s_i \neq 0$ for $1 \leq i \leq r$, such that $s_i | s_{i+1}$ for each i. Furthermore, if d_k denotes the gcd of all $k \times k$ minors of A, for $1 \leq k \leq r$, then $s_1 = d_1$ and $s_k = d_k / d_{k-1}$ for $2 \leq k \leq r$. The number d_k is the k^{th} *determinantal divisor* of A, $1 \leq k \leq r$, and s_k are the *invariant factors* of A.

We can now give a straightforward reformulation of the characterization of Theorem 2 for the feasibility of $B\mathbf{z} = \mathbf{b}$ over $\mathrm{Z}_{(\mathrm{p})}$ in terms of determinantal divisors.

Theorem 3. *Let B be an $m \times n$ integer matrix of full row rank, and $\mathbf{b}$ be an integral column vector of length m. Let d_m be the m^{th} determinantal divisor of B and d'_m be the m^{th} determinantal divisor of the augmented matrix $[B; \mathbf{b}]$. The system of linear equations $B\mathbf{z} = \mathbf{b}$ has a solution in the ring $\mathrm{Z}_{(\mathrm{p})}$ if and only if $ord_p(d_m) = ord_p(d'_m)$.*

Thus the problem of testing feasibility of $B\mathbf{z} = \mathbf{b}$ over the ring $\mathrm{Z}_{(\mathrm{p})}$ is equivalent to checking if $ord_p(d_m) = ord_p(d'_m)$, where d_m be the m^{th} determinantal divisor of B and d'_m be the m^{th} determinantal divisor of the matrix $[B; \mathbf{b}]$.

We will use the following result of Giesbrecht [Gi95] to design a randomized algorithm to test if $ord_p(d_m) = ord_p(d'_m)$, without actually computing the numbers d_m and d'_m.

Recall that the *content* $cont(f)$ of a multivariate polynomial f (over any Euclidean Domain, in particular integers) is the gcd of all the coefficients of f.

Theorem 4. [Gi95–Theorem 2.1] *Let B be an $m \times n$ integer matrix of rank r. Let $X = (X_{ij})$ be an $r \times m$ matrix and $Y = (Y_{lk})$ be an $n \times r$ matrix of distinct indeterminates X_{ij} and Y_{lk}, $1 \leq i, k \leq r$, $1 \leq j \leq m$, and $1 \leq l \leq n$. Then the content of the determinant of the t^{th} leading minor of the $r \times r$ matrix XBY equals the t^{th} determinantal divisor d_t, $1 \leq t \leq r$.*

In particular, it is clear from Theorems 3 and 4 that $d_m = cont(\det(XBY))$ and $d'_m = cont(\det(X[B; \mathbf{b}]Z)$, where Z is an $(n+1) \times r$ matrix of distinct indeterminates. Thus we have the following.

Lemma 2. *Let B be an $m \times n$ integer matrix of full row rank, and $\mathbf{b}$ be an integral column vector of length m. The system of linear equations $B\mathbf{z} = \mathbf{b}$ has a solution in $\mathrm{Z}_{(\mathrm{p})}$ if and only if $ord_p(cont(\det(XBY))) = ord_p(cont(\det(X[B; \mathbf{b}]Z))$, where X, Y, Z are matrices of indeterminates.*

We now focus on the problem of computing $ord_p(cont(\det(XBY)))$, where B is an $m \times n$ integer matrix of rank m. Notice that we cannot directly compute $\det(XBY)$ as there are exponentially many terms. Instead, following Giesbrecht [Gi95] and analogous to the Schwartz-Zippel test, the idea is to compute the determinant $\det(XBY)$, where the indeterminates in X and Y are randomly picked from a suitable domain (over which computing the determinant will be easy). We will use the following variant of the Schwartz-Zippel test (as stated in Giesbrecht [Gi95]).

Lemma 3. [Gi95–Lemma 2.2] *Let $g \in D[z_1, z_2, \ldots, z_s]$ be a nonzero polynomial, where D is an integral domain. Let W be a finite subset of D. Suppose elements $a_1, \ldots, a_s$ are picked independently at random from W such that each a_i is assigned to any one element of W with probability at most ϵ. Then* $\text{Prob}[g(a_1, \ldots, a_s) = 0; a_i \in W] \leq \epsilon \deg(g)$, *where* $\deg(g)$ *is the total degree of g.*

For ease of notation in the sequel, we denote the multivariate polynomial $\det(XBY)$ by $f(z_1, \ldots, z_s) \in \mathbb{Z}[z_1, \ldots, z_s]$, where indeterminates in X and Y have been renamed as the z_i's. Our goal is to compute $ord_p(cont(f))$. By factoring out the content of f, we can write $f(z_1, \ldots, z_s) = c \cdot g(z_1, z_2, \ldots, z_s)$, where $cont(g) = 1$. We are interested in computing $ord_p(c)$.

Now, suppose we substitute for z_i a univariate polynomial $a_i(x) \in \mathbb{Z}[x]$, $1 \leq i \leq s$. We claim that $ord_p(c) = ord_p(cont(f(a_1(x), \ldots, a_s(x))))$ if and only if $g(a_1(x), \ldots, a_s(x)) \neq 0 \pmod p$.

It follows because $ord_p(c) \leq ord_p(cont(f(a_1(x), \ldots, a_s(x))))$, and $ord_p(c) < ord_p(cont(f(a_1(x), \ldots, a_s(x))))$ if and only if $cont(g(a_1(x), \ldots, a_s(x)))$ is divisible by p.

Now, we define the following finite subset W of $\mathbb{Z}[x]$ from which we will randomly pick the polynomials a_i, and argue that with high probability we have $g(a_1(x), \ldots, a_s(x)) \neq 0 \pmod p$. Choose $\beta = 2p + 1$, and let $L = \{1, \ldots, \beta\}$. Let $\deg(g) = t$. Define $W = \{a(x) \mid \deg(a) \leq t - 1$ and coefficient of a are in $L\}$.

We now state a lemma that is a slightly modified version of [Gi95–Lemma 2.6]. For the sake of completeness, we give a proof.

Lemma 4. [Gi95] *Let g be a polynomial in $\mathbb{Z}[z_1, \ldots, z_s]$ of degree t and $cont(g) = 1$. If $(a_1, \ldots, a_s)$ are s elements chosen uniformly and independently at random from W then*

$$\text{Prob}[g(a_1, \ldots, a_s) = 0 \ (mod\ p)] \leq t(4/5)^t.$$

Proof. Let Γ be an irreducible polynomial of degree t modulo p. Consider the domain D of Lemma 3 to be the finite field $\mathbb{Z}[x]/(p, \Gamma)$ of size p^t. Notice that we can consider g to be a nonzero polynomial in $D[z_1, \ldots, z_s]$ (g is surely nonzero modulo p as its content is 1).

Likewise, we wish to consider the set W as a subset $\hat{W}$ of D: an element a of W is already a polynomial of degree at most $t - 1$, and the coefficients of a have to be reduced modulo p to get the corresponding element in $\hat{W}$. Now, if we pick an element $a \in W$ uniformly at random we wish to bound the probability that it is equal to a specific element $a' \in \hat{W}$ modulo p. Each coefficient of a,

when reduced modulo p, takes any specific value in $\mathbb{Z}_p$ with probability at most $\lceil \frac{\beta}{p} \rceil \cdot 1/\beta \leq (1/p + 1/\beta) \leq 4/5$. It follows that $\text{Prob}_{a \in W}[a = a' \pmod p] \leq (4/5)^t$.

Now, applying Lemma 3 to the polynomial $g \in D[z_1, \ldots z_s]$, we immediately get the desired probability bound.

We have the following corollary.

Corollary 1. *Let B be an $m \times n$ integer matrix of rank m. In the matrices X and Y, let each indeterminate be picked independently and uniformly at random from the set W, and let X' and Y' be the resulting matrices thus obtained. Then*

$$\text{Prob}[ord_p(cont(\det(XBY))) = ord_p(cont(\det(X'BY')))] \geq 1 - 2m(4/5)^{2m}.$$

Proof. Notice that the degree of $\det(XBY)$ is $2m$. Thus, setting $g = \det(XBY)$ and $t = 2m$ in Lemma 3 we obtain the probability bound immediately.

Now we return to the problem of LCON. Let $A\mathbf{x} = \mathbf{b} \pmod q$ be an instance of LCON, where q is given by its prime factorization that is a product of tiny factors. By Chinese remainder theorem, we can assume that q is a tiny prime power p^e. By applying Proposition 1, Lemma 1, and Theorem 3, we can easily (in logspace) transform the input to a system of linear equations $B\mathbf{z} = \mathbf{b}$, where B and $\mathbf{b}$ are integral and B is full row rank. Now, by Lemma 2 we can further transform this into the problem of checking if $ord_p(cont(\det(XBY))) = ord_p(cont(\det(X[B; \mathbf{b}]Z))$. In order to do this, we apply Corollary 1. More precisely, in order to compute $ord_p(cont(\det(XBY)))$, we pick values for the indeterminates in X and Y from the set W to obtain corresponding random matrices X' and Y'. It is easy to see that $\det(X'BY')$ is a polynomial in x of degree $2m(t-1)$. Let $\sum_{i=0}^{2m(t-1)} \mu_i x^i$ be this polynomial. Now, each coefficient μ_i of this polynomial can be computed with a GapL oracle (see e.g. [ABO99]). An L^{GapL} algorithm can keep track of the largest power of p that divides $\mu_0, \ldots, \mu_i$. When the coefficient μ_{i+1} is computed, the algorithm can also update the highest power of p that divides $\mu_0, \ldots, \mu_i, \mu_{i+1}$. Since p is small, this can done by carried out by a logspace machine with access to the GapL oracle. Thus, we can compute $ord_p(cont(\det(X'BY')))$, which is correct with high probability by Corollary 1. We can similarly compute $ord_p(cont(\det(X[B; \mathbf{b}]Z))$. Clearly, if we fix the random bits the rest of the computation is an L^{GapL} computation. Thus the problem LCON is in RNC^2. Now, after amplifying the success probability by standard methods, we can fix the random bits to get a suitable advice that will work for all inputs of a given length. It follows that LCON is also in $\text{L}^{\text{GapL}}/\text{poly}$. We have proved our main theorem.

Theorem 5. *The problem* LCON *is in* RNC^2*, and also in the nonuniform class* $\text{L}^{\text{GapL}}/\text{poly}$.

We can obtain a conditional derandomization of the above upper bound for LCON by exactly following the same arguments as in Allender et al [ARZ99, Theorem 5.5], which is based on the results from Klivans and Melkebeek [KM99].

Theorem 6. *If there is a set A in* DSPACE(n) *and an $\epsilon > 0$ such that no circuit of size smaller than $2^{\epsilon n}$ accepts exactly strings of length n in A, then* LCON *is in* $\mathrm{L}^{\mathrm{GapL}}$.

3 Linear Equations over a Finite Ring

We now study the complexity of the following general problem:

Given as input a finite ring R with unity and a system of linear equations $A\mathbf{x} = \mathbf{b}$, where A is an $m \times n$ matrix and $\mathbf{b}$ is an m-dimensional column vector over R, test if there is a solution for $\mathbf{x}$ over R. Here we assume that R is given by its addition (denoted by $+$) and multiplication (denoted by concatenation) tables. Furthermore, we assume that the additive abelian group $(R, +)$, denoted R^+ is given as a direct sum $C_1 \oplus \cdots \oplus C_r$, where each C_i is a cyclic group of prime power order.[3]

Notice that the ring R is small as its size can be encoded in unary in the size of the input. The above problem generalizes the problem of solving $A\mathbf{x} = \mathbf{b}$ modulo p^e, where p^e is tiny, as we can set $R = \mathbb{Z}_{p^e}$. In this section we show that the above problem is logspace reducible to the problem of solving $A\mathbf{x} = \mathbf{b}$ modulo composites q (with tiny prime-power factors). Thus we show that the above problem is also in the class $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$.

Remark 2. Notice that the ring R is not assumed to be commutative. The following example indicates how our claimed reduction is going to work and also motivates our approach: Let $R = M_k(\mathbb{F}_q)$, the ring of $k \times k$ matrices over the finite field $\mathbb{F}_q$. Now, consider linear equations $A\mathbf{x} = \mathbf{b}$ over $M_k(\mathbb{F}_q)$, where A is an $m \times n$ matrix and $\mathbf{b}$ an m-vector over $M_k(\mathbb{F}_q)$. By expanding each entry of $\mathbf{x}$ into a $k \times k$ block of variables (that will take values in $\mathbb{F}_q$), and likewise treating A as an $mk \times nk$ matrix and $\mathbf{b}$ as an $m \times k$ matrix, both over $\mathbb{F}_q$, we can consider the equations $A\mathbf{x} = \mathbf{b}$ as a system of linear equations over $\mathbb{F}_q$. Now, applying ideas from [ABO99], we can easily see that testing feasibility of this system is in $\mathrm{L}^{\mathrm{GapL}}$.

We proceed to show that the idea in the above remark can be extended to handle any finite ring R with unity, and reduce it to LCON.

Let $|R| = n$ and $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ be the prime factorization of n. As R is an abelian group under addition, by the fundamental theorem of finite abelian groups, $(R, +)$ can be written as a direct sum of its Sylow subgroups. Let R_i denote the p_i-Sylow subgroup of R, $1 \leq i \leq k$. Decomposing the additive group $(R, +)$ into its Sylow subgroups R_i we can write

$$R = R_1 \oplus R_2 \oplus \cdots \oplus R_k.$$

[3] By the fundamental theorem of finite abelian groups, such a decomposition exists. Indeed, for $(R, +)$ we can compute it in time polynomial in $|R|$.

Now, let $x \in R$ and $a \in R_i$. Notice that the (additive) order[4] of xa must divide $p_i^{e_i}$ as $p_i^{e_i}xa$ can be written as $x(p_i^{e_i}a)$, and $p_i^{e_i}a = 0$ since $a \in R_i$. Since $(R, +)$ is an abelian group, R_i is the set of all elements of R whose order is a power of p_i. Thus, $xa \in R_i$. Similarly, $ax \in R_i$. Therefore, each R_i is a two-sided ideal of R. Since R has unity, it follows that $RR_i = R_iR = R_i$ for each i. Furthermore, it is easy to see that for $i \neq j$, $R_iR_j = 0$. This follows because R_iR_j is contained in $R_i \cap R_j$ which contains only the additive identity 0. Putting it together, we can see that the R_i's actually yield a ring decomposition $R = R_1 \oplus R_2 \oplus \cdots \oplus R_k$. Thus, we can express each $x \in R$ uniquely as $x = x_1 + \cdots + x_k$, where $x_i \in R_i$.

There is another crucial property of R_i. Since R has unity 1, the above ring decomposition gives a unique expression for 1 as $1 = a_1 + a_2 + \cdots + a_k$, $a_i \in R_i$.

We claim that $a_i \neq 0$. Furthermore, we also claim that a_i is *not* a zero divisor in the subring R_i. To see this, consider any $y \in R_i$. We can write $y = y \cdot 1 = y(a_1 + \cdots + a_k) = ya_1 + \cdots + ya_k$. Now, since $y \in R_i$, for any $j \neq i$ it holds that $ya_j = 0$. Thus, $a_i = 0$ forces $y = 0$ for all $y \in R_i$ which is a contradiction as R_i is a p_i-Sylow subgroup of R. By the same argument, a_i cannot be a zero divisor of R_i. For, if $ya_i = 0$ for $y \in R_i$ then the above equation forces $y = 0$. We summarize our observations below.

Lemma 5. *Let R be a finite ring with unity. Then R has the ring decomposition $R = R_1 \oplus R_2 \oplus \cdots \oplus R_k$, where each R_i is a Sylow subgroup of R. Furthermore, each R_i has at least one nonzero element which is not a zero-divisor of R_i.*

Since $R = R_1 \oplus R_2 \oplus \cdots \oplus R_k$ is a direct sum decomposition, it is clear that we can decompose A and $\mathbf{b}$ in the linear system into A_i and $\mathbf{b}_i$ (which are the components of the entries of A and $\mathbf{b}$ in R_i) for each i. Thus, it follows easily that $A\mathbf{x} = \mathbf{b}$ is feasible over R if and only if $A_i\mathbf{x} = \mathbf{b}_i$ is feasible over R_i for each i. Since R is given by its addition table, we can find the ring decomposition of R even in logspace. Thus, the above reduction can be carried out it logspace.

We can henceforth assume that R is of size p^e and we have to test feasibility of $A\mathbf{x} = \mathbf{b}$ over R. Notice that R need not have unity. However, by Lemma 5 we can assume that R has at least one element which is not a zero-divisor(namely, the element a_i in R_i where $1 = \sum_{i=1}^{k} a_i$).

We now give a suitable matrix representation to a finite ring R which has an element that is not a zero divisor where $|R|$ is a prime power p^e. This will be an important step in the reduction of feasibility testing of linear equations over R to linear equations over $\mathbb{Z}_{p^e}$.

In the sequel, we denote the additive abelian group $(R, +)$ by R^+. By the fundamental theorem of finite abelian groups, the abelian p-group R^+ can be expressed as a direct sum of cyclic groups: $R^+ = C_1 \oplus \cdots \oplus C_r$, where each $|C_i| = p^{e_i}$, such that $e_1 \geq e_2 \geq \cdots \geq e_r$, and $e = \sum_{i=1}^{r} e_i$. The tuple $(e_1, \ldots, e_r)$ characterizes the abelian p-group up to isomorphism.

[4] When we talk of order of an element $a \in R$, we shall mean the order of a as an element of the additive group $(R, +)$. In other words, it is the least positive integer t such that $ta = 0$.

We are interested in describing the endomorphisms of the group R^+ (an endomorphism of R^+ is a group homomorphism from R^+ to R^+). The following theorem [Sho28] shows that each endomorphism of R^+ can be given a matrix representation. To see this we first note that R^+ can be expressed as the direct sum $C_1 \oplus \cdots \oplus C_r$, we can choose an independent generating set for R^+ by picking a generator g_i for each cyclic group C_i in the above direct sum. Thus, the elements of R^+ are of the form $\sum_{i=1}^{r} x_i g_i$, where x_i is an integer modulo p^{e_i} for each i. Hence, R^+ can be identified with the set of integer column vectors $(x_1, x_2, \ldots, x_r)^T$, where x_i is an integer modulo p^{e_i}, and addition of these vectors is done coordinate-wise, where addition in the ith coordinate is modulo p^{e_i}.

Therefore, an endomorphism ψ of R^+ can be described by writing down $\psi(g_i)$ for each i as a linear combination $\sum_{j=1}^{r} h_{ij} g_j$. The $r \times r$ matrix with integral entries h_{ij} will describe an endomorphism. The following theorem [Sho28] characterizes the integral matrices that define endomorphisms of R^+ (The original paper writes $\psi(g_i)$ as a row vector, whereas we write it as a column vector).

Theorem 7. [Sho28, Satz1] *Let A be an abelian p-group of order p^e of type $(e_1, \ldots, e_r)$. I.e. $A = C_1 \oplus \cdots \oplus C_r$ with $|C_i| = p^{e_i}$ for each i. For $1 \leq i, j \leq r$, define integers μ_{ij} as follows: $\mu_{ij} = 1$ for $i \geq j$ and $\mu_{ij} = p^{e_i - e_j}$ for $i < j$.*

Then an $r \times r$ integral matrix $M = (m_{ij})$ describes an endomorphism of A if and only if $m_{ij} = \mu_{ij} h_{ij}$, for some integer h_{ij}, where m_{ij} is an integer computed modulo p^{e_i} for $1 \leq i, j \leq r$.

As explained in [Sho28], the set of integral matrices defined by Theorem 7 forms a ring $\mathrm{End}(A)$ (the endomorphism ring). The addition and multiplication of two matrices in $\mathrm{End}(A)$ is defined as the usual matrix operation where the entries are computed with the modulo operation prescribed by Theorem 7: the ijth entry is computed modulo p^{e_i}. It is easy to verify that $\mathrm{End}(A)$ is a ring under these operations.

Now we show that the ring R can be embedded inside $\mathrm{End}(R^+)$. Thus, R is essentially a subring of $\mathrm{End}(R^+)$, which means that we can view the elements of R as $r \times r$ integral matrices.

To every element $a \in R$, we associate the endomorphism $T_a \in \mathrm{End}(R^+)$ defined as $T_a(x) = ax$ for $x \in R^+$. We claim that T_a defines the zero element of $\mathrm{End}(R^+)$ if and only if $a = 0$. To see this, recall that: R has an element a_0 which is not a zero divisor. Thus, if T_a defines the zero endomorphism, $T_a(a_0) = aa_0 = 0$. Since a_0 is not a zero divisor, we have $a = 0$. As an immediate consequence, we have the following lemma (that R can be seen as a subring of $\mathrm{End}(R^+)$).

Lemma 6. *The homomorphism $\psi : R \longrightarrow \mathrm{End}(R^+)$ defined by $\psi(a) = T_a$, for $a \in R$ is an embedding (i.e. ψ has trivial kernel and is thus 1-1).*

Given R as input by its addition and multiplication tables, we can construct a logspace machine that converts every $a \in R$ into the matrix $T_a \in \mathrm{End}(R^+)$: it follows essentially from the assumption that the decomposition $R^+ = C_1 \oplus \cdots \oplus C_r$ is given as part of the input. Let g_i be a generator for C_i for each i. Thus, we can identify any element $y \in R$ with the corresponding integer vector

$\overline{y} = (x_1, \ldots, x_r)$, where $y = \sum x_i g_i$ and x_i is computed modulo p^{e_i}. Now, given $a \in R$, it is easy to see that the jth column of the matrix T_a is the vector $\overline{ag_j}$. Now, a logspace machine can compute $\overline{y}$ for any given $y \in R$. Thus, a logspace machine can compute T_a, given a.

Therefore, without loss of generality, we can assume that the ring R is already given by $r \times r$ matrices denoting elements of $\mathrm{End}(R^+)$, where the additive abelian group R^+ is given by decomposition $R^+ = C_1 \oplus \cdots \oplus C_r$.

Now, consider the system of linear equations $A\mathbf{x} = \mathbf{b}$ over R, where each entry of A and $\mathbf{b}$ is an $r \times r$ integer matrix, and each entry of the column vector $\mathbf{x}$ is an indeterminate that will take values in R. As we did earlier with matrices in $M_n(\mathbb{F}_q)$, we can convert $A\mathbf{x} = \mathbf{b}$ into a system of linear equations modulo prime powers (the main difference is that different equations may be computed modulo different powers of p):

We replace each variable x_i of $\mathbf{x}$ by the linear combination $\sum_{a \in R} y_{ai} T_a$, where $y_{ai} \in \mathbb{Z}_{p^e}$. This ensures that x_i will take values only in R. Thus, A is now an $mr \times nr$ matrix with integer entries. Now, notice that $\mathbf{b}$ is an $mr \times r$ matrix, where the (i,j)th entry in each $r \times r$ block is evaluated modulo p^{e_i}. Thus, corresponding to each entry of the $mr \times r$ matrix $\mathbf{b}$, if it is the (i,j)th entry of an $r \times r$ block, we get a linear equation modulo p^{e_i}. It will assume the form $\sum_{k=1}^{nr} \alpha_j z_j = \beta \pmod{p^{e_i}}$, where the indeterminates z_j are actually appropriate y_{aj}'s and α_j are from the appropriate entries of A. As $p^{e_i} \leq p^e$, the above linear equation is equivalent to $\sum_{k=1}^{nr} p^{e-e_i} \alpha_j z_j = p^{e-e_i} \beta \pmod{p^e}$.

Thus, we have reduced the feasibility of $A\mathbf{x} = \mathbf{b}$ over R to an instance of LCON (modulo a tiny prime power p^e). We can now derive the following.

Theorem 8. *The problem of testing feasibility of linear equations $A\mathbf{x} = \mathbf{b}$ over a finite R with unity is in $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$, where R is given as input by its addition (denoted by $+$) and multiplication (denoted by concatenation) tables, and the additive abelian group $(R, +)$, denoted R^+ is given as a direct sum $C_1 \oplus \cdots \oplus C_r$, where each C_i is a cyclic group of prime power order.*

4 Concluding Remarks

In [AV04] we had claimed that LCON is in the uniform class $\mathrm{L}^{\mathrm{GapL}}$. This was based on an observation in [ABO99] about computing ranks of matrices over general commutative rings. Subsequently, it was pointed out to us by Allender and McKenzie that the notion of rank over rings(such as $\mathbb{Z}_q$, for composite q) is not well defined. Unlike the case of linear equations over fields, there does not seem to be a notion of rank for rings that can be used to test feasibility of linear equations over rings. In this paper we find a different approach to the problem, but succeed in proving only the weaker upper bound of $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$.

It is remarked in [ABO99], based on the results of [Gi95], that solving linear diophatine equations is randomized NC reducible to computing the gcd of a list of integers. With this as a starting point, we have explored the problem of feasibility of linear equations modulo composites. We also consider the feasibility

of linear equations over arbitrary rings with unity. Surprisingly, it turns out that, by giving a suitable matrix representation to elements of the arbitrary ring, we can reduce this problem to solving linear equations modulo prime powers.

Specifically, we have show in this paper that the problem LCON of testing the feasibility of linear equations modulo composites q (with tiny prime power factors) is in the class $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$. Indeed, under a hardness assumption, it is in $\mathrm{L}^{\mathrm{GapL}}$. Using the approach based on the isolation lemma, as explained in our previous paper [AV04], we can easily show that finding a solution to an instance of LCON is in the function class $\mathrm{FL}^{\mathrm{GapL}}/\mathrm{poly}$ (which can also be derandomized under the same hardness assumption as used in Theorem 6). As we show in Section 3, it turns out that over arbitrary (even noncommutative) rings with unity, the same upper bound holds for the feasibility problem.

We leave open the question if the upper bounds can be improved to $\mathrm{L}^{\mathrm{GapL}}$ without the hardness assumption.

Acknowledgment. For discussions and comments on this work, and for bringing to our notice the problems with using a notion of rank to test the feasibility of linear equations modulo composites, we are very grateful to Eric Allender and Pierre McKenzie. We also thank the referees for useful remarks.

References

[AV04] V. Arvind and T.C. Vijayaraghavan. Abelian Permutation Group Problems and Logspace Counting Classes. *IEEE Annual Conference on Computational Complexity (CCC'04)*, pp.204–214, 2004.

[ABO99] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8:99–126, 1999.

[ARZ99] E. Allender, K. Reinhardt, and S. Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

[BDHM92] G. Buntrock, C. Damm, U. Hertrampf, C. Meinel. Structure and Importance of Logspace-MOD Classes. *Mathematical Systems Theory*, 25(3):223-237, 1992.

[Dic92] L.E. Dickson. *History of the theory of numbers, vol II: Diophantine analysis.* Chelsea Publishing Company, 1992.

[Gi95] M. Giesbrecht. Fast computations of the Smith Normal Form of an integer matrix. *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pp.110–118, 1995.

[Gi97] M. Giesbrecht. Efficient parallel solution of sparse systems of linear diophantine equations. *Proceedings of the second international symposium on Parallel Symbolic Computation*, pp.1–10, 1997.

[KM99] A. Klivans and D. van Melkebeek. Graph Isomorphism has subexponential size provers unless the polynomial time hierarchy collapses. *SIAM Journal of Computing*, 31(5):1501-1526, 2002.

[MC87] P. McKenzie and S. Cook. The parallel complexity of abelian permutation group problems. *Siam Journal on Computing*, 16:880–909, 1987.

[Sch98] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley, 1998.

[Sho28] K. Shoda. Über die Automorphismen einer endlichen Abelschen Gruppe. *Mathematische Annalen*, 100:674–686, 1928. Source: The Göttingen State and University Library (http://www.sub.uni-goettingen.de).

[Tod91] S. Toda. Counting problems computationally equivalent to computing the determinant. *Technical Report CSIM 91-07, Department of Computer Science, University of Electro-Communications, Tokyo, Japan*, May 1991.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Structure in Complexity Theory Conference*, pp.270–284, 1991.

A Lower Bound on the Complexity of Polynomial Multiplication over Finite Fields

(Extended Abstract)

Michael Kaminski*

Department of Computer Science,
Technion – Israel Institute of Technology, Haifa 32000, Israel
kaminski@cs.technion.ac.il

Abstract. It is shown that computing the coefficients of the product of two degree-n polynomials over a q-element field by means of a quadratic algorithm requires at least $(3 + \frac{(q-1)^2}{q^5+(q-1)^3})n - o(n)$ multiplications, whereas the best lower bound known from the literature is $3n - o(n)$.

1 Introduction

In infinite fields it is possible to compute the coefficients of the product of two polynomials of degree n in $2n+1$ non-scalar multiplications. It is known from [14] that each algorithm for computing the product in $2n+1$ non-scalar multiplications must evaluate the multiplicands at $2n+1$ distinct points (possibly including ∞), multiply the samples, and interpolate the result. However in finite fields this method fails, if $2n$ exceeds the number of the field elements. Thus, in general, the $2n+1$ tight bound cannot be achieved in finite fields.

Let $\mu_F(n)$ denote the number of multiplications required to compute the coefficients of the product of a polynomial of degree n and a polynomial of degree $n-1$ over field F by means of quadratic algorithms[1] and let F_q denote the q-element field. For $q \geq 3$, the best lower bound on $\mu_{F_q}(n)$ known from the literature is $3n - o(n)$, see [8] and [5–Theorem 18.10] and, for sufficiently large n, $\mu_{F_2}(n) > 3.52n$, see [3]. On the other hand, if $q \geq 3$, computing the coefficients of the minimal degree residue of the product of two polynomials of degree n modulo a fixed irreducible polynomial of degree $n+1$ over F_{q^2} can be done in $2(1 + \frac{1}{q-3})n + o(n)$ multiplications, see [6] and [12]. Thus, for $q \geq 3$, $\mu_{F_{q^2}}(n) \leq 4(1 + \frac{1}{q-3})n + o(n)$. This small difference between the upper and the lower bounds motivates a further search for better (both upper and lower)

* Supported by the Technion V.P.R. fund.

[1] A straightforward substitution argument shows that the number of multiplications required to compute the coefficient of the product of two polynomials of degree n over F exceeds $\mu_F(n)$ by at least 1.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 485–495, 2005.

bounds on the complexity of polynomial multiplications. Also, apart from being of interest in their own right, algorithms for polynomial multiplication over finite fields are tightly connected to error-correcting codes, see [2, 3, 6, 7, 9, 10].

In our paper we prove the following lower bound on $\mu_{F_q}(n)$.

Theorem 1. *We have*

$$\mu_{F_q}(n) \geq (3 + \frac{(q-1)^2}{q^5 + (q-1)^3})n - o(n).$$

The proof of Theorem 1 is based on a novel combination of two known techniques. One technique is the analysis of *Hankel* matrices representing bilinear forms defined by linear combinations of the coefficients of the polynomial product, see [8]. The other technique is a counting argument from the coding theory, see [10].

The reason for combining these two techniques is that the Hankel matrix approach uses very few properties of finite fields and the coding approach does not at all use a very special structure of bilinear forms defined by linear combinations of the coefficients of the polynomial product. In fact, our paper indicates that the Hankel matrix approach is, in some sense, richer than Baur's technique, see [5–Proof of Theorem 18.10].

We end this section with the note that if a set of quadratic forms over an infinite field can be computed by a (general) *straight-line algorithm* in t multiplications/divisions, then it can be computed in t multiplications by a quadratic algorithm whose total number of operations differs from that of the original one by a factor of a small constant, see [13]. It is unknown whether a similar result holds for finite fields, cf. [4]. However, the proof in [13] easily extends to finite fields, if for some input that belongs to the underlying finite field the original straight-line algorithm does not divide by zero. Therefore, in the case of multiplication of polynomials over F_q, our lower bound applies to all straight-line algorithm which compute the coefficients of the product of at least one pair of polynomials whose all coefficients belong to F_q. Finally, one can easily prove that quadratic algorithms for computing a set of bilinear forms are optimal within the class of algorithms without divisions, and all algorithms for polynomial multiplication over finite fields known from the literature are quadratic (and even *bilinear*).

This extended abstract is organized as follows. In the next section we gather the definitions and some basic results known from the literature. Then, in Section 3, we indicate some limitations of the known tools and present an example on which we develop the technique used for the proof of Theorem 1. The proof itself is sketched in Section 4.

2 Notation, Definitions, and Auxiliary Results

First we define the notion of a quadratic algorithm. Then we introduce notation and definitions from linear recurring sequence theory and state the major

auxiliary technical lemmas. We conclude this section with some some basic calculations from the coding theory and an example that will be used for the proof of Theorem 1.

2.1 Quadratic Algorithms for Polynomial Multiplication

Let $\boldsymbol{s}$ be a set of indeterminates. We remind the reader that a *quadratic* algorithm over field F for computing a set $\boldsymbol{Q}$ of quadratic forms of the elements of $\boldsymbol{s}$ is a straight-line algorithm whose non-scalar multiplications are of the form $L' * L''$, where L' and L'' are linear forms of the elements of $\boldsymbol{s}$ over F and each form in $\boldsymbol{Q}$ is a linear combination of these products. The minimal number of such multiplications is called the *quadratic complexity of* $\boldsymbol{Q}$ *over* F and is denoted by $\mu_F(\boldsymbol{Q})$.

Let $\boldsymbol{Q} = \{\boldsymbol{x}^T H \boldsymbol{y} : H \in \boldsymbol{S}\}$, where $\boldsymbol{S}$ is a set of matrices. That is the quadratic (in fact, *bilinear*) forms in $\boldsymbol{Q}$ are defined by the elements of $\boldsymbol{S}$.

In what follows we identify $\boldsymbol{Q}$ *with* $\boldsymbol{S}$ *and often write* $\mu_F(\boldsymbol{S})$ *instead of* $\mu_F(\boldsymbol{Q})$. *Also, we identify a quadratic algorithm with the corresponding set of pairs of linear forms.*

Now, let $\boldsymbol{x} = (x_0, x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$. We have to compute $z_k = \sum_{i+j=k} x_i y_j$, $k = 0, \ldots, 2n-1$. Let $\boldsymbol{z} = (z_0, z_1, \ldots, z_{2n-1})$. Assume that $\mu_F(n) = t$. That is, there exist t linear forms $L'_1, \ldots, L'_t$ and t linear forms $L''_1, \ldots, L''_t$ such that each z_k is a linear combination of products $L'_1 L''_1, \ldots, L'_t L''_t$. Let $\boldsymbol{p} = (L'_1 L''_1, \ldots, L'_t L''_t)$. Then there exists a $2n \times t$ matrix U such that $\boldsymbol{z}^T = U \boldsymbol{p}^T$. Since the components of $\boldsymbol{z}$ are linearly independent, $\mathrm{rank}(U) = 2n$. Therefore, permuting the components of $\boldsymbol{p}$, we may assume that the first $2n$ columns of U are linearly independent. Hence there exist a $2n \times 2n$ matrix W and a $2n \times (t-2n)$ matrix V such that

$$W \boldsymbol{z}^T = (\boldsymbol{I}_{2n}, V) \boldsymbol{p}^T, \tag{1}$$

where $\boldsymbol{I}_{2n}$ denotes the $2n \times 2n$ identity matrix.

Let $W \boldsymbol{z}^T = (\boldsymbol{x} D_1 \boldsymbol{y}^T, \boldsymbol{x} D_2 \boldsymbol{y}^T, \ldots, \boldsymbol{x} D_{2n} \boldsymbol{y}^T)^T$ and $\boldsymbol{D} = \{D_1, D_2, \ldots, D_{2n}\}$. Proposition 1 below immediately follows from the above notation.

Proposition 1. (Cf. [1–Proposition 1].) *Let* $\boldsymbol{D}' \subseteq \boldsymbol{D}$. *Then*

$$\mu_F(n) (= \mu_F(\boldsymbol{D})) \geq 2n + \mu_F(\boldsymbol{D}') - |\boldsymbol{D}'|.^2$$

We apply Proposition 1 for the proof of Theorem 1 in a standard manner, see [1],[5–Proof of Theorem 18.10],[4], and [8]. Namely, we prove that there exists a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality $o(n)$ such that $\mu_{F_q}(\boldsymbol{D}') \geq (1 + \frac{(q-1)^2}{q^5 + (q-1)^3})n - o(n)$, cf. [5–Proof of Theorem 18.10] and [8]. For this purpose we refine the technique developed in [8].

[2] For a set X, we denote by $|X|$ the number of elements of X.

2.2 Linear Recurring Sequences and Hankel Matrices

Let k be a positive integer and let $a_0, \ldots, a_{k-1}$ belong to a field F. A sequence $\sigma = s_0, s_1, \ldots, s_\ell$ of elements of F satisfying the relation

$$s_{m+k} = a_{k-1}s_{m+k-1} + a_{k-2}s_{m+k-2} + \cdots + a_0 s_m, \quad m = 0, 1, \ldots, \ell - k$$

is called a (finite k-th-order homogeneous) *linear recurring sequence* in F. The terms $s_0, s_1, \ldots, s_{k-1}$ are called *initial values*. The polynomial

$$f(\alpha) = \alpha^k - a_{k-1}\alpha^{k-1} - a_{k-2}\alpha^{k-2} - \cdots - a_0 \in F[\alpha]$$

is called a *characteristic polynomial* of σ.

Proposition 2 below shows that "sufficiently long" finite linear recurring sequences behave like the infinite ones.

Proposition 2. ([8–Proposition 1]) *Let σ and $f(\alpha)$ be as above, and let $f_\sigma(\alpha)$ be a characteristic polynomial of σ of the minimal degree. If* $\deg f_\sigma(\alpha) + \deg f(\alpha) \leq \ell + 1$, *then $f_\sigma(\alpha)$ divides $f(\alpha)$.*

A uniquely determined monic polynomial $f_\sigma(\alpha) \in F[\alpha]$ given by Proposition 2 is called the *minimal polynomial* of σ.

For a sequence $\sigma = s_0, s_1, \ldots, s_{2n-1}$ we define the $(n+1) \times n$ *Hankel* matrix $H(\sigma)$ by

$$H(\sigma) = \begin{pmatrix} s_0 & s_1 & \cdots & s_{n-1} \\ s_1 & s_2 & \cdots & s_n \\ \vdots & \vdots & & \vdots \\ s_n & s_{n+1} & \cdots & s_{2n-1} \end{pmatrix}.$$

The proof of Theorem 1 is based on the fact that linear combinations of the coefficients of the polynomial product are defined by Hankel matrices and vice versa.

Let $H^i(\sigma)$ denote the $(i+1)$th row of $H(\sigma)$ and let k be the minimal integer for which there exist $a_0, a_1, \ldots, a_{k-1} \in F$ such that

$$\sum_{i=0}^{k-1} a_i H^i(\sigma) = H^k(\sigma).$$

We define sequence $\tilde{\sigma} = \tilde{s}_0, \tilde{s}_1, \ldots, \tilde{s}_{2n-1}$ by the linear recurrence

$$\tilde{s}_{i+k} = a_{k-1}\tilde{s}_{i+k-1} + a_{k-2}\tilde{s}_{i+k-2} + \cdots + a_0 \tilde{s}_i,$$

with initial values $\tilde{s}_i = s_i$, $i = 0, 1, \ldots, k-1$.

Let $f_{\tilde{H}(\sigma)}(\alpha) = \alpha^k - \sum_{i=0}^{k-1} a_i \alpha^i$. Then $f_{\tilde{H}(\sigma)}(\alpha)$ is the *minimal* polynomial of $\tilde{\sigma}$, see [8–Section 3]. We define the (integral) *divisor* $\boldsymbol{f}_{H(\sigma)}(\alpha)$ by $\boldsymbol{f}_{H(\sigma)}(\alpha) = f_{\tilde{H}(\sigma)}(\alpha)(\alpha - \infty)^{\mathrm{rank}(H(\sigma - \tilde{\sigma}))}$.

For Lemmas 1 and 2 below we need the following notation.

For a set $\boldsymbol{S}$ of $(n+1)\times n$ Hankel matrices we denote $\mathrm{lcm}\{\boldsymbol{f}_H(\alpha) : H \in \boldsymbol{S}\}$ by $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$, where, as usual, lcm is an abbreviation for "the least common multiple."

Next, for a vector space V and a subset V' of V, we denote by $[V']$ the linear subspace of V spanned by the elements of V'.

Finally, we denote the maximal possible number of distinct factors of a polynomial of degree n over F_q by $i_q(n)$. It well-known that for $q \geq 3$, $i_q(n) < \frac{n}{\lg_q n - 3}$, see [8–Appendix 1].

Lemma 1. ([8–Lemma 2] and [4–Lemma 1]) *Let $\boldsymbol{S}$ be a set of $(n+1)\times n$ Hankel matrices over a field F such that $\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \geq n+1$. Then $\mu_F(\boldsymbol{S}) \geq n+1$.*

Lemma 2. ([8–Lemma 4]) *Let $\boldsymbol{S}$ be a set of $(n+1)\times n$ Hankel matrices over F_q. Then for each $m \leq \dim([\boldsymbol{S}])$ there exists a subset $\boldsymbol{S}'$ of $\boldsymbol{S}$ such that $|\boldsymbol{S}'| \leq i_q(m)$ and $\deg \boldsymbol{f}_{\boldsymbol{S}'}(\alpha) \geq m$.*

Example 1. ([8–Theorem 1].) Let $\boldsymbol{D}$ be as in the end of the previous section. Then $\dim(\boldsymbol{D}) = 2n$ and, by Lemma 2, there is a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality not exceeding $i_q(n+1)$ such that $\deg \boldsymbol{f}_{\boldsymbol{D}'}(\alpha) \geq n+1$. By Lemma 1, $\mu_{F_q}(\boldsymbol{D}') \geq n+1$. Thus, by Proposition 1, $\mu_{F_q}(n) = 3n - o(n)$.

2.3 Bounds from the Coding Theory

The most basic notion of the coding theory is the *weight* of a vector $\boldsymbol{v}$, denoted $\boldsymbol{wt}(\boldsymbol{v})$, that is the number of non-zero components of $\boldsymbol{v}$. We need the following corollaries to [11–Problem 3.6, p. 73].

Proposition 3. *Let U be a $2 \times t$ matrix over F_q without zero columns. Then*

$$t = \frac{\boldsymbol{wt}((0,1)U) + \sum_{v \in F_q} \boldsymbol{wt}((1,v)U)}{q}.$$

Proposition 4. (Cf. [10–Proof of Theorem 1].) *Let $\boldsymbol{S}$ be a set of k linearly independent $(n+1)\times n$ Hankel matrices over F_q. Then*

$$\mu_{F_q}(\boldsymbol{S}) \geq \frac{\sum_{H \in [\boldsymbol{S}]} \deg \boldsymbol{f}_H(\alpha)}{q^k - q^{k-1}}.$$

Example 2. Let $\boldsymbol{D}$ be as in the end of Section 2.1 and let

$$\epsilon_q = \frac{q^4}{q^5 + (q-1)^3}. \tag{2}$$

If there exists a subset $\boldsymbol{D}'$ *of* $\boldsymbol{D}$ *such that* $|\boldsymbol{D}'| \leq \lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$ *and for each* $H \in [\boldsymbol{D}']$, $\deg \boldsymbol{f}_H(\alpha) \geq (1-\epsilon_q)n$, *then*

$$\mu_{F_q}(n) \geq (3 + \frac{(q-1)^2}{q^5 + (q-1)^3})n - o(n).$$

That is, Theorem 1 is true in this degenerate case. Indeed, by Proposition 4,

$$\begin{aligned}
\mu_{F_q}(\boldsymbol{D}') &\geq \frac{\sum\limits_{H \in [\boldsymbol{D}']} \deg \boldsymbol{f}_H(\alpha)}{q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil} - q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil - 1}} \\
&\geq \frac{(q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil} - 1)(1-\epsilon_q)n}{q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil} - q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil - 1}} \\
&\geq (1 + \frac{1}{q-1})(1-\epsilon_q)n - \frac{(1-\epsilon_q)n \lg_q \lg_q n}{\lg_q n}
\end{aligned}$$

and the desired inequality follows from (2) and Proposition 1.

3 Limitations of Lemma 1 and Its Extension

The proof of Example 1 does not use much of the finiteness of the underlying field, because Lemma 1 holds for any field. In addition the proof uses only "a half" of $\dim([\boldsymbol{D}])(= 2n)$. Therefore, a better bound might be achieved, if we succeed to extend Lemma 1 "beyond $n+1$" (just in the case of a finite field), which would allow us to use a "bigger portion of $\dim([\boldsymbol{D}])$." Indeed, as shows Theorem 2 in this section, Lemma 1 can be extended and in Section 4 we apply its extension for the proof of Theorem 1.

We illustrate Theorem 2 by Example 3 below that is based on the following lemma.

Lemma 3. *Let* H' *and* H'' *be* $(n+1) \times n$ *Hankel matrices over* F_q *and let* $\alpha \in F_{q^2} \setminus F_q$. *Then each quadratic algorithm over* F_q *that computes* $\boldsymbol{x}(H' + \alpha H'')\boldsymbol{y}^T$ *also computes* $\{\boldsymbol{x}H'\boldsymbol{y}^T, \boldsymbol{x}H''\boldsymbol{y}^T\}$.

Example 3. Let n be odd and let M, M', and M'' be $(n+1) \times n$ Hankel matrices over F_q such that $\boldsymbol{f}_M(\alpha)$, $\boldsymbol{f}_{M'}(\alpha)$, and $\boldsymbol{f}_{M''}(\alpha)$ are pairwise coprime polynomials of degree $\frac{n+1}{2}$. We shall prove that $\mu_{F_q}(\{M, M', M''\}) \geq (1 + \frac{1}{2q^2})(n+1)$, whereas Lemma 1 gives us $\mu_{F_q}(\{M, M', M''\}) \geq n+1$, only.

Let $\{(L_i', L_i'')\}_{i=1,2,\ldots,t}$ be a quadratic algorithm over F_q that computes the bilinear forms defined by those three matrices and let $\boldsymbol{p} = (L_1'L_1'', \ldots, L_t'L_t'')$. Then this algorithm also computes the bilinear forms over F_{q^2} defined by $M + \alpha M'$ and M'', where $\alpha \in F_{q^2} \setminus F_q$ is as in Lemma 3. That is, there exists a $2 \times t$ matrix U over F_{q^2} such that

$$\begin{pmatrix} \boldsymbol{x}(M + \alpha M')\boldsymbol{y}^T \\ \boldsymbol{x}M''\boldsymbol{y}^T \end{pmatrix} = U\boldsymbol{p}^T.$$

Therefore, $\boldsymbol{x}M''\boldsymbol{y}^T = (0,1)U\boldsymbol{p}^T$, implying

$$\boldsymbol{wt}((0,1)U) \geq \mu_{F_q}(\{M''\}) \tag{3}$$

and for $u, v \in F_q$,

$$\boldsymbol{x}((M + uM'') + \alpha(M' + vM''))\boldsymbol{y}^T = (1, u + \alpha v)U\boldsymbol{p}^T,$$

implying, by Lemma 3,

$$\boldsymbol{wt}((1, u + \alpha v)U) \geq \mu_{F_q}(\{M + uM'', M' + vM''\}). \tag{4}$$

Now, combining (3) and (4) with Proposition 3 we obtain

$$t \geq \frac{\mu_{F_q}(\{M''\}) + \sum_{u,v \in F_q} \mu_{F_q}(\{M + uM'', M' + vM''\})}{q^2}. \tag{5}$$

Since $\mu_{F_q}(\{M''\}) = \deg \boldsymbol{f}_{M''}(\alpha) = \frac{n+1}{2}$ and, by Lemma 1, $\mu_{F_q}(\{M + uM'', M' + vM''\}) \geq n + 1$, the desired inequality $t \geq (1 + \frac{1}{2q^2})(n + 1)$ follows from (5).

Theorem 2 below generalizes Example 3 by extending Lemma 1 "beyond $n + 1$." To state the theorem we need the following notation.

Let $\boldsymbol{S}$ be a set of $(n + 1) \times n$ Hankel matrices. We denote $\min\{\deg \boldsymbol{f}_H(\alpha) \neq 0 : H \in [\boldsymbol{S}]\}$ and $\min\{\deg \boldsymbol{f}_{\boldsymbol{S}'}(\alpha) : [\boldsymbol{S}'] = [\boldsymbol{S}]\}$ by $\|\boldsymbol{S}\|$ and $\deg \boldsymbol{S}$,[3] respectively.

Theorem 2. *Let $\boldsymbol{M}$ be a set of three linearly independent $(n + 1) \times n$ Hankel matrices over F_q such that $\deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) > n$. Then*

$$\mu_{F_q}(\boldsymbol{M}) \geq n + 1 + \frac{\min(\deg \boldsymbol{M} - n - 1, \|\boldsymbol{M}\|)}{q^2}.$$

Thus, we shall search for a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality $o(n)$ such that $[\boldsymbol{D}']$ includes a set $\boldsymbol{M}$ of three linearly independent $(n+1) \times n$ Hankel matrices satisfying

$$\min(\deg \boldsymbol{M} - n - 1, \|\boldsymbol{M}\|) \geq \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{6}$$

[3] Note the difference between $\deg \boldsymbol{S}$ and $\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha)$. Whereas the latter refers to a particular set of Hankel matrices $\boldsymbol{S}$, the former is the minimum of $\deg \boldsymbol{f}_{\boldsymbol{S}'}(\alpha)$. over all sets $\boldsymbol{S}'$ which span $[\boldsymbol{S}]$. Nevertheless, they are equal, if either of them is less than $n + 1$.

4 Sketch of the Proof of Theorem 1

Let ϵ_q be as in (2) and let $\boldsymbol{D} = \{D_1, D_2, \ldots, D_{2n}\}$ be the set of Hankel matrices defining the components of $W\boldsymbol{z}^T$ in (1). In view of Example 2, we may assume that the linear closure of each subset of $\boldsymbol{D}$ of cardinality $\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$ contains an element H such that $\deg \boldsymbol{f}_H(\alpha) < (1-\epsilon_q)n$. Under this assumption we shall construct a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality at most $\frac{(1+\epsilon_q)n \lg_q \lg_q n}{\lg_q((1+\epsilon_q)n)}$ whose linear closure includes a subset $\boldsymbol{M} = \{M, M', M''\}$ that satisfies (6), which together with Theorem 2 and Proposition 1 (and Example 2, of course) will prove Theorem 1.

Lemma 4. *Assume that the linear closure of each subset of $\boldsymbol{D}$ of cardinality $\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$ contains a matrix H such that $\deg \boldsymbol{f}_H(\alpha) < (1-\epsilon_q)n$. Then there exists a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality at most $\frac{(1+\epsilon_q)n \lg_q \lg_q n}{\lg_q((1+\epsilon_q)n)}$ and a subset $\boldsymbol{H}$ of $[\boldsymbol{D}']$ such that $\deg \boldsymbol{H} \geq (1+\epsilon_q)n$ and for each $H \in \boldsymbol{H}$, $\deg \boldsymbol{f}_H(\alpha) < (1-\epsilon_q)n$.*

A set $\boldsymbol{M} = \{M, M', M''\}$ that satisfies (6) is, actually, chosen from $[\boldsymbol{H}]$ as follows. First we construct Hankel matrices M and M', $M, M' \in [\boldsymbol{H}]$, such that

$$\deg \boldsymbol{f}_M(\alpha), \deg \boldsymbol{f}_{M'}(\alpha) \leq (1 - \frac{(q-1)^2}{q^2}\epsilon_q)n, \tag{7}$$

$$\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \geq (1 - \frac{(q-1)^2}{q^2}\epsilon_q)n, \tag{8}$$

and

$$\|\{M, M'\}\| \geq \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{9}$$

Then we complete $\{M, M'\}$ to $\boldsymbol{M}$. For this we need the following technical notation.

Let $\boldsymbol{H} = \{H_1, H_2, \ldots, H_k\}$ and let $\boldsymbol{f}_{\boldsymbol{H}}(\alpha) = \prod_{i=1}^{\ell} p_i^{d_i}(\alpha)$ be the decomposition of $\boldsymbol{f}_{\boldsymbol{H}}(\alpha)$ into irreducible factors. Then for each $H \in \boldsymbol{H}$ and each $i = 1, 2, \ldots, \ell$ there is the unique matrix $H|_i$ – the $p_i(\alpha)$-component of H – such that $\boldsymbol{f}_{H|_i}$ divides $p_i^{d_i}(\alpha)$ and $H = \sum_{i=1}^{\ell} H|_i$.

The set $\{H|_i : H \in \boldsymbol{H}\}$ of all $p_i(\alpha)$-components of the elements of $\boldsymbol{H}$ will be denoted by $\boldsymbol{H}|_i$.

Also, we denote by $\boldsymbol{H}$ and $\boldsymbol{H}|_i$, $i = 1, 2, \ldots, \ell$, the column vectors of the elements of $\boldsymbol{H}$ and $\boldsymbol{H}|_i$, respectively. That is, $\boldsymbol{H} = (H_1, H_2, \ldots, H_k)^T$ and $\boldsymbol{H}|_i = (H_1|_i, H_2|_i, \ldots, H_k|_i)^T$, say.

Next, for a vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_k) \in F_q^k$ we denote by $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha)$ the divisor $\boldsymbol{f}_{\{\boldsymbol{v}\boldsymbol{H}|_i : i=1,2,\ldots,\ell\}}(\alpha)$. That is, $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha)$ is the product of the divisors

associated with the summands $\boldsymbol{vH}|_i$, $i = 1, 2, \ldots, \ell$, of $\boldsymbol{vH} = \sum_{i=1}^{k} \boldsymbol{vH}|_i$. In particular, if $\boldsymbol{f}_{H_i}(\alpha)$ is the minimal polynomial of an infinite linear recurring sequence σ_i, $i = 1, 2, \ldots, \ell$, then $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha)$ is the minimal polynomial of $\sum_{i=1}^{k} v_i \sigma_i$.

Remark 1. Note the difference between $\boldsymbol{f}_{\boldsymbol{vH}}(\alpha)$ and $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha)$. The degree of the former is at most n, whereas the degree of the latter can exceed it. The divisors are equal if and only if $\deg \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha) \le n$.

The following proposition is an easy corollary to the definition of $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha)$ and Remark 1.

Proposition 5. *Let* $\boldsymbol{v}', \boldsymbol{v}'' \in F_q^k$. *Then*

$$\deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}'\mathcal{C}(\boldsymbol{H})}(\alpha), \boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\boldsymbol{H})}(\alpha)\} \ge \deg \boldsymbol{f}_{\boldsymbol{v}'+\boldsymbol{v}''\mathcal{C}(\boldsymbol{H})}(\alpha).$$

The proof of Proposition 6 below is based on a counting argument from the coding theory.

Proposition 6. *Let* $\boldsymbol{H} = \{H_1, H_2, \ldots, H_k\}$. *Then, for each divisor* $\boldsymbol{f}(\alpha)$,

$$\sum_{\boldsymbol{v} \in F_q^k} (\deg \operatorname{lcm}\{\boldsymbol{f}(\alpha), \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha)\} - \deg \boldsymbol{f}(\alpha)) \ge (q^k - q^{k-1})(\deg \boldsymbol{f}_{\boldsymbol{H}}(\alpha) - \deg \boldsymbol{f}(\alpha)).$$

Example 4. Let $\boldsymbol{H}$ be as above. Assume that for all $\boldsymbol{v} \in F_q^k$, $\deg \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha) \le n$. Then

$$\mu_{F_q}(\boldsymbol{H}) \ge \frac{\sum_{\boldsymbol{v} \in F_q^k} \deg \boldsymbol{f}_{\boldsymbol{vH}}(\alpha)}{q^k - q^{k-1}} = \frac{\sum_{\boldsymbol{v} \in F_q^k} \deg \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha)}{q^k - q^{k-1}} \ge \deg \boldsymbol{f}_{\boldsymbol{H}}(\alpha),$$

where the first inequality is by Proposition 4 (because $[\boldsymbol{H}] = \{\boldsymbol{vH} : \boldsymbol{v} \in F_q^k\}$), the equality is by Remark 1, and the last inequality follows from Proposition 6 with $\boldsymbol{f}(\alpha) = 1$.

Since $\deg \boldsymbol{f}_{\boldsymbol{H}}(\alpha) \ge (1 + \epsilon_q)n$, by Example 4, we may assume that there exists a vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_k) \in F_q^k$ such that $\deg \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha) \ge n$. We also assume that $\boldsymbol{v}$ is the minimum weight vector such that $\deg \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\boldsymbol{H})}(\alpha) \ge n$ and fix it till the end of Section 4.1.

Roughly speaking, M and M' are appropriate subsums of $\boldsymbol{vH}$ and M'' is defined by the quotient vector space $[H]/[\{M, M'\}]$. We shall distinguish between the cases of

$$\min\{\deg \boldsymbol{f}_H(\alpha) : H \in \boldsymbol{H}\} < \frac{(q-1)^2}{q^2} \epsilon_q n \tag{10}$$

and

$$\min\{\deg \boldsymbol{f}_H(\alpha) : H \in \boldsymbol{H}\} \geq \frac{(q-1)^2}{q^2}\epsilon_q n$$

and use the following notation.

For $I \subseteq \{1, 2, \ldots, k\}$, we define a k-dimensional vector $\boldsymbol{v}^I = (v_1^I, v_2^I, \ldots, v_k^I)$ by

$$v_i^I = \begin{cases} v_i, \text{ if } \ i \in I \\ 0, \ \text{otherwise} \end{cases} \qquad i = 1, 2, \ldots, k.$$

4.1 The Case of $\min\{\deg \boldsymbol{f}_H(\alpha) : H \in \boldsymbol{H}\} < \frac{(q-1)^2}{q^2}\epsilon_q n$

Let I be a maximal (with respect to inclusion) subset $\{1, 2, \ldots, k\}$ such that

$$\deg \boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\boldsymbol{H})}(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n$$

and let I' be a minimal (with respect to inclusion) subset of $\{1, 2, \ldots, k\} \setminus I$ such that

$$\deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\boldsymbol{H})}(\alpha), \boldsymbol{f}_{\boldsymbol{v}^{I'} \mathcal{C}(\boldsymbol{H})}(\alpha)\} \geq (1 - \frac{(q-1)^2}{q^2}\epsilon_q)n. \tag{11}$$

The existence of I' follows from (10) and Proposition 5.

We put M and M' be $\boldsymbol{v}^I \boldsymbol{H}$ and $\boldsymbol{v}^{I'} \boldsymbol{H}$, respectively. Then (7) – (9) follow from (10),(11), Proposition 5, and Remark 1. In addition, by (10),(11), and Remark 1,

$$\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) < n. \tag{12}$$

Now we construct M''. Proposition 6 implies that for some $\boldsymbol{v}'' \in F_q^k$,

$$\deg \operatorname{lcm}\{\boldsymbol{f}_{\{M,M'\}}(\alpha), \boldsymbol{f}_{\boldsymbol{v}'' \mathcal{C}(\boldsymbol{H})}(\alpha)\} - \deg \boldsymbol{f}_{\{M,M'\}}(\alpha)$$
$$\geq (1 - \frac{1}{q})(\deg \boldsymbol{f}_{\boldsymbol{H}}(\alpha) - \deg \boldsymbol{f}_{\{M,M'\}}(\alpha))$$

and we put M'' be $\boldsymbol{v}'' \boldsymbol{H}$.

Using the properties of $\boldsymbol{H}$ provided by Lemma 4, we can prove that

$$\deg\{M, M', M''\} \geq (1 + \frac{(q-1)^2}{q^2}\epsilon_q)n \tag{13}$$

and it remains to show that

$$\|\{M, M', M''\}\| \geq \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{14}$$

For the proof, assume to the contrary that for some matrix $H \in [\{M, M', M''\}]$, $\deg \boldsymbol{f}_H(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n$. By (9), $H \notin [\{M, M'\}]$, which implies $[\{M, M', H\}] = [\{M, M', M''\}]$. Since

$$\deg \boldsymbol{f}_{\{M,M',H\}}(\alpha) \leq \deg \boldsymbol{f}_{\{M,M'\}}(\alpha) + \deg \boldsymbol{f}_H(\alpha) < (1 + \frac{(q-1)^2}{q^2}\epsilon_q)n,$$

where the right inequality follows from (12), our assumption contradicts (13).

4.2 The Case of $\min\{\deg f_H(\alpha) : H \in \boldsymbol{H}\} \geq \frac{(q-1)^2}{q^2}\epsilon_q n$

The construction of $\boldsymbol{M}$ is similar to that in the previous case using appropriate maximal subsets of $\{1, 2, \ldots, k\}$ and the above vector $\boldsymbol{v}''$. The only difference is that (9) and (12) do not necessarily hold and, consequently, we cannot guarantee (14) either.

To overcome this obstacle we start with a basis $\{H_1, H_2, \ldots, H_k\}$ of $[\boldsymbol{H}]$ satisfying $\deg \boldsymbol{f}_{H_i}(\alpha) \leq (1-\epsilon_q)n$ and containing the maximal number of elements H such that

$$\deg \boldsymbol{f}_H(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{15}$$

Then, if (14) does not hold, we can find a basis of $\boldsymbol{H}$ with more elements H satisfying (15), in contradiction with the above maximality property of the basis $\{H_1, H_2, \ldots, H_k\}$.

References

1. Averbuch, A., Bshouty, N., Kaminski, M.: A classification of quadratic algorithms for multiplying polynomials of small degree over finite fields. Journal of Algorithms **13** (1992) 577–588
2. Brockett, R., Dobkin, D.: On the optimal evaluation of a set of bilinear forms. Linear Algebra and Its Applications **19** (1978) 207–235
3. Brown, M., Dobkin, D.: An improved lower bound on polynomial multiplication. IEEE Transactions on Computers **29** (1980) 337–340
4. Bshouty, N., Kaminski, M.: Multiplication of polynomials over finite fields. SIAM Journal on Computing **19** (1990) 452–456
5. Bürgisser, P., Clausen, M., Shokrollahi, A.: Algebraic complexity theory. Springer, Berlin (1997)
6. Chudnovsky, D., Chudnovsky, G.: Algebraic complexities and algebraic curves over finite fields. Journal of Complexity **4** (1988) 285–316
7. Kaminski, M.: A lower bound for polynomial multiplication. Theoretical Computer Science **40** (1985) 319–322
8. Kaminski, M., Bshouty, N.: Multiplicative complexity of polynomial multiplication over finite fields. Journal of the ACM **36** (1989) 150–170
9. Lempel, A., Winograd, S.: A new approach to error-correcting codes. IEEE Transactions on Information Theory **23** (1977) 503–508
10. Lempel, A., Seroussi, G., Winograd, S.: On the complexity of multiplication in finite fields. Theoretical Computer Science **22** (1983) 285–296 184–202
11. Peterson, W., Weldon, E.: Error-Correcting Codes. MIT Press, Cambridge, MA (1972)
12. Shparlinski, I., Tsfasman, M., Vladut, S.: Curves with many points and multiplication in finite fields. In Stichtenoth, H., Tsfasman, M., eds.: Coding Theory and Algebraic Geometry, Berlin, Springer (1992) 145–169 Lecture Notes in Mathematics 1518.
13. Strassen, V.: Vermeidung von divisionen. Journal für Reine und Angewandte Mathematik **264** (1973)
14. Winograd, S.: Some bilinear forms whose multiplicative complexity depends on the field constants. Mathematical System Theory **10** (1976/77) 169–180

Characterizing TC^0 in Terms of Infinite Groups

Andreas Krebs, Klaus-Jörn Lange, and Stephanie Reifferscheid

WSI, Sand 13, D-72076 Tübingen
{krebs, lange, reiffers}@informatik.uni-tuebingen.de

Abstract. We characterize the languages in $TC^0 = \mathcal{L}(Maj[<, Bit])$ and $\mathcal{L}(Maj[<])$ as inverse morphic images of certain groups. Necessarily these are infinite, since nonregular sets are concerned. To limit the power of these infinite algebraic objects, we equip them with a finite *type set* and introduce the notion of a *finitely typed* (infinite) monoid. Following this approach we investigate *type respecting* mappings and construct a new type of block product, which is more adequate to deal with infinite monoids. We exhibit two classes of finitely typed groups which exactly characterize TC^0 and $\mathcal{L}(Maj[<])$ via inverse morphisms.

1 Introduction

There are very close relations between families of regular languages and low level complexity classes ([1, 2]). In particular the class AC^0 corresponds to the family of star-free regular languages, the class ACC^0 to the family of solvable regular languages, and NC^1 to the family of all regular languages. Unfortunately, the class TC^0 is not treatable in the framework of regular languages unless it coincides with ACC^0 or NC^1.

These connections between families of regular languages and low level complexity classes are exhibited in the unifying framework of logic, where their difference is expressed by being allowed to use the BIT predicate or not. The class TC^0 is expressible as the family of all languages acceptable by formulae built of first-order and majority quantifiers using order and the BIT predicate ([1]), hence a natural candidate for a formal language counterpart of TC^0 is the family $\mathcal{L}(Maj[<])$ of all languages represented by majority formulae using only the order predicate and not the BIT predicate. The various families of regular languages can be characterized algebraically, e.g. the star-free regular sets are those recognizable by aperiodic finite monoids using inverse morphisms. It was the original starting point of this work to exhibit monoids which recognize by inverse morphisms exactly the languages in $\mathcal{L}(Maj[<])$.

When characterizing languages recognized by various classes of finite monoids in terms of certain formula classes the crucial point is to exhibit algebraic objects which correspond to logic operators. For instance, the application of a modular counting quantifier is characterized by building the block product with a finite cyclic group, and in the same way, first-order quantifiers are expressed by the block product with the two-element monoid U_1 ([2]). It seems natural to relate

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 496–507, 2005.

majority (and counting and threshold quantifiers as well) to the block product with the group $\mathbb{Z}$ of the integers. Unfortunately, for infinite monoids M and N the set of all functions from $N \times N$ to M is uncountable, thus the ordinary block product $M \,\square\, N \cong M^{N\times N} * *N$ is too powerful and it wouldn't be possible to simulate these monoids by logical formulae. Hence we need a new, more restricted version of the algebraic approach. We obtain this in two steps: First, we introduce the notion of a *finitely typed monoid*, that is a monoid which is the finite disjoint union of subsets called *types*. When recognizing languages with such a monoid, the inverse morphism is in some sense not allowed to distinguish between the elements inside a type. In a second step we restrict the set of all functions from $N \times N$ to M to the set V of all so-called *type respecting functions*, i.e. functions compatible with the types of N, and define a restricted version of the block product as the bilateral semidirect product of V with N. Building on that we construct two families $\mathbf{M}_{<,Sq}$ and $\mathbf{M}_{<}$ of finitely typed infinite groups, such that TC^0 (resp. $\mathcal{L}(Maj[<])$) coincides precisely with the family of languages recognized by the elements of $\mathbf{M}_{<,Sq}$ (resp. $\mathbf{M}_{<}$).

The paper is organized as follows: Section 2 collects the needed preliminaries. The following two sections define the new block product and use it to construct families of infinite groups which are related to the classes TC^0 and $\mathcal{L}(Maj[<])$. Section 5 summarizes our main results which are proven in the final two sections.

2 Preliminaries

In the following we denote by $\mathbb{Z}$ the integers, by $\mathbb{Z}^+$ the positive integers, by $\mathbb{Z}_0^-$ the nonpositive integers, by $\mathbb{S}$ the positive quadratic numbers, and by $\overline{\mathbb{S}}$ its complement. As usual 2^d stands for $\{0,1\}^d$ for $d \in \mathbb{Z}^+$, whereas $2^{\mathcal{V}}$ denotes the set of all subsets of $\mathcal{V}$. We use e_M to denote the neutral element of a monoid M.

Logic Formulae Over Words. Throughout this paper we consider languages defined by logical formulae. We use essentially the notation as it is presented in the book of Straubing [2]. In general, i, j, k, n will denote positive integers, while x, y, z will denote position variables with positive integer values. Thus, variables will range over $\{1, 2, \ldots, |w|\}$. The predicate $Q_a(x)$ expresses that the position a variable x is pointing to contains the symbol a.

As usual, a formula ϕ with set $\mathcal{V}$ of free variables is interpreted over words as structures $w = (a_1, \mathcal{V}_1)(a_2, \mathcal{V}_2)\cdots(a_n, \mathcal{V}_n)$ such that $\mathcal{V}$ is the union of the $\mathcal{V}_i$ and that the $\mathcal{V}_i$ are pairwise disjoint. Words of this kind are called *$\mathcal{V}$-structures*. A letter $(a, \emptyset)$ is simply denoted by a. We denote the set of all $\mathcal{V}$-structures over Σ by $\Sigma^* \otimes \mathcal{V}$ whereas $(\Sigma \times 2^{\mathcal{V}})^*$ contains also words with multiple occurrences of a variable. The set of $\mathcal{V}$-structures modeling ϕ is denoted by $L_{\phi,\mathcal{V}}$. If ϕ is a sentence, then $L_\phi := L_{\phi,\emptyset} = \{w \in \Sigma^* \mid w \models \phi\}$. We call this the set of words defined by ϕ.

If $w = (a_1, \mathcal{V}_1)(a_2, \mathcal{V}_2)\cdots(a_n, \mathcal{V}_n) \in \Sigma^* \otimes \mathcal{V}$ and $x \notin \mathcal{V}$ then $w_{x=i}$ denotes $(a_1, \mathcal{V}_1)\cdots(a_{i-1}, \mathcal{V}_{i-1})(a_i, \mathcal{V}_i \cup \{x\})(a_{i+1}, \mathcal{V}_{i+1})\cdots(a_n, \mathcal{V}_n) \in \Sigma^* \otimes (\mathcal{V} \cup \{x\})$. As

abbreviation for $w_{x=i} \models \phi$ we often write $w \models \phi(x = i)$, or, if x is understood, $w \models \phi(i)$. We abbreviate the sequence $\phi_1, \ldots, \phi_d$ by $\vec{\phi}$, and $w \models \vec{\phi}$ denotes the boolean vector $w \models \phi_1, \ldots, w \models \phi_d$.

Let $\mathcal{Q}$ be a set of quantifier types (e.g. first-order) and $\mathcal{X}$ a set of numerical predicates (not necessarily containing the order predicate $<$). We denote by $\mathcal{Q}[\mathcal{X}]$ the set of all formulae built over the elements of $\mathcal{X} \cup Q_a(\cdot)$ as atomic formulae by conjunction, negation and quantification using quantifier types from $\mathcal{Q}$. By $\mathcal{L}(\mathcal{Q}[\mathcal{X}])$ we denote the class of all languages definable by $\mathcal{Q}[\mathcal{X}]$ formulae.

Formulae not using the $Q_a(\cdot)$ predicates are *numerical predicates*. In the presence of the order predicate first-order quantifiers can define addition and multiplication by use of the BIT predicate and vice versa. A very comprehensive treatment of related results is given by Schweikardt [3].

We will use $\mathrm{Maj}\, x$ to denote the majority quantifier. $w \models \mathrm{Maj}\, x\ \phi$ is fulfilled iff the number of all $1 \leq i \leq |w|$ such that $w_{x=i} \models \phi$ is larger than $|w|/2$. The majority quantifier rejects in case of a draw.

Ruhl [4] proved that $\mathcal{L}(Maj[<, +, *])$ is not contained in $\mathcal{L}(Maj[<, +])$. Later Lautemann et al. [5] extended this result in showing that all numerical predicates definable by $Maj[<]$-formulae are even definable by $FO[+]$ formulae.

The (unary) counting quantifier is denoted by $\exists^{=y} x$, thus $w_{y=j} \models \exists^{=y} x\ \phi$ is fulfilled iff there are exactly j positions $1 \leq i \leq |w|$ such that $w_{x=i,y=j} \models \phi$ where j is the numerical value of variable y. The counting quantifier can take values in the range $\{0, 1, \ldots, n\}$ which is one more than there are positions available in inputs of size n. If we take care of the case $y = 0$ by the formula $\forall x \neg \phi$ and only treat the case $y \geq 1$ it is possible to show:

Lemma 1 ([6]). *The counting quantifier is definable in* $Maj[<]$.

For the ease of handling we will use the counting quantifier without this restriction and note that more formally in all formulae using the counting quantifier the exception handling of the case $y = 0$ should be added.

Circuits. The reader is assumed to be acquainted with language classes defined by uniform circuits as they are presented by Barrington et al. [1]. In particular we will use the relation

$$\mathrm{TC}^0 = \mathcal{L}(FO + Maj[<, +, *]).$$

TC^0 and the Square Predicate. Let $square(x)$ denote the numerical predicate that the position number the variable x is pointing to is a positive square number. It is well known, that with respect to first-order quantifiers addition and multiplication are equivalent to addition and the square predicate (see e.g. [3][Theorem 2.3.f]). Combined with results in [6] this yields $\mathrm{TC}^0 = \mathcal{L}(Maj[<, square])$.

We will use $\mathrm{Sq}\, x$ to denote the *square quantifier*. $w \models \mathrm{Sq}\, x\ \phi$ is fulfilled iff the number of all $1 \leq i \leq |w|$ such that $w_{x=i} \models \phi$ is a positive square number. It is easy to show that the square predicate *square* is definable in $FO{+}Sq[<]$ logic. On the other hand, the square quantifier can be simulated by the counting quantifier in connection with the square predicate. Hence we have $\mathrm{TC}^0 = \mathcal{L}(Maj + Sq[<])$.

3 Finitely Typed Monoids

We call a monoid M *finitely typed with type set* $\mathfrak{M} = \{\mathcal{M}_i \mid i \in I\}$ iff $M = \dot{\bigcup}_{i\in I}\mathcal{M}_i$ for a finite set I. The pairwise disjoint sets $\mathcal{M}_i,\ i \in I$, are called the *types* of M. We call the elements of $\mathcal{B}(\mathfrak{M})$, (the boolean algebra generated by $\mathfrak{M}$), *extended types* of M. If the type set $\mathfrak{M}$ of M is understood we often simply write M instead of $(M, \mathfrak{M})$.

Let $(M, \mathfrak{M}),\ (N, \mathfrak{N})$ be finitely typed monoids. We call $\phi := (\phi_M, \phi_{\mathfrak{M}}) : (M, \mathfrak{M}) \to (N, \mathfrak{N})$ a *type morphism* iff (i) $\phi_M : M \to N$ is a monoid morphism (ii) $\phi_{\mathfrak{M}} : \mathcal{B}(\mathfrak{M}) \to \mathcal{B}(\mathfrak{N})$ is a lattice morphism and (iii) $\phi_M(\mathcal{M}) = \phi_{\mathfrak{M}}(\mathcal{M}) \cap \phi_M(M)$ for all extended types $\mathcal{M}$ of M. A type morphism $\phi = (\phi_M, \phi_{\mathfrak{M}})$ is called injective (resp. surjective) iff $\phi_M, \phi_{\mathfrak{M}}$ are injective (resp. surjective). The image $\phi_M(M)$ of a finitely typed monoid $(M, \mathfrak{M})$ under a type morphism $\phi = (\phi_M, \phi_{\mathfrak{M}})$ is equipped with the type set $\phi_M(\mathfrak{M})$ given by the set $\mathcal{B}(\{\phi_M(\mathcal{M}) \mid \mathcal{M} \in \mathfrak{M}\})$.

Remark 2. Let M be a monoid and let $\mathfrak{M}_1,\ \mathfrak{M}_2$ be type sets such that $\mathfrak{M}_1$ is coarser than $\mathfrak{M}_2$ (that is, $\mathcal{B}(\mathfrak{M}_1) \subseteq \mathcal{B}(\mathfrak{M}_2)$). Then $\phi = (\phi_M, \phi_{\mathfrak{M}_1}) : (M, \mathfrak{M}_1) \to (M, \mathfrak{M}_2)$ with $\phi_M, \phi_{\mathfrak{M}_1}$ the identities, is an injective type morphism.

We call a monoid $(U, \mathfrak{U})$ a *submonoid* of $(M, \mathfrak{M})$ ($(U, \mathfrak{U}) \leq (M, \mathfrak{M})$) iff there exists an injective type morphism $\phi : (U, \mathfrak{U}) \to (M, \mathfrak{M})$. Analogously, we call a monoid $(N, \mathfrak{N})$ a *quotient* of (M,$\mathfrak{M}$) iff there is a surjective type morphism $\phi : (M, \mathfrak{M}) \to (N, \mathfrak{N})$. Finally we define the *direct product* $(M, \mathfrak{M}) \times (N, \mathfrak{N})$ of finitely typed monoids $(M, \mathfrak{M})$ and $(N, \mathfrak{N})$ as the usual Cartesian product equipped with $\mathfrak{M} \times \mathfrak{N} := \{\mathcal{M} \times \mathcal{N} \mid \mathcal{M} \in \mathfrak{M},\ \mathcal{N} \in \mathfrak{N}\}$.

Let $(M, \mathfrak{M}),\ (N, \mathfrak{N})$ be finitely typed monoids. Recall that the ordinary block product of M with N is defined as the bilateral semidirect product of $M^{N\times N}$ with N, where the right (resp. left) action of N on $M^{N\times N}$ is given by $(f \cdot n)(n_1, n_2) := f(n_1, nn_2)$(resp. $(n \cdot f)(n_1, n_2) := f(n_1 n, n_2)$), $n,\ n_1,\ n_2 \in N,\ f \in M^{N\times N}$. As mentioned above, this structure would be too powerful for our purposes, so in order to define a more restricted version of the block product we introduce two kinds of functions compatible with the type structure. We call a function $f : N \times N \to M$ *strongly type respecting* iff $f|_{\mathcal{N}_1\times\mathcal{N}_2}$ is constant for all $\mathcal{N}_1,\ \mathcal{N}_2 \in \mathfrak{N}$. The second kind of functions are the *type dependent* functions: Assign to each $\mathcal{N} \in \mathfrak{N}$ an element $m_{\mathcal{N}} \in M$ and denote this collection by $m_{\mathfrak{N}} = (m_{\mathcal{N}})_{\mathcal{N}\in\mathfrak{N}}$. Then for $c \in N$ we denote the type dependent function $f_c^{m_{\mathfrak{N}}} : N \times N \to M$ by $f_c^{m_{\mathfrak{N}}}(n_1, n_2) = m_{\mathcal{N}}$ iff $n_1 \cdot c \cdot n_2 \in \mathcal{N}$.

Definition 3 (Block product). *Let* $(M, \mathfrak{M}),\ (N, \mathfrak{N})$ *be finitely typed monoids. The* finitely typed block product $(W, \mathfrak{W}) := (M, \mathfrak{M}) \,\square\, (N, \mathfrak{N})$ *of* $(M, \mathfrak{M})$ *with* $(N, \mathfrak{N})$ *is defined as the bilateral semidirect product* $V * * N$ *of* V *with* N *(with respect to the action given above) where* V *is the monoid generated by* $V^{(1)}$ *and* $V^{(2)}$ *with*

- $V^{(1)} = \{f_{s,t} : N \times N \to M \mid s \cdot f \cdot t$ *is strongly type respecting* $, s, t \in N\}$
- $V^{(2)} = \{f : N \times N \to M \mid f$ *is type depending* $\}$.

We call the elements of V type respecting *(w.r.t.* N *and* M*). The type set* $\mathfrak{W}$ *of* W *consists of all types* $\mathcal{W}_{\mathcal{M}} = \{(f,n) \in W \mid f(e_N, e_N) \in \mathcal{M}\}$*, where* $\mathcal{M} \in \mathfrak{M}$*.*

Remark 4. (a) As usual we write the operation in V additively to provide a more readable notation. Note that this does not imply that V is commutative. By definition of the bilateral semidirect product we have

$$(f_1, n_1) \dots (f_r, n_r) = (\sum_{i=1}^{r} n_1 \dots n_{i-1} \cdot f_i \cdot n_{i+1} \dots n_r, n_1 \dots n_r).$$

The neutral element of $(M, \mathfrak{M}) \,\square\, (N, \mathfrak{N})$ is $(\mathbf{e}, e_N)$ where $\mathbf{e}$ is the function mapping all elements to the neutral element of M. If M, N are groups, then $(M, \mathfrak{M}) \,\square\, (N, \mathfrak{N})$ is a group as well and $(f,n)^{-1} = (-n^{-1} \cdot f \cdot n^{-1}, n^{-1})$.

(b) The range of f is finite for every $f \in V$.

(c) For $f \in V$ there is a finite set $\{f_j \mid j \in J\} \subseteq V^{(1)} \cup V^{(2)}$ with $f = \sum_{j \in J} f_j$. Using the notation above we have $f_j = f_{s_j,t_j}$ iff $f_j \in V^{(1)}$ and $f_j = f_{c_j}^{m_{\mathfrak{N}}^j}$ iff $f_j \in V^{(2)}$. Thus we can determine $f(n_1, n_2)$ by determining types $\mathcal{N}_{j_1}$, $\mathcal{N}_{j_2}$, $\mathcal{N}_{j_3} \in \mathfrak{N}$ such that (i) $n_1 \cdot s_j \in \mathcal{N}_{j_1}$ for $f_j \in V^{(1)}$ (ii) $t_j \cdot n_2 \in \mathcal{N}_{j_2}$ for $f_j \in V^{(1)}$ and (iii) $n_1 \cdot c_j \cdot n_2 \in \mathcal{N}_{j_3}$ for $f_j \in V^{(2)}$.

Definition 5. *A finitely typed monoid* $(M, \mathfrak{M})$ *is said to* accept *a language* $L \subseteq \Sigma^*$ *iff there is a morphism* $h : \Sigma^* \to M$ *and a subset* $\{\mathcal{M}_1, \dots \mathcal{M}_k\} \subseteq \mathfrak{M}$ *of types of* M *such that* $L = h^{-1}(\bigcup_{i=1}^{k} \mathcal{M}_i)$*. If the type set* $\mathfrak{M}$ *of* M *is fixed we simply say that* M *accepts* L*.*

Let $\mathbf{M}$ *be a class of finitely typed monoids. We denote the set of all languages accepted by some element of* $\mathbf{M}$ *with* $H^{-1}(\mathbf{M})$*.*

Remark 6. If a language L is accepted by a submonoid or a quotient of M, then M accepts L as well.

4 The Class $\mathbf{M}_{\mathfrak{Z}}$

Let $\mathfrak{Z}$ be a type set of $\mathbb{Z}$. We define the class $\mathbf{M}_{\mathfrak{Z}}$ as the smallest class of finitely typed groups containing the group $(\mathbb{Z}, \mathfrak{Z})$ and being closed under forming finite direct products and block products. Let $\mathbf{M}$ be a class of finitely typed monoids closed under forming finite direct products. We denote by $\overline{\mathbf{M}}$ the smallest class $\mathbf{C}$ of finitely typed monoids containing $\mathbf{M}$ such that $\mathbf{C}$ is closed under forming submonoids and under forming quotients. As usual (see for instance [7]), the following can be shown:

Remark 7. (a) $\overline{\mathbf{M}}$ is the class of all finitely typed monoids $(G, \mathfrak{G})$ such that there exists $(H, \mathfrak{H}) \in \mathbf{M}$, a submonoid $(U, \mathfrak{U})$ of $(H, \mathfrak{H})$ and a surjective type morphism $\phi : (U, \mathfrak{U}) \to (G, \mathfrak{G})$.

(b) $\overline{\mathbf{M}}$ is closed under forming finite direct products.
(c) For a partition $\mathfrak{Z}$ of $\mathbb{Z}$ we have $H^{-1}(\mathbf{M}_{\mathfrak{Z}}) = H^{-1}(\overline{\mathbf{M}_{\mathfrak{Z}}})$.
(d) Let $\mathfrak{Z}_1$, $\mathfrak{Z}_2$ be partitions of $\mathbb{Z}$ such that $\mathfrak{Z}_1$ is coarser than $\mathfrak{Z}_2$. Then $\overline{\mathbf{M}_{\mathfrak{Z}_1}} \subseteq \overline{\mathbf{M}_{\mathfrak{Z}_2}}$. In particular, $H^{-1}(\mathbf{M}_{\mathfrak{Z}_1}) \subseteq H^{-1}(\mathbf{M}_{\mathfrak{Z}_2})$.

Lemma 8. *Let* $\mathfrak{Z}_1$, $\mathfrak{Z}_2$ *be type sets of* $\mathbb{Z}$. *Then* $\overline{\mathbf{M}_{\mathfrak{Z}_1} \times \mathbf{M}_{\mathfrak{Z}_2}} = \overline{\mathbf{M}_{\mathfrak{Z}_1 \cap \mathfrak{Z}_2}}$, *where* $\mathfrak{Z}_1 \cap \mathfrak{Z}_2 = \{\mathcal{Z}_1 \cap \mathcal{Z}_2 \mid \mathcal{Z}_i \in \mathfrak{Z}_i,\ i = 1, 2\}$, *and* $\mathbf{M}_{\mathfrak{Z}_1} \times \mathbf{M}_{\mathfrak{Z}_2} = \{M_1 \times M_2 \mid M_i \in \mathbf{M}_{\mathfrak{Z}_i}\}$.

Definition 9. *In the following we denote by* $\mathbf{M}_<$ *the class* $\mathbf{M}_{\mathfrak{Z}}$ *for* $\mathfrak{Z} = \mathfrak{Z}_< = \{\mathbb{Z}^+, \mathbb{Z}_0^-\}$ *and by* $\mathbf{M}_{<,Sq}$ *the class* $\mathbf{M}_{\mathfrak{Z}}$ *for* $\mathfrak{Z} = \mathfrak{Z}_{<,Sq} = \{\mathbb{S}, \mathbb{Z}^+ \setminus \mathbb{S}, \mathbb{Z}_0^-\}$.

5 Main Results

The central result of this work is the characterization of the languages in $\mathcal{L}(Maj[<])$ and in TC^0 as inverse morphic images of groups in $\mathbf{M}_<$ and $\mathbf{M}_{<,Sq}$. We prove this in two parts. Theorem 10 covers the direction from logic to groups. The more difficult direction from groups to logic is done in theorem 11.

Theorem 10. *Let* ϕ *be a formula of* $Maj[<]$ *(resp.* $Maj + Sq[<]$*) and* $\mathcal{V}$ *be a set of first-order variables containing the free variables of* ϕ. *Then there is a group* $(G, \mathfrak{G}) \in \mathbf{M}_<$ *(resp.* $\mathbf{M}_{<,Sq}$*) and a monoid morphism* $h : (\Sigma \times 2^{\mathcal{V}})^* \to G$ *fulfilling* $L_{\phi,\mathcal{V}} = h^{-1}(\mathcal{G}) \cap (\Sigma^* \otimes \mathcal{V})$ *for some extended type* $\mathcal{G} \in \mathcal{B}(\mathfrak{G})$.

Theorem 11. *Let* $(G, \mathfrak{G}) \in \mathbf{M}_<$ *(resp.* $\mathbf{M}_{<,Sq}$*),* $\mathcal{V}$ *be a set of variables and let* $h : (\Sigma \times 2^{\mathcal{V}})^* \to G$ *be a monoid morphism. Then there is for each* $\mathcal{G} \in \mathfrak{G}$ *a* $Maj[<]$*-formula (resp.* $Maj + Sq[<]$*-formula) with free variables in* $\mathcal{V}$ *that defines the language* $L = h^{-1}(\mathcal{G}) \cap (\Sigma^* \otimes \mathcal{V})$.

Corollary 12. $\mathcal{L}(Maj[<]) = H^{-1}(\mathbf{M}_<)$ *and* $TC^0 = H^{-1}(\mathbf{M}_{<,Sq})$.

6 Proof of Theorem 10

This proof is done by induction on the term structure of ϕ.

(1) ϕ is $Q_a(x)$ for some $a \in \Sigma$: We recognize the set $L := \{(a_1, S_1) \dots (a_n, S_n) \in (\Sigma \times 2^{\mathcal{V}})^* \mid$ at position x there is an $a\}$ by $(\mathbb{Z}, \mathfrak{Z}_<)$ with the morphism

$$h : \ (\Sigma \times 2^{\mathcal{V}})^* \to \mathbb{Z},\ (\alpha, S) \mapsto \begin{cases} 1 & \text{if } \alpha = a \text{ and } x \in S, \\ 0 & \text{otherwise.} \end{cases}$$

Then $L = h^{-1}(\mathbb{Z}^+)$, thus $L_{\phi,\mathcal{V}} = h^{-1}(\mathbb{Z}^+) \cap (\Sigma^* \otimes \mathcal{V})$.

(2) ϕ is $x < y$: We recognize the set L of all words in $(\Sigma \times 2^{\mathcal{V}})^*$ such that x is positioned before y by the finitely typed block product $(G, \mathfrak{G}) = (\mathbb{Z}, \mathfrak{Z}_<) \,\square\, (\mathbb{Z}, \mathfrak{Z}_<) = V * * \mathbb{Z}$. We use the morphism $h : (\Sigma \times 2^{\mathcal{V}})^* \to G$ given by

$$h(a, S) := \begin{cases} (\mathbf{e}, 0) & \text{if } \{x, y\} \cap S = \emptyset \\ (\mathbf{e}, 1) & \text{if } x \in S \text{ and } y \notin S \\ (f_>, 0) & \text{if } y \in S, \end{cases}$$

where $\mathbf{e}(m_1, m_2) = 0$ for all $m_1, m_2 \in \mathbb{Z}$, and $f_>$ denotes the characteristic function on $\mathbb{Z}^+ \times \mathbb{Z}$. Obviously $f_> \in V^{(1)} \subseteq V$.
Set $\mathcal{G} := \{(f, n) \in V * *\mathbb{Z} \mid f(0,0) \in \mathbb{Z}^+\}$. Then $L = h^{-1}(\mathcal{G})$, thus $L_{\phi,\mathcal{V}} = h^{-1}(\mathcal{G}) \cap (\Sigma^* \otimes \mathcal{V})$.

(3) ϕ is $\phi_1 \wedge \phi_2$: By induction $L_{\phi_i,\mathcal{V}} = h_i^{-1}(\mathcal{G}_i) \cap (\Sigma^* \otimes \mathcal{V})$, for suitable $h_i, \mathcal{G}_i$ and $i = 1, 2$, hence $L_{\phi,\mathcal{V}} = h^{-1}(\mathcal{G}) \cap (\Sigma^* \otimes \mathcal{V})$ where $h = h_1 \times h_2$, $G = G_1 \times G_2$ and $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$.

(4) ϕ is $\neg\psi$: By induction $L_\psi = h^{-1}(\mathcal{G}) \cap (\Sigma^* \otimes \mathcal{V})$, h, $\mathcal{G}$ suitable. Hence $L_{\phi,\mathcal{V}} = h^{-1}(\overline{\mathcal{G}}) \cap (\Sigma^* \otimes \mathcal{V})$ where $\overline{\mathcal{G}} = G \setminus \mathcal{G}$.

(5) ϕ is $\mathrm{Maj}\, x\ \psi$: By induction there exist $(G_\psi, \mathcal{G}_\psi)$ and a monoid morphism $h_\psi : (\Sigma \times 2^{\mathcal{V} \cup \{x\}})^* \to G_\psi$ with $L_{\psi,\mathcal{V}\cup\{x\}} = h_\psi^{-1}(\mathcal{G}_\psi) \cap (\Sigma^* \otimes \mathcal{V} \cup \{x\})$ for a suitable extended type $\mathcal{G}_\psi \in \mathcal{B}(\mathfrak{G}_\psi)$.
We set $(G, \mathfrak{G}) := (\mathbb{Z}, \mathfrak{Z}_<) \square (G_\psi, \mathfrak{G}_\psi) = V * *G_\psi$ and define a morphism $h : (\Sigma \times 2^{\mathcal{V}})^* \to G$ by $h(a, S) := (f_{(a,S)}, h_\psi(a, S))$ where

$$f_{(a,S)}(m_1, m_2) = \begin{cases} 1 & \text{if } m_1 \cdot h_\psi(a, S \cup \{x\}) \cdot m_2 \in \mathcal{G}_\psi, \\ -1 & \text{otherwise.} \end{cases}$$

Then $f_{(a,S)}$ is in $V^{(2)} \subseteq V$. By assumption, an input word $(a_1, S_1) \cdots (a_n, S_n) \in \Sigma^* \otimes \mathcal{V}$ is a model for ϕ iff the number of positions $1 \le i \le n$ fulfilling $h_\psi((a_1, S_1) \cdots (a_{i-1}, S_{i-1})(a_i, S_i \cup \{x\})(a_{i+1}, S_{i+1}) \cdots (a_n, S_n)) \in \mathcal{G}_\psi$ is larger than that of nonfulfilling positions. Hence by definition of $f_{(a,S)}$

$$\pi_1(h((a_1, S_1) \cdots (a_n, S_n)))(e_{G_\psi}, e_{G_\psi}) = \sum_{i \text{ fulfilling } \psi} 1 - \sum_{i \text{ not fulfilling } \psi} 1$$

and consequently $L_{\phi,\mathcal{V}} = h^{-1}(\mathcal{G}) \cap (\Sigma^* \otimes \mathcal{V})$ for $\mathcal{G} := \{(f, m) \in V * *G_\psi \mid f(e_{G_\psi}, e_{G_\psi}) > 0\}$.

(6) ϕ is $\mathrm{Sq}\, x\ \psi$: We define the morphism h as for $\mathrm{Maj}\, x\ \psi$, but now mapping nonfulfilling positions to 0 instead of -1. Thus we simply count the number of satisfying positions and ignore the unsatisfying ones. Hence $L_{\phi,\mathcal{V}} = h^{-1}(\mathbb{S}) \cap (\Sigma^* \otimes \mathcal{V})$.

An Example

The following example demonstrates how to simulate modular counting by groups in $\mathbf{M}_<$. We deal with the formula $\chi := \mathrm{Maj}\, x\ \mathrm{Maj}\, y\ x \le y$ which accepts the set of all words of odd length. Since we do not use $Q_a(\cdot)$-predicates, let's simply take $\Sigma := \{a\}$ to be a singleton. The syntactic monoid of L_χ is the cyclic group with two elements.

As a first step, we simulate the formula $\psi = (x \le y)$ with the set of free variables $\mathcal{V} := \{x, y\}$. We begin with the group $(G_\psi, \mathfrak{G}_\psi) = (\mathbb{Z}, \mathfrak{Z}_<) \square (\mathbb{Z}, \mathfrak{Z}_<) = V_\psi * *\mathbb{Z}$. Since we simulate $\le$ (and not $<$), we define $h_\psi : (\{a\} \times 2^{\mathcal{V}})^* \to G_\psi$ by

$$h_\psi(a, S) := \begin{cases} (\mathbf{e_1}, 0) & \text{if } S = \emptyset \\ (\mathbf{e_1}, 1) & \text{if } S = \{x\} \\ (f_>, 0) & \text{if } S = \{y\} \\ (\mathbf{e_1}, 1)(f_>, 0) = (1 \cdot f_>, 1) & \text{if } S = \{x, y\} \end{cases}$$

where $f_>$ denotes the the characteristic function on $\mathbb{Z}^+ \times \mathbb{Z}$, and $\mathbf{e_1}$ is the constant zero mapping from $\mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z}$. As acceptance type we use the set $\mathcal{G}_\psi := \{(f,n) \mid f(0,0) > 0\}$. Then the set of all $\mathcal{V}$-structures contained in $h_\psi^{-1}(\mathcal{G}_\psi)$ is precisely the set of words where variable x is not positioned after variable y.

In the second step we simulate the formula $\phi := \mathrm{Maj}\, y\ \psi$ which has the set of free variable $\mathcal{V}' := \{x\}$. We set $(G_\phi, \mathfrak{G}_\phi) := (\mathbb{Z}, \mathfrak{Z}_<) \square (G_\psi, \mathfrak{G}_\psi) = V_\phi * *\mathbb{Z}$, and define maps $g_{(a,\emptyset)}$ and $g_{(a,\{x\})}$ from $G_\psi \times G_\psi$ to $\mathbb{Z}$ by

$$g_{(a,\emptyset)}((f_1,n_1),(f_2,n_2)) := \begin{cases} +1 & \text{if } (f_1,n_1)h_\psi(a,\{y\})(f_2,n_2) \in \mathcal{G}_\psi \\ -1 & \text{otherwise} \end{cases}$$

$$g_{(a,\{x\})}((f_1,n_1),(f_2,n_2))) := \begin{cases} +1 & \text{if } (f_1,n_1)h_\psi(a,\{y,x\})(f_2,n_2) \in \mathcal{G}_\psi \\ -1 & \text{otherwise} \end{cases}.$$

Define $h_\phi : (\{a\} \times 2^{\{x\}})^* \to G_\phi$ by $h_\phi(a,\emptyset) := (g_{(a,\emptyset)}, (\mathbf{e_1}, 0))$ and $h_\phi(a,\{x\}) := (g_{a,\{x\}}, (\mathbf{e_1}, 1))$ and set $\mathcal{G}_\phi := \{(g,(f,n)) \mid g((\mathbf{e_1},0),(\mathbf{e_1},0)) > 0\}$. Then the set of all $\{x\}$-structures lying in $h_\phi^{-1}(\mathcal{G}_\phi)$ is exactly the set of words where the variable x is positioned in the first half of the word and this is the language accepted by formula ϕ.

In the final step we simulate the whole formula χ which is $\mathrm{Maj}\, x\ \phi$ and has no free variables. We set $(G_\chi, \mathfrak{G}_\chi) := (\mathbb{Z}, \mathfrak{Z}_<) \square (G_\phi, \mathfrak{G}_\phi) = V_\chi * *\mathbb{Z}$ and define $h_\chi : \{a\}^* \to G_\chi$ by $h_\chi(a,\emptyset) = (F_{(a,\emptyset)}, h_{\phi(a,\emptyset)})$ where $F_{(a,\emptyset)} : G_\phi \times G_\phi \to \mathbb{Z}$ is given as

$$F_{(a,\emptyset)}(m_1,m_2) = \begin{cases} 1 & \text{if } m_1 \cdot h_\phi(a,\{x\}) \cdot m_2 \in \mathcal{G}_\phi \\ -1 & \text{otherwise} \end{cases}$$

Further set $\mathcal{G}_\chi := \{(F,(g,(f,n))) \in V_\chi * *G_\phi \mid F((\mathbf{e_2},(\mathbf{e_1},0)),(\mathbf{e_2},(\mathbf{e_1},0))) > 0\}$ where $\mathbf{e_2}$ denotes the constant zero mapping from $G_\psi \times G_\psi$ to $\mathbb{Z}$. Since $\pi_1(h_\chi(a^n))(e_{G_\chi}, e_{G_\chi}) = \lfloor \frac{n+1}{2} \rfloor - n + \lfloor \frac{n+1}{2} \rfloor$, we have $h^{-1}(\mathcal{G}_\chi) = (aa)^*a$ which is the language accepted by formula χ.

Remark 13. For an arbitrary natural number k the language $L = \{a^n \mid n \equiv 1 \mod (k+1)\}$ can be essentially recognized as above but now setting the positive value of $g_{(a,\emptyset)}$ to k and the negative value of $F_{(a,\emptyset)}$ to $-k$.

7 Proof of Theorem 11

Group quantifiers turn out to be a key ingredient to prove the converse.

Definition 14 (Group quantifier). *Let $(G,\mathfrak{G})$ be a finitely typed group, and let $\mathcal{G}$ be an extended type of G. Further let $d \in \mathbb{N}$, $\phi_1(x),\ldots,\phi_d(x)$ be formulae and $m : 2^d \to G$ be a function such that $e_G \in im(m)$.*

Then $\Gamma^{G,\mathcal{G},m}\, x\, \langle \phi_1(x),\ldots,\phi_d(x)\rangle$ is also a formula, which is modeled by w iff

$$\left(\prod_{i=1}^{n} m(w \models \phi_1(i),\ldots,w \models \phi_d(i))\right) \in \mathcal{G}.$$

In this context we identify false with 0 and true with 1.

This definition is an extension of the group quantifier over finite groups [1]. The difference between this and the finite case is that the map m does not need to be surjective and the accepting set cannot be a single element but has to be a type. We add the map m to the notation of the group quantifier since it strongly influences its power. The requirement $e_G \in \mathrm{im}(m)$ is only for technical reasons and is not substantial as seen in example 17.

Definition 15. $(G, \mathfrak{G})$ *is definable in* $\mathcal{Q}[\mathcal{X}]$ *(denoted by:* $(G, \mathfrak{G}) \triangleright \mathcal{Q}[\mathcal{X}]$*) iff* $\forall d \in \mathbb{N}$, $\forall m : 2^d \to G$ *and* $\forall \mathcal{G} \in \mathfrak{G}$*:* $\Gamma^{G,\mathcal{G},m}$ *is definable in* $\mathcal{Q}[\mathcal{X}]$.

Example 16. We can write the $\exists$ quantifier as a group quantifier over $\mathbb{Z}$.

$$\exists x\, \phi(x) = \Gamma^{\mathbb{Z},\mathbb{Z}^+,m}\, x\, \langle \phi(x) \rangle$$

where $m(false) = 0$ and $m(true) = 1$.

Example 17. We can write the Maj quantifier as a group quantifier over $\mathbb{Z}$.

$$\mathrm{Maj}\, x\, \phi(x) = \Gamma^{\mathbb{Z},\mathbb{Z}^+,m}\, x\, \langle \phi(x) \rangle$$

where $m(false) = -1$ and $m(true) = 1$. This does not quite meet the definition since $0 \notin im(m)$, but we could easily extend the definition to:

$$\mathrm{Maj}\, x\, \phi(x) = \Gamma^{\mathbb{Z},\mathbb{Z}^+,m'}\, x\, \langle \phi(x), false \rangle$$

where $m'(false, false) = -1$ and $m'(true, false) = 1$ and to meet the definition $m'(false, true) = m'(true, true) = 0$.

Definition 18 (Relative group quantifier). *Let the notation be as in definition 14 and let* p, q *be two free variables. Then the formula* $\Gamma^{G,\mathcal{G},m}_{p,q}\, x\, \langle \vec{\phi}(x) \rangle$ *is modeled by the word* $w_{p=k,q=l}$ *iff* $\left(\prod_{i=k}^{l} m(w_{p=k,q=l} \models \vec{\phi}(i)) \right) \in \mathcal{G}$.

Lemma 19. *If a group quantifier is definable in* $\mathcal{Q}[<, \mathcal{X}]$ *then the relative group quantifier is definable in* $\mathcal{Q}[<, \mathcal{X}]$ *as well.*

Proof. We simply map all values outside the relevant interval to the identity. □

If the group quantifier is definable in a logic $\mathcal{Q}[\mathcal{X}]$ then all inverse morphic images of this group belong to $\mathcal{L}(\mathcal{Q}[\mathcal{X}])$:

Lemma 20. $(G, \mathfrak{G}) \triangleright \mathcal{Q}[\mathcal{X}]$ *implies that for each morphism* $h : \Sigma^* \to G$, $h^{-1}(\mathcal{G}) \in \mathcal{L}(\mathcal{Q}[\mathcal{X}])$ *for all* $\mathcal{G} \in \mathfrak{G}$.

Proof. Let h and $\mathcal{G}$ be as above, $\Sigma = \{\alpha_1, \ldots, \alpha_r\}$ and set $\phi_i(x) = Q_{\alpha_i}(x)$ for $i = 1, \ldots, r$. We define a mapping $m : 2^{|\Sigma|} \to G$ by

$$m(false, \ldots, false, true, false, \ldots, false) := h(\alpha_i)$$

where i is the position of $true$. Then $w \models \Gamma^{G,\mathcal{G},m} \langle \vec{\phi}(x) \rangle$ iff $h(w) \in \mathcal{G}$ for every $w \in \Sigma^*$. □

We prove by induction on the block structure that the elements of $\mathbf{M}_<$ and $\mathbf{M}_{<,Sq}$ are definable in $Maj[<]$ and $Maj + Sq[<]$.

Proposition 21. $(\mathbb{Z}, \{\mathbb{Z}^+, \mathbb{Z}_0^-\}) \triangleright Maj[<]$.

Proof. Since the group quantifier has only a finite map in the group, we only have to deal with linear bounded sums in this case. We use extended arithmetic as described in [3] and extend counting quantifiers from lemma 1 to the linear case. Then we can compute the value in $\mathbb{Z}$. □

Proposition 22. $(\mathbb{Z}, \{\mathbb{S}, \overline{\mathbb{S}}\}) \triangleright Maj + Sq[<]$.

Proof. Similar to the previous proof. We compute the value in $\mathbb{Z}$ and use an extended Sq quantifier to check if the value is a square. □

This combined with lemma 8 yields:

Corollary 23. $(\mathbb{Z}, \{\mathbb{S}, \mathbb{Z}^+ \setminus \mathbb{S}, \mathbb{Z}_0^-\}) \triangleright Maj + Sq[<]$.

We now start with the induction. First, we simulate Cartesian products. This simply reduces to propositional conjunction:

Proposition 24. *Let $\mathcal{Q}[\mathcal{X}]$ be any logic. Assume $(G, \mathfrak{G}), (H, \mathfrak{H}) \triangleright \mathcal{Q}[\mathcal{X}]$, then $(G, \mathfrak{G}) \times (M, \mathfrak{M}) \triangleright \mathcal{Q}[\mathcal{X}]$.*

Proof. Let $\mathcal{A} = \mathcal{G} \times \mathcal{H}$ be a type of $\mathfrak{G} \times \mathfrak{H}$, $m : 2^d \to G \times H, v \mapsto (m_G(v), m_H(v))$, and $\phi_1(x), \ldots, \phi_d(x)$ be formulae in $\mathcal{Q}[\mathcal{X}]$. Then $\Gamma^{G\times H, A, m}\, x\, \vec{\phi}(x)$ is true iff $\Gamma^{G,\mathcal{G},m_G}\, x\, \vec{\phi}(x) \wedge \Gamma^{H,\mathcal{H},m_H}\, x\, \vec{\phi}(x)$ is true. □

Finally we show that groups constructed by the block product also belong to $Maj[<]$ respectively $Maj + Sq[<]$.

Theorem 25. *Let $\mathcal{Q}[\mathcal{X}]$ be any logic.*
Assume $(G, \mathfrak{G}), (H, \mathfrak{H}) \triangleright \mathcal{Q}[\mathcal{X}]$, then $(G, \mathfrak{G}) \,\square\, (H, \mathfrak{H}) \triangleright \mathcal{Q}[\mathcal{X}]$.

Proof. Set $(W, \mathfrak{W}) = (G, \mathfrak{G}) \,\square\, (H, \mathfrak{H})$ and let $\mathcal{W} \in \mathfrak{W}$ and d, $m : 2^d \to W$, $\phi_1(x), \ldots, \phi_d(x)$ as in definition 14. We need to show that $\Gamma^{W,\mathcal{W},m} x\, \vec{\phi}(x)$ is definable in $\mathcal{Q}[\mathcal{X}]$.

For an arbitrary but fixed input w of length n we have $m(\vec{\phi}(\cdot)) : \{1, \ldots, n\} \to W$, $i \mapsto m(w \models \phi_1(i), \ldots, w \models \phi_d(i)) =: (f_i, h_i)$. $\mathcal{W}$ is a type of $\mathfrak{W}$, thus $\mathcal{W} = \{(f, h) \in W \mid f(e_H, e_H) \in \mathcal{G}\}$ for a suitable type $\mathcal{G}$ of G.

$$
\begin{aligned}
w \models \Gamma^{W,\mathcal{W},m} x\, \vec{\phi}(x) &\Leftrightarrow \prod_{i=1}^{n} m(w \models \vec{\phi}(i)) \in \mathcal{W} \\
&\Leftrightarrow \pi_1 \left(\prod_{i=1}^{n} m(w \models \vec{\phi}(i)) \right) (e_H, e_H) \in \mathcal{G} \\
&\Leftrightarrow \sum_{i=1}^{n} f_i(h_1 \cdots h_{i-1}, h_{i+1} \cdots h_n) \in \mathcal{G}
\end{aligned}
$$

In the following we need to show that we can construct formulae $\vec{\psi}(y)$ and a mapping $m' : 2^{d'} \to G$ such that $m'(w_{y=i} \models \vec{\psi}) = f_i(h_1 \cdots h_{i-1}, h_{i+1} \cdots h_n)$. Then we can use the group quantifier for G and get:

$$I^{G,\mathcal{G},m'}\, y\, \vec{\psi}(y) \text{ iff } I^{W,\mathcal{W},m} x\, \vec{\phi}(x).$$

We already have the mapping $\pi_1 \circ m$ that determines together with $\vec{\phi}(x)$ the function f_i at any position. There are only finitely many different functions f_i since the image of $\pi_1 \circ m$ is finite. So let's fix a function f_i. By remark 4(c) we have $f_i = \sum_{j \in J_i} f_{i,j}$ for $f_{i,j} \in V^{(1)} \cup V^{(2)}$ and we can determine the value of f_i if we have formulae for cases (i)-(iii) of remark 4(c):
(i) Let $s \in \{s_j \mid j \in J_i\}$ and $\mathcal{H} \in \mathfrak{H}$. We define $\phi_{d+1}(x)$ as $x = y$ and $m''_s(\vec{v}, false) := \pi_2 \circ m(\vec{v})$ and $m''_s(\vec{v}, true) := s$ then

$$w_{p=1,y=i} \models I^{H,\mathcal{H},m''_s}_{1,y}\, x\, \langle \vec{\phi}(x), \phi_{d+1}(x) \rangle \text{ iff } h_1 \cdots h_{i-1} \cdot s \in \mathcal{H}$$

Hence we can construct a finite number of formulae to determine the case of clause (i).
(ii) Similar to case (i).
(iii) With the same definitions as above:

$$w_{y=i} \models I^{H,\mathcal{H},m''_s}\, x\, \langle \vec{\phi}(x), \phi_{d+1}(x) \rangle \text{ iff } h_1 \cdots h_{i-1} \cdot s \cdot h_{i+1} \cdots h_n \in \mathcal{H}$$

Hence we can determine the arguments of f_i up to an equivalence class that results in the same value of f_i and use a finite case differentiation to build the formulae $\vec{\psi}(x)$ and the mapping m'. This completes the proof of theorem 11. □

8 Discussion

We characterized the languages in $\mathcal{L}(Maj[<])$ and TC^0 as inverse morphic images of the elements of the group classes $\mathbf{M}_<$ and $\mathbf{M}_{<,Sq}$. As is easily seen, every element of these classes, and consequently every morphic image of a subgroup of such an element, is solvable. Thus, in order to conclude that languages with nonsolvable syntactic monoid (as for example A_5) are not recognizable by elements of $\mathbf{M}_<$, it would be sufficient to show that the image of Σ^* under the recognizing morphism is a subgroup of the accepting finitely typed group. Unfortunately, this is not true in general, since, in contrast to the finite case, a submonoid of an infinite group is not necessarily a group; on the contrary, it is not hard to see that every finitely generated free monoid emerges as submonoid of the (restricted) block product $\mathbb{Z} \,\square\, \mathbb{Z}$.

The example in section 6 demonstrates that all finite cyclic groups are divisors of groups in $\mathbf{M}_<$ of (block) depth four. It can indeed be shown that depth three is sufficient to accept all finite cyclic groups, and, moreover, that any finite solvable group G with solvable length d can be recognized by a finitely typed group of depth $2d+1$. On the other hand, it is not possible to recognize any finite group by a group $(M, \mathfrak{M})$ in $\mathbf{M}_<$ of depth two or less, since, in this case, there is for

each element $m \in M$ a positive integer k such that m^k has the same type as m^{k+l} for an arbitrary $l \in \mathbb{Z}^+$. (Observe that this does not hold for $\mathbf{M}_{<,Sq}$.) Finitely typed groups with this property might be called *pseudo-aperiodic* and it should be interesting to investigate their properties.

The algebraic representations of $\mathcal{L}(Maj[<])$ and TC^0 led to the families of groups $\mathbf{M}_<$ and $\mathbf{M}_{<,Sq}$. A corresponding characterization is possible for the complexity classes AC^0 or ACC^0 both in the case of $FO[<, BIT]$ (or equivalently $FO[+, square]$) uniformity and in the case of $FO[+]$ uniformity. We only have to provide specific groups simulating the addition and the square predicates as initial building blocks which are not allowed to be used in the induction step when simulating quantifiers. It should be remarked that the resulting algebraic families don't enjoy the same closure properties as $\mathbf{M}_<$ and $\mathbf{M}_{<,Sq}$ do.

The next step should be to prove lower bounds. This should be possible for classes like $\mathcal{L}(Maj[<])$ or $FO[+]$ uniform ACC^0 where separation results are not known to have severe and surprising consequences. Here the representation in terms of groups might be helpful: when transforming a circuit class into a logical class of the form $\mathcal{L}(\mathcal{Q}(\mathcal{X}))$ we often lose a reasonable notion of depth. For example, we know that $Maj[<]$ can do counting modulo k for each fixed k, but the depth of the corresponding formula seems inherently to increase with growing k. This is not the case when using groups in $\mathbf{M}_<$. Thus it might be a reasonable task to try to correlate the block depth notion of a group or monoid to the depth of a circuit.

Acknowledgments

We thank Howard Straubing, Pascal Tesson, and Denis Therien for valuable discussions on this topic and the anonymous referees for their very careful reading and many helpful comments.

References

1. Barrington, D., Immerman, N., Straubing, H.: On uniformity within NC^1. J. Comp. System Sci. **41** (1990) 274–306
2. Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Birkhäuser (1994)
3. Schweikardt, N.: On the Expressive Power of First-Order Logic with Built-In Predicates. Dissertation, Universität Mainz (2001)
4. Ruhl, M.: Counting and addition cannot express deterministic transitive closure. In: Proc. of 14th IEEE Symposium On Logic in Computer Science. (1999) 326–334
5. Lautemann, C., McKenzie, P., Schwentick, T., Vollmer, H.: The descriptive complexity approach to logcfl. J. Comp. System Sci. **62** (2001) 629–652
6. Lange, K.: Some results on majority quantifiers over words. In: Proc. of the 19th IEEE Conference on Computational Complexity. (2004) 123–129
7. Lawson, M.: Finite Automata. Chapman & Hall/CRC (2004)

Fast Pruning of Geometric Spanners*

Joachim Gudmundsson[1], Giri Narasimhan[2], and Michiel Smid[3]

[1] Department of Mathematics and Computing Science,
TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
h.j.gudmundsson@tue.nl
[2] School of Computer Science,
Florida International University, Miami, FL 33199, USA
giri@cs.fiu.edu
[3] School of Computer Science,
Carleton University, Ottawa, Canada K1S 5B6
michiel@scs.carleton.ca

Abstract. Let S be a set of points in $\mathbb{R}^d$. Given a geometric spanner graph, $G = (S, E)$, with constant stretch factor t, and a positive constant ε, we show how to construct a $(1+\varepsilon)$-spanner of G with $\mathcal{O}(|S|)$ edges in time $\mathcal{O}(|E| + |S| \log |S|)$. Previous algorithms require a preliminary step in which the edges are sorted in non-decreasing order of their lengths and, thus, have running time $\Omega(|E| \log |S|)$. We obtain our result by designing a new algorithm that finds the pair in a well-separated pair decomposition separating two given query points. Previously, it was known how to answer such a query in $\mathcal{O}(\log |S|)$ time. We show how a sequence of such queries can be answered in $\mathcal{O}(1)$ amortized time per query, provided all query pairs are from a polynomially bounded range.

1 Introduction

Complete graphs represent ideal communication networks, but they are expensive to build; sparse spanners represent low cost alternatives. The number of edges of the spanner network is a measure of its sparseness; other sparseness measures include the weight, the maximum degree and the number of Steiner points. Spanners for complete Euclidean graphs as well as for arbitrary weighted graphs find applications in robotics, network topology design, distributed systems, design of parallel machines, and many other areas, and have been the subject of considerable research [1, 2, 6, 7, 15].

Consider a set S of n points in $\mathbb{R}^d$. Throughout this paper, we will assume that d is constant. A network on S can be modelled as an undirected graph G with vertex set S and with edges $e = (u, v)$ of weight $wt(e)$. In this paper we consider geometric networks, where the weight of the edge $e = (u, v)$ is equal to the Euclidean distance $|uv|$ between its two endpoints u and v. Let $\delta_G(p, q)$

* J.G. was supported by the Netherlands Organisation for Scientific Research (NWO) and M.S. was supported by NSERC.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 508–520, 2005.

denote the length of a shortest path in G between p and q. Then, G is a *t-spanner* for S, if $\delta_G(p,q) \leqslant t \cdot |pq|$ for any two points p and q of S. The minimum value t such that G is a t-spanner for S is called the *stretch factor* of G. A subgraph G' of G is a t'-spanner of G, if $\delta_{G'}(p,q) \leqslant t' \cdot \delta_G(p,q)$ for any points p and q of S.

Many algorithms are known that compute t-spanners with useful properties such as linear size ($\mathcal{O}(n)$ edges), bounded degree, small spanner diameter (i.e., any two points are connected by a t-spanner path consisting of only a small number of edges), low weight (i.e., the total length of all edges is proportional to the weight of a minimum spanning tree of S), and fault-tolerance; for example, see [1, 2, 3, 4, 6, 7, 8, 10, 14, 15, 17, 19], and the surveys [9, 18]. All these algorithms compute t-spanners for any given constant $t > 1$. However, all these algorithms either start with a point set, or with a spanner that has a linear number of edges.

We consider the problem of efficiently *pruning* a given t-spanner, even if it has a superlinear number of edges. That is, given a geometric graph $G = (S, E)$ in $\mathbb{R}^d$ with n points and constant stretch factor t, and a positive constant ε, we consider the problem of constructing a $(1+\varepsilon)$-spanner of G with $\mathcal{O}(n)$ edges. Thus the resulting subgraph of G is guaranteed to be a $(t(1+\varepsilon))$-spanner of S. The greedy algorithm of [7, 10] can be used to compute a $(1+\varepsilon)$-spanner G' of G. However, the greedy algorithm starts by sorting the edges and, thus, has running time $\Omega(|E| \log n)$. In [11], an algorithm was presented with running time $\mathcal{O}(|E| \log n)$, that produces a $(1+\varepsilon)$-spanner G' of G with $\mathcal{O}(n)$ edges.

In this paper, we show how the running time can be improved to $\mathcal{O}(|E| + n \log n)$ time. Furthermore, using the results in [10], we show that with the same time complexity, we can compute a $(1+\varepsilon)$-spanner of G with $\mathcal{O}(n)$ edges and with total weight $\mathcal{O}(wt(MST(S)))$.

In a series of papers by Gudmundsson et al. [11, 12, 13], it was shown that approximate shortest path queries can be answered in constant time using $\mathcal{O}(|E| \log n)$ preprocessing, provided that the given graph is a t-spanner. The time complexity of the preprocessing depends on the time to prune the graph, which was shown to be $\mathcal{O}(|E| \log n)$. Using the pruning algorithm presented here, we improve the preprocessing time of the data structure in [11, 12, 13] to $\mathcal{O}(|E| + n \log n)$. We also improve the time complexity in [16] for computing a $(1+\varepsilon)$-approximation to the stretch factor of a geometric graph to $\mathcal{O}(|E| + n \log n)$, provided we know in advance that the stretch factor is bounded from above by a constant. In Section 5 we consider several other applications.

Our model of computation is the traditional algebraic computation tree model with the added power of indirect addressing.

2 Preliminaries

In the next sections, we will show how to prune a graph. Our construction uses the well-separated pair decomposition of Callahan and Kosaraju [5]. We briefly review this decomposition below.

If X is a bounded subset of $\mathbb{R}^d$, then we denote by $R(X)$ the smallest axes-parallel d-dimensional rectangle that contains X. We call $R(X)$ the *bounding box* of X. Let $l(R(X))$, or $l(X)$, be the length of the longest side of $R(X)$.

Definition 1. *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are* well-separated *with respect to s, if there are two disjoint d-dimensional balls C_A and C_B, having the same radius, such that (i) C_A contains the bounding box $R(A)$ of A, (ii) C_B contains the bounding box $R(B)$ of B, and (iii) the minimum distance between C_A and C_B is at least s times the radius of C_A.*

The parameter s will be referred to as the *separation constant*.

Lemma 1 ([5]). *Let A and B be two finite sets of points that are well-separated w.r.t. s, let x and p be points of A, and let y and q be points of B. Then (i) $|xy| \leqslant (1 + 4/s) \cdot |pq|$, and (ii) $|px| \leqslant (2/s) \cdot |pq|$.*

Definition 2 ([5]). *Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A* well-separated pair decomposition *(WSPD) for S with respect to s is a sequence of pairs of non-empty subsets of S, $\{A_1, B_1\}, \ldots, \{A_m, B_m\}$, such that*

1. *$A_i \cap B_i = \emptyset$, for all $i = 1, \ldots, m$,*
2. *for any two distinct points p and q of S, there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $q \in A_i$ and $p \in B_i$,*
3. *A_i and B_i are well-separated w.r.t. s, for all $i = 1, \ldots, m$.*

Callahan and Kosaraju showed that the fair split tree T can be computed in $\mathcal{O}(n \log n)$ time, and that, given T, a WSPD of size $m = \mathcal{O}(n)$ can be computed in $\mathcal{O}(n)$ time. Each pair $\{A_i, B_i\}$ in this WSPD is represented by two nodes u_i and v_i of T, i.e., we have $A_i = S_{u_i}$ and $B_i = S_{v_i}$. We end this section with two lemmas that will be used later on.

Lemma 2. *Let u and u' be two nodes in the fair split tree T such that u' is in the subtree of u and the path between them contains at least d edges. Then the length of the longest side of the bounding box of u' is at most half the length of the longest side of the bounding box of u.*

Lemma 3. *Let A and B be two sets of points in $\mathbb{R}^d$ that are well-separated with respect to s, and let p and q be points in A and B, respectively. The length of each side of the bounding boxes of A and B is at most $(2/s)|pq|$.*

3 A General Pruning Approach

Recall that we are given a set S of n points in $\mathbb{R}^d$, a t-spanner $G = (S, E)$ for some real constant $t > 1$ and a real constant $\varepsilon > 0$.

Our goal is to compute a sparse $(1 + \varepsilon)$-spanner G' of G. Suppose that there exists a set pairs of points, $P = \{\{a_1, b_1\}, \ldots, \{a_m, b_m\}\}$, with the property that for each edge (p, q) in E, there is an index i such that for some real number s,

1. $|pa_i| \leqslant (2/s)|a_ib_i|$ and $|qb_i| \leqslant (2/s)|a_ib_i|$, or
2. $|pb_i| \leqslant (2/s)|a_ib_i|$ and $|qa_i| \leqslant (2/s)|a_ib_i|$.

In other words, for each edge (p, q) in E, the set P contains a "close approximation". Then, we show below that, if $s = \frac{1}{\varepsilon}((1+\varepsilon)(8t+4)+4)$, there exists a $(1+\varepsilon)$-spanner of G with at most m edges. As the keen reader may have guessed, we will show later that the set P can be easily constructed from a WSPD of S.

To prove the existence of the subgraph G' as a $(1+\varepsilon)$-spanner of G, we prune G *with respect to* the set P of pairs as follows. Let C_i, $1 \leqslant i \leqslant m$, be m lists that are initially empty. For each edge (p, q) in E, pick any index i for which condition 1. or 2. above is satisfied, and add the edge (p, q) to the list C_i. We define G' to be the graph (S, E'), where the edge set E' contains exactly one edge from each non-empty list C_i, $1 \leqslant i \leqslant m$.

Lemma 4. *The graph $G' = (S, E')$ is a $(1+\varepsilon)$-spanner of G.*

The above process essentially prunes G using set P as a "guide". Each edge of G is "mapped" to a pair in P, and in the pruned subgraph, for each pair in P, we retain one edge that is mapped to it (if any). In order to apply the above general result, we need algorithms that do the following:

1. Compute $P = \{a_i, b_i\}_{1 \leqslant i \leqslant m}$, with $m = \mathcal{O}(n)$.
2. For each edge (p, q) in E, compute an index i such that the condition for the set P holds.

A straight-forward approach for step 1, which appears already in [11], is as follows. Compute the WSPD with separation constant $s = ((1+\varepsilon)(8t+4)+4)/\varepsilon$, see Section 2. Given this WSPD, define the set P as follows. For each well-separated pair $\{A_i, B_i\}$, choose a pair consisting of an arbitrary point in A_i and an arbitrary point in B_i; see Fig 1. Using Lemma 1, the properties that are needed for P are satisfied, thus we can apply Lemma 4.

As for step 2, Arya *et al.* [3] showed that, after an $\mathcal{O}(n \log n)$-time preprocessing of the fair split tree, the index i can be computed in $\mathcal{O}(\log n)$ time, for any edge (p, q) in E. Hence, the entire graph G' can be computed in $\mathcal{O}((n + |E|) \log n) = \mathcal{O}(|E| \log n)$ time. We have proved the following result.

Theorem 1. *[11] Given a geometric graph $G = (S, E)$ with n vertices, which is a t-spanner for S, for some real constant $t > 1$, and a real constant $\varepsilon > 0$ we can compute a $(1+\varepsilon)$-spanner of G having $\mathcal{O}(n)$ edges in $\mathcal{O}(|E| \log n)$ time.*

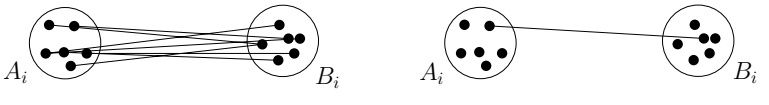


Fig. 1. Pruning the spanner graph using the WSPD

4 An Improved Algorithm

Above we showed that the time-complexity of the algorithm can be written as $\mathcal{O}(n \log n + |E| \cdot \tau(n))$, where $\tau(n)$ is the time needed to find the pair $\{a_i, b_i\}$ in P, given a query (p, q), such that the condition mentioned at the beginning of Section 3 holds. Below, we show a stepwise refinement of the basic scheme.

4.1 Improvements for a Restricted Case – Bounded Aspect Ratio

Let T be the fair split tree for S, and let $\{A_i, B_i\}$, $1 \leqslant i \leqslant m$, be the well-separated pair decomposition of S obtained from T, with separation constant $s > 0$. Let $L > 0$ be a real number, let $c \geqslant 1$ be an integer constant, and let F be a set of k pairs of points in S such that $L/n^c \leqslant |xy| \leqslant L$ holds for each pair $\{x, y\} \in F$. We say that F has *polynomially bounded aspect ratio.*

In this section, we show how to compute, for every $\{x, y\} \in F$, the corresponding well-separated pair, i.e., the index i for which $x \in A_i$ and $y \in B_i$ or $x \in B_i$ and $y \in A_i$. Recall that every node of T stores the bounding box of the set of all points stored in its subtree. Let $\alpha = 2/(\sqrt{d}(s + 4))$. For each point $x \in S$, we define the following nodes in T:

u_x: the highest node on the path from the leaf storing x to the root, such that its bounding box has sides of length at most $(2/s)L$.

u'_x: the highest node on the path from the leaf storing x to the root, such that its bounding box has sides of length at most $\alpha L/n^c$.

Moreover, for each pair $e = \{x, y\} \in F$, we define the following nodes in T.

u_{ex}: the highest node on the path from the leaf storing x to the root, such that its bounding box has sides of length at most $(2/s)|xy|$.

u'_{ex}: the highest node on the path from the leaf storing x to the root, such that its bounding box has sides of length at most $\alpha|xy|$.

Observation 5. *By traversing T, all nodes u_x and u'_x, $x \in S$, can be computed in $\mathcal{O}(n)$ time.*

Because of the polynomially bounded aspect ratio assumption, the path from u'_x to u_x contains all nodes whose subsets contain x and are involved in well-separated pairs corresponding to pairs in F. In particular, the path from u'_{ex} to u_{ex} contains the node whose subset is A_i. This is formalized in the lemma below.

Lemma 6. *Let $e = \{x, y\}$ be a pair in F, and let i be the index such that $x \in A_i$ and $y \in B_i$. Let v_i and w_i be the nodes of T that represent A_i and B_i, respectively. Then,*

1. *if we walk in T from the leaf storing x to the root, then we will encounter the nodes, u'_x, u'_{ex}, v_i, u_{ex}, and u_x, in this order;*
2. *the path in T between u'_x and u_x contains $\mathcal{O}(\log n)$ nodes; and,*
3. *the path in T between u'_{ex} and u_{ex} contains $\mathcal{O}(1)$ nodes.*

Lemma 7. *Let $e = \{x, y\}$ be a pair in F, and let i be the index such that $x \in A_i$ and $y \in B_i$. Let v_i and w_i be the nodes of T that represent A_i and B_i, respectively. Given pointers to the nodes u_{ex} and u_{ey}, the nodes v_i and w_i can be computed in $\mathcal{O}(1)$ time.*

The original problem has now been reduced to finding, for each query pair $e = \{x, y\}$ in F, the nodes u_{ex} and u_{ey} in T, where u_{ex} and u_{ey} correspond to nodes whose bounding boxes are of size close to $(2/s)|xy|$. A simple solution would be as follows. For each point x in S, let $\mathcal{T}_x$ be a balanced binary search tree storing the nodes on the path in T between u'_x and u_x, which by Lemma 6.2 has only $\mathcal{O}(\log n)$ nodes. The key value for these nodes is the length of a longest side of the bounding box.

Lemma 8. *Let $e = \{x, y\}$ be a vertex pair in F. Using the trees $\mathcal{T}_x$ and $\mathcal{T}_y$, the nodes u_{ex} and u_{ey} can be computed in $\mathcal{O}(\log \log n)$ time.*

As a result, we have shown that our restricted problem can be solved in $\mathcal{O}(n \log n + k \log \log n)$ time. Each tree uses $\mathcal{O}(\log n)$ space. Thus, the amount of space used is $\mathcal{O}(n \log n)$. Next we will show that the size can be reduced to $\mathcal{O}(n)$ by observing that all queries are known in advance.

4.2 Achieving Linear Space

Let $x_1, \ldots, x_n$ be the vertices stored in the leafs of T, ordered from left to right. Note that a query pair $e = \{x, y\}$ in F asks for u_{ex} and u_{ey}. This can be viewed as two different queries, i.e., (x, y) and (y, x).

We process the queries in batches. Initially we set $i = 1$. Build $\mathcal{T}_{x_1}$ in linear time. For each query in F of the form $e = (x_1, x_j)$, return u_{ex_1}. A pointer to u_{ex_1} is stored together with x_1 in the query pair (x_1, x_j) in F. When all queries involving x_1 have been answered, i is incremented.

In a generic step we build the binary tree $\mathcal{T}_{x_i}$ from $\mathcal{T}_{x_{i-1}}$ by first deleting the nodes in T that lie on the path between $u_{x_{i-1}}$ and $u'_{x_{i-1}}$, but not on the path between u_{x_i} and u'_{x_i}; and then inserting all nodes in T that lie on the path between u_{x_i} and u'_{x_i}, but not on the path between $u_{x_{i-1}}$ and $u'_{x_{i-1}}$. Since each node in T is inserted and removed at most once, the total time complexity of building the trees $\mathcal{T}_1, \ldots, \mathcal{T}_n$ is $\mathcal{O}(n \log \log n)$.

After $\mathcal{T}_{x_i}$ has been constructed, all queries involving x_i are solved, and the answers are stored together with the pairs in F. The process continues until all queries have been answered. At all times exactly one tree $\mathcal{T}_{x_i}$ is active, thus the total space complexity is dominated by the fair-split tree and the number of edges in F, which is bounded by $\mathcal{O}(k + n)$. We obtain the following lemma.

Lemma 9. *Given the k query pairs $\{e_i = \{p_i, q_i\}\}_{1 \leqslant i \leqslant k}$ in F, one can compute $u_{e_i p_i}$ and $u_{e_i q_i}$ for each $1 \leqslant i \leqslant k$ in total $\mathcal{O}(n \log n + k \log \log n)$ time using $\mathcal{O}(k + n)$ space.*

4.3 Improving the Running Time

In this section we will improve the running time in Lemma 9 to $\mathcal{O}(k + n \log n)$; instead of using the tree $\mathcal{T}_x$ for answering the queries we will use a different data structure, namely an array $A_x[0..\lfloor \log(2n^c) \rfloor]$. Recall that $s = \frac{1}{\varepsilon}((1+\varepsilon)(8t+4)+4)$. Each entry $A_x[j]$ stores a pointer to the highest node on the path in T between u'_x and u_x whose bounding box has sides of length at most $\frac{2^j L}{sn^c}$.

Lemma 10. *Let $e = \{x, y\}$ be a pair in F, let $j = \lfloor \log(\frac{2n^c}{L}|xy|) \rfloor$, and let $A_x[j]$ point to node u^A_{ex}. Then the path between u_{ex} and u^A_{ex} contains $\mathcal{O}(1)$ nodes.*

Since node u^A_{ex} is close to u_{ex}, we can show the following lemma, which is similar to Lemma 7.

Lemma 11. *Let $e = \{x, y\}$ be a pair in F, and let i be the index such that $x \in A_i$ and $y \in B_i$. Let v_i and w_i be the nodes of T that represent A_i and B_i, respectively, and let j be as in Lemma 10. Given $A_x[j]$ and $A_y[j]$, the nodes v_i and w_i can be computed in $\mathcal{O}(1)$ time.*

The above lemma assumes that the index j, defined by $j = \lfloor \log(\frac{2n^c}{L}|xy|) \rfloor$ can be easily determined. We will refer to this index as the *index* of two points x and y in $\mathbb{R}^d$. It remains to prove how k index queries can be answered in total time $\mathcal{O}(k + n \log n)$.

4.4 Answering Index Queries Efficiently

Next we consider how to "bucket" distances in constant time, without using the floor function since the floor function is a non-algebraic function. This problem was considered in [11], but there it was only shown for the special case when the points in the set lie in a polynomially bounded interval, see Fact 14. We extend the result to hold for any point set for which the queries have polynomially bounded aspect ratio. The idea is to scale the point set and then partition it into subsets such that each subset consists of points in a polynomially bounded interval. Furthermore, for every pair in F it will be shown that the two corresponding points in the scaled set will belong to the same subset. Consequently, one may apply the results from [11] to each subset.

The aim of this section is to show the following theorem.

Theorem 2. *Let S be a set of n points in $\mathbb{R}^d$, let $L > 0$ be a real number, and let c be a positive constant. We can preprocess S in $\mathcal{O}(n \log n)$ time, such that for any two points x and y in S with $L/n^c \leqslant |xy| \leqslant L$, we can compute the quantity $\lfloor \log(\frac{2n^c}{L}|xy|) \rfloor$ in constant time, using only algebraic operations and indirect addressing.*

For each $x \in S$, define $x' = \frac{2n^c}{L}x$. This gives a set $V = \{x' : x \in S\}$ of scaled points. Let F' be the set of scaled query pairs $\{x', y'\}$, where $\{x, y\}$ ranges over all pairs in F. If $\{x, y\} \in F$, then $L/n^c \leqslant |xy| \leqslant L$ and, hence, $2 \leqslant |x'y'| \leqslant 2n^c \leqslant n^{c+1}$. Furthermore, $\lfloor \log(\frac{2n^c}{L}|xy|) \rfloor = \lfloor \log |x'y'| \rfloor$.

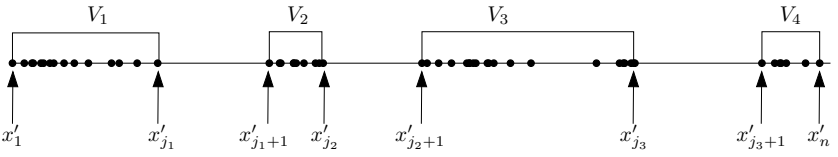


Fig. 2. Illustrating how V is divided into the sets $V_1, \ldots, V_4$. The gap between two sets is larger than n^{c+1}

The One-Dimensional Case. We will assume that V is a set of n points on the real line. First the algorithm partitions V into groups $V_1, \ldots, V_\ell$, in $\mathcal{O}(n \log n)$ time as follows. Sort the points of V in increasing order $x'_1, x'_2, \ldots, x'_n$. Let $j_1 < j_2 < \ldots < j_{\ell-1}$ be *all* the indices such that $x'_{j_1+1} > x'_{j_1} + n^{c+1}$, $x'_{j_2+1} > x'_{j_2} + n^{c+1}$, $\ldots$, $x'_{j_{\ell-1}+1} > x'_{j_{\ell-1}} + n^{c+1}$. In other words, the gaps following $x'_{j_1}, x'_{j_2}, \ldots, x'_{j_{\ell-1}}$ are greater than n^{c+1}. Then we define $V_1 = \{x'_1, \ldots, x'_{j_1}\}$, $V_\ell = \{x'_{j_{\ell-1}+1}, \ldots x'_n\}$, and $V_i = \{x'_{j_{i-1}+1}, \ldots, x'_{j_i}\}$ for $2 \leqslant i \leqslant \ell - 1$; this is illustrated by an example in Fig. 2. The following observation about the sequence $V_1, \ldots, V_\ell$ follows immediately from the above partitioning algorithm.

Observation 12. *Let i and j be two positive integers, such that $i < j \leqslant \ell$. Then, the following statements hold:*

1. *If $x' \in V_i$ and $y' \in V_j$, then $|x'y'| > n^{c+1}$.*
2. *If $x', y' \in V_i$, then $|x'y'| \leqslant n^{c+2}$.*
3. *$V_i \cap V_j = \emptyset$, and $V = V_1 \cup \ldots \cup V_\ell$*

The following lemma gives properties of the scaled queries in F'.

Lemma 13. *Consider the sets $V_1, \ldots, V_\ell$ and F' as defined above. Then,*

1. *For each i with $1 \leqslant i \leqslant \ell$ there exists a real number D_i such that the set V_i is contained in the interval $[D_i, D_i + n^{c+2}]$.*
2. *For every pair $\{x, y\}$ in F, there exists an i, such that both x' and y' are contained in V_i. Moreover, the pair $\{x', y'\}$ in F' satisfies*

$$2 \leqslant |x'y'| \leqslant n^{c+1}, \quad \text{and} \quad \left\lfloor \log \left(\frac{2n^c}{L} |xy| \right) \right\rfloor = \lfloor \log |x'y'| \rfloor.$$

Fact 14 *(Theorem 2.1 in [11]). Let X be a set of n real numbers that are contained in the interval $[D, D + n^k]$, for some real number D and some positive integer constant k. We can preprocess X in $\mathcal{O}(n \log n)$ time, using $\mathcal{O}(n)$ space, such that for any two points p and q of X, with $|pq| \geqslant \beta$, where $\beta > 0$ is a constant, we can compute $\lfloor \log |pq| \rfloor$ in constant time.*

In [11], Fact 14 was only proved for the case when $D = 0$. By translating the points, and observing that this does not change distances, it is clear that Fact 14 holds for any real number D. As a result of Lemma 13, it follows that Fact 14 can be applied to every subset V_i. Furthermore, according to Lemma 13, F' has

polynomially bounded aspect ratio, thus every query pair in F' can be answered in constant time according to Fact 14. Note that, for each point x, we need to store a pointer to the subset V_i of the partition that it belongs to. As a result, we have proved Theorem 2 for the one-dimensional case.

The *d*-Dimensional Case. This is inspired by the algorithm in [11]. Now assume that V is a d-dimensional set of points, where d is a constant. Let $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ be any two points of V with $|pq| \geqslant \beta$, for some constant $\beta > 0$, let j be such that $|p_j - q_j|$ is maximum, and let $i = \lfloor \log |p_j - q_j| \rfloor$. Since $|p_j - q_j| \leqslant |pq| \leqslant \sqrt{d}|p_j - q_j|$, we have $i \leqslant \lfloor \log |pq| \rfloor \leqslant \frac{1}{2} \log d + i$. This suggests the following solution. For each ℓ, $1 \leqslant \ell \leqslant d$, we build the data structure above for the one-dimensional case for the set of ℓ-th coordinates of all points of V. Given two distinct points p and q of V, we compute the index j such that $|p_j - q_j|$ is maximum. Then we use the one-dimensional algorithm to compute the integer $i = \lfloor \log |p_j - q_j| \rfloor$. Note that this algorithm also gives us the value 2^i. Given i and 2^i, we then compute $\lfloor \log |pq| \rfloor$ in $\mathcal{O}(\log \log d)$ time. Observe that we can indeed apply the one-dimensional case, since $|p_j - q_j| \geqslant \beta/\sqrt{d}$. This concludes the proof of Theorem 2.

We say that an edge $\{x, y\}$ *belongs* to a well-separated pair $\{A_i, B_i\}$ if and only if $x \in A_i$ and $y \in B_i$, or vice versa. Our results can be stated as follows:

Theorem 3. *Let S be a set of n points in $\mathbb{R}^d$, let F be a set of pairs of points in S having polynomially bounded aspect ratio, let T be the fair split tree for S, and let $\{A_i, B_i\}$, $1 \leqslant i \leqslant m$, be the corresponding well-separated pair decomposition of S. In $\mathcal{O}(n \log n + |F|)$ time, we can compute, for every $\{x, y\} \in F$, the index i such that $\{x, y\}$ belongs to the pair $\{A_i, B_i\}$.*

Since we have an off-line problem, we can use the approach of Section 4.2 to reduce the space requirement to $\mathcal{O}(n + |F|)$.

4.5 The General Case – Unbounded Aspect Ratio

As a result of the previous section it holds that a t-spanner can be pruned efficiently in the case when the "aspect ratio" of the edge set is polynomially bounded. To generalize this result we will use the following technical theorem that is implicit in [13].

Theorem 4. *Given a set S of n points in $\mathbb{R}^d$ and an integer constant $c \geqslant 7$ we can compute a data structure $D(S)$ in $\mathcal{O}(n \log n)$ time consisting of:*

1. *a sequence $L_1, L_2, \ldots, L_\ell$ of real numbers, where $\ell = \mathcal{O}(n)$, and*
2. *a sequence $S_1, S_2, \ldots, S_\ell$ of subsets of S such that $\sum_{i=1}^{\ell} |S_i| = \mathcal{O}(n)$,*

such that the following holds. For any two distinct points p and q of S, we can compute in $\mathcal{O}(1)$ time an index i with $1 \leqslant i \leqslant \ell$ and two points x and y in S_i such that

a. $L_i/n^{c+1} \leqslant |xy| < L_i$, and
b. both $|px|$ and $|qy|$ are less than $|xy|/n^{c-2}$.

Algorithm PRUNEGRAPH($G = (S, E), t, \varepsilon$)

Step 1: Compute the data structure $D(S)$ with $c = 7$, according to Thm. 4.
Step 2: For each $1 \leqslant i \leqslant \ell$, set $F_i := \emptyset$.
Step 3: For each edge $(p, q) \in E$, compute (i, x_p, y_q), according to Thm. 4, and add $\{x_p, y_q\}$ to F_i.
Step 4: For $i := 1$ to ℓ do
Step 4a. Compute the fair split tree T_i for S_i.
Step 4b. Compute the well-separated pair decomposition of S_i, $W_i(S_i) := \{\{A_1^i, B_1^i\}, \ldots, \{A_{m_i}^i, B_{m_i}^i\}\}$, using separation constant $2s$, where $s = ((1+\varepsilon)(8t+4)+4)/\varepsilon$.
Step 4c. For each $\{x, y\} \in F_i$, compute the pair $\{A_j^i, B_j^i\}$ that it belongs to.
Step 5: For each $1 \leqslant i \leqslant \ell$ and each $1 \leqslant j \leqslant m_i$, set $C_j^i := \emptyset$
Step 6: For each edge $(p, q) \in E$, add (p, q) to C_j^i, where j is the index such that $\{x_p, y_q\}$ belongs to $\{A_j^i, B_j^i\}$.
Step 7: Output $G' = (S, E')$, where E' contains exactly one edge from each non-empty set C_j^i.

Fig. 3. Algorithm PRUNEGRAPH

Figure 3 shows the complete algorithm, referred to as algorithm PRUNEGRAPH. Recall that the input to algorithm PRUNEGRAPH is a t-spanner $G = (S, E)$ and a positive real value ε.

Theorem 5. *Algorithm* PRUNEGRAPH *runs in* $\mathcal{O}(|E| + n \log n)$ *time and requires* $\mathcal{O}(|E|)$ *space.*

Theorem 6. *The graph* $G' = (S, E')$ *computed by algorithm* PRUNEGRAPH$(G = (S, E), t, \varepsilon)$ *is a* $(1+\varepsilon)$*-spanner of the* t*-spanner* $G = (S, E)$ *such that* $E' \subseteq E$ *and* $|E'| = \mathcal{O}(n)$.

Proof. For each $1 \leqslant i \leqslant \ell$ and each $1 \leqslant j \leqslant m_i$, consider the j-th well-separated pair $\{A_j^i, B_j^i\}$ in the i-th length partition. Let a_j^i be an arbitrary point in A_j^i and let b_j^i be an arbitrary point in B_j^i. Define $P := \{\{a_j^i, b_j^i\} : 1 \leqslant i \leqslant \ell, 1 \leqslant j \leqslant m_i\}$. First, observe that

$$|P| = \sum_{i=1}^{\ell} m_i = \mathcal{O}\left(\sum_{i=1}^{\ell} |S_i|\right) = \mathcal{O}(n).$$

We will show that the set P satisfies the premises of the general framework of Section 3. This will imply that the graph G' is obtained by pruning G with respect to P, as described in the beginning of Section 3, and, therefore, by Lemma 4, G' is a $(1+\varepsilon)$-spanner of G.

Let (p, q) be an arbitrary edge of E. By Theorem 4, there exists an index i, and two points x and y in S_i, such that $|px| < |xy|/n^5$ and $|qy| < |xy|/n^5$. By

the definition of the WSPD, there exists an index j such that (i) $x \in A_j^i$ and $y \in B_j^i$ or (ii) $x \in B_j^i$ and $y \in A_j^i$. We may assume that (i) holds.

Consider the point a_j^i in the set A_j^i and the point b_j^i in the set B_j^i. Since we chose the separation ratio for the WSPD to be $2s$, we know from Lemma 1 that $|xa_j^i| \leqslant |a_j^i b_j^i|/s$ and $|xy| \leqslant (1+2/s)|a_j^i b_j^i|$. It follows that $|pa_j^i| \leqslant |px| + |xa_j^i| \leqslant |xy|/n^5 + |a_j^i b_j^i|/s \leqslant \big((1+2/s)/n^5 + 1/s\big)\,|a_j^i b_j^i| \leqslant (2/s)|a_j^i b_j^i|$, where the last inequality assumes that n is sufficiently large. In exactly the same way, it can be shown that $|qb_j^i| \leqslant (2/s)|a_j^i b_j^i|$.

This completes the proof of the theorem. □

The above theorem can be combined with results in [10] to prove the following corollary.

Corollary 1. *Given a geometric t-spanner $G = (S, E)$ of the set S of n points in $\mathbb{R}^d$, for some constant $t > 1$, and given a positive real constant $\varepsilon' > 0$, we can compute in $\mathcal{O}(n \log n + |E|)$ time, a $(1+\varepsilon')$-spanner of G having $\mathcal{O}(n)$ edges and whose weight is $\mathcal{O}(wt(MST(S)))$.*

5 Applications

The tool presented in this paper for pruning a spanner graph is important as a preprocessing step in many situations. We briefly mention a few. In [11], the algorithm to approximate the length of the shortest path between two query points in a given geometric spanner has a preprocessing time of $\mathcal{O}(|E| \log n)$. The results here reduce the preprocessing time to $\mathcal{O}(|E| + n \log n)$. As a second application, a similar improvement can be achieved for the algorithm to compute an approximation to the stretch factor of a spanner graph [11, 16]. Using this result, an approximate stretch factor can be computed in $\mathcal{O}(|E| + n \log n)$ time, provided the stretch factor is bounded by a constant. Finally, similar improvements are achieved for several variants of the closest pair problem. In the monochromatic version, a given geometric spanner $G = (V, E)$ (with n vertices corresponding to n points in $\mathbb{R}^d$) is to be preprocessed, in order to answer closest pair queries for a query subset $S \subseteq V$ where distances between points are defined as the length of the shortest path in G. In the bichromatic version, the graph G is to be preprocessed, in order to answer closest pair queries between two given query subsets $X, Y \subseteq V$. Using this result, the preprocessing can be done in $\mathcal{O}(|E| + n \log n)$ time instead of $\mathcal{O}(|E| \log n)$.

In all the above cases, the idea is to first prune the spanner using the algorithm in this paper to obtain a spanner graph with approximately the same stretch factor, but with only a linear number of edges, consequently speeding up the previously designed algorithms.

6 Conclusions

Given a t-spanner $G = (S, E)$, where S is a set of n points in $\mathbb{R}^d$, we have shown how to compute, in $\mathcal{O}(|E| + n \log n)$ time, a $(1+\varepsilon)$-spanner of G having $\mathcal{O}(n)$

edges. The interesting fact about this result is that it shows that the pruning problem can be solved *without* sorting the edges of E. We leave open the problem of pruning a spanner in $\mathcal{O}(|E|)$ time.

References

1. I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
2. S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th ACM Symposium on Theory of Computing*, pages 489–498, 1995.
3. S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 703–712, 1994.
4. P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry: Theory and Applications*, 28:11–18, 2004.
5. P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.
6. B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *International Journal of Computational Geometry and Applications*, 5:124–144, 1995.
7. G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry and Applications*, 7:297–315, 1997.
8. G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.
9. D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers, Amsterdam, 2000.
10. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM Journal of Computing*, 31(5):1479–1500, 2002.
11. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric graph. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 828–837, 2002.
12. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles revisited. In *Proc. 13th International Symposium on Algorithms and Computation*, volume 2518 of *LNCS*, pages 357–368. Springer-Verlag, 2002.
13. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric spanners. Technical report TR-04-08, Carleton University, School of Computer Science, 2004.
14. J. M. Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop on Algorithmic Theory*, pages 208–213, 1988.
15. C. Levcopoulos, G. Narasimhan, and M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32:144–156, 2002.

16. G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal on Computing*, 30:978–989, 2000.
17. J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry & Applications*, 1:99–107, 1991.
18. M. Smid. Closest point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science Publishers, Amsterdam, 2000.
19. P. M. Vaidya. A sparse graph almost as good as the complete graph on points in K dimensions. *Discrete Computational Geometry*, 6:369–381, 1991.

The PIGs Full Monty – A Floor Show of Minimal Separators

Gerard Jennhwa Chang[1,⋆], Antonius J.J. Kloks[⋆⋆], Jiping Liu[2], and Sheng-Lung Peng[3,⋆⋆⋆]

[1] Department of Mathematics,
National Taiwan University, Taipei 10617, Taiwan
gjchang@math.ntu.edu.tw
[2] Department of Mathematics and Computer Science,
The university of Lethbridge, Alberta, T1K 3M4, Canada
liu@cs.uleth.ca
[3] Department of Computer Science and Electrical Engineering,
National Dong Hwa University, Hualien, Taiwan
lung@csie.ndhu.edu.tw

Abstract. Given a class of graphs $\mathcal{G}$, a graph G is a *probe graph of* $\mathcal{G}$ if its vertices can be partitioned into two sets $\mathbb{P}$ (the probes) and $\mathbb{N}$ (non–probes), where $\mathbb{N}$ is an independent set, such that G can be embedded into a graph of $\mathcal{G}$ by adding edges between certain vertices of $\mathbb{N}$. We show that the recognition problem of *probe interval graphs*, i.e., probe graphs of the class of interval graphs, is in P.

Classification: Algorithms and Data Structures, Algorithms for Computational Biology.

1 Introduction

Probe interval graphs, PIGs, were introduced in [17, 19, 20] to model certain problems in physical mapping of DNA. The application in molecular biology is the problem of reconstructing the arrangement of fragments of DNA taken from multiple copies of the same genome. The results of laboratory tests tell us which pairs of fragments occupy intersecting intervals of the genome. The genetic information is physically organized in a linear arrangement, and, when full information is available, an arrangement of intervals can be created in linear time [3].

More recently, a variant that makes more efficient use of laboratory resources has been studied. A subset of the fragments is designated as probes, and for each probe one may test all non–probe fragments for intersection with the probe.

⋆ Supported in part by the National Science Council under grant NSC93-2115-M002-003. Member of the Mathematics Division, National Center for Theoretical Sciences at Taipei.
⋆⋆ I am grateful to Institute[1] and Institute[2] for their hospitality and support.
⋆⋆⋆ Corresponding author.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 521–532, 2005.

In graph–theoretic terms, the input to the problem is a graph G and a subset of probe vertices. The other vertices, the non–probes, form an independent set in G. The objective is to add edges between certain non–probe vertices such that the graph becomes an interval graph. This problem has been solved successfully by Johnson and Spinrad [15] in $O(n^2)$ time (where n is the number of vertices of the graph). Soon after, using modular decomposition, McConnell and Spinrad showed that this could be improved to $O(n + m \log n)$ time (where m is the number of edges of the graph) [16].

In contrast, the problem of recognizing PIGs when the partition of the vertex set is *not* part of the input remained until now an open problem [2]. It was also mentioned as an interesting problem in [15] whether there would be a change in the complexity when we are given only the graph and must decide if there is an assignment of vertices as probes or non–probes such that the result is a probe interval graph. In this paper we give evidence that this is not the case.

Some attempts to gain a better understanding of the structure of these graphs were made by Golumbic and Lipshteyn by introducing and analyzing probe *chordal* graphs [11]. In the paper polynomial time recognition algorithms were given for the family of probe chordal graphs[1] both in the case when a fixed partition is given and when no partition is given in advance. Further results in this direction were obtained by Berry *et al.* [1, 2].

In this paper we give the first polynomial time recognition algorithm for the class of unpartitioned probe interval graphs, PIGs for short.

2 Preliminaries

The graphs in this paper are undirected and finite. A graph G is a pair $G = (V, E)$, where the elements of V are called the vertices of G and where E is a family of two-element subsets of V, called the edges. We write $n = |V|$ for the number of vertices and $m = |E|$ for the number of edges.

For two sets A and B of elements of a common universe, we write $A + B$ and $A - B$ instead of $A \cup B$ and $A \setminus B$ respectively. For a set A and an element x we write $A - x$ instead of $A \setminus \{x\}$ and $A + x$ instead of $A \cup \{x\}$.

For a graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices, we write $G[S]$ for the subgraph of G *induced* by S. For a subset $W \subseteq V$ of vertices of a graph $G = (V, E)$ we write $G - W$ for the graph $G[V - W]$, i.e., the subgraph induced by $V - W$. For a vertex x we write $G - x$ rather than $G - \{x\}$. For other conventions on graph–related notations we refer to any standard textbook. For graph classes not defined here we refer to [5, 9].

Definition 1. *A graph* $G = (V, E)$ *is an* interval graph *if there exists a one-to-one correspondence between* V *and a family* F *of intervals of the real line such that two vertices in* V *are adjacent if and only if their corresponding intervals in* F *overlap.*

[1] In [11] these graphs were called *chordal probe graphs*. We were told that this swap took place because of the abbreviation PIG.

Definition 2. *A graph* $G = (V, E)$ *is an* (unpartitioned) probe interval graph, *a* PIG, *if its vertices can be partitioned into two sets, the probes* $\mathbb{P}$ *and the non–probes* $\mathbb{N}$*, where* $\mathbb{N}$ *is an independent set, such that* G *can be embedded in an interval graph by adding edges between certain pairs of non–probes.*

This paper deals with the recognition problem of unpartitioned PIGs and in this section we review some of the background and structural properties of this graph class.

Probe interval graph were introduced by Zhang in [19]. Recently an $O(n^2)$ recognition algorithm was proposed for *partitioned* PIGs which uses PQ–trees [15]. A continuance of this research was presented in [16], and lead to an improved complexity of $O(n+m \log n)$ using modular decomposition. Until now, the recognition of PIGs when the partition is *not* part of the input was an open problem.

Theorem 1. *A probe* AT*–free graph* G *cannot have chordless cycles of length more than 6. Furthermore, every chordless 6–cycle has exactly two non–probes at distance 3 in the cycle.*

Proof. Consider a chordless cycle C. Consider the set of probes in C. There can be at most two components in $C - \mathbb{N}$, otherwise there is an AT in any embedding. Hence, if the length of C is more than 5, there must be exactly two components in $C - \mathbb{N}$. Furthermore, each component must be a single vertex of an edge, otherwise there still will be an AT in any embedding. Since the set of non–probes is independent, there must be exactly two non–probes, otherwise there are more than two components in $C - \mathbb{N}$. Hence C must have length 6, and the non–probes must be at distance 3. □

The following theorem was also shown in [10, 17].

Theorem 2. *Probe interval graphs are weakly chordal.*

Proof. Notice that PIGs are probe AT–free. Hence, according to Theorem 1, there can be no cycles of length more than 6. Also, a chord in a C_6 would leave a C_4. Golumbic and Lipschteyn showed in [11] that probe chordal graphs cannot have an induced $\overline{C}_k$ for $k \geq 5$. □

Definition 3. *Let* G *be a* PIG*. An* embedding of $G = (V, E)$ *is an interval graph* $H = (V, E')$ *together with a partition of* V *into probes* $\mathbb{P}$ *and non–probes* $\mathbb{N}$ *such that all edges of* $E' - E$ *are edges between non–probes.*

Recall, from, e.g., [5, 8, 9], that a graph is an interval graph if and only if the maximal cliques of G can be linearly ordered such that for each vertex x, the maximal cliques that contain x occur consecutively. Such an ordering of the maximal cliques is called a *consecutive clique arrangement* [8].

Definition 4. *Let* G *be a* PIG*. A* crammed embedding H *of* G *is an embedding with the minimum number of non-probes, and, among these, with the minimum number of maximal cliques. Finally, among these embeddings we take one with a minimum number of edges.*

Let $\mathcal{D} = [D_1, \ldots, D_t]$ be a consecutive clique arrangement of a crammed embedding H [5, 9]. The objective of this paper is to find such a sequence of subsets. In the rest of this paper we call two vertices *adjacent* if they are adjacent *in* G.

Recall [7] that a *splitgraph* $G = (C, S)$ is a graph with a given partition of its vertices into a clique C and an independent set S. It is easy to see that splitgraphs can be recognized in linear time [12].

Definition 5. *A* tight split *is a splitgraph such that every vertex of the independent set is adjacent to* all *vertices of the clique.*

Remark 1. In [19] these tight splits were called *quasi-cliques*. The set of probes in a quasi-clique is called the *core*. In his paper [19], Zhang calls a *maximal* quasi-clique a *quasi-maximal clique*.

We refer to an *induced* tight split M in G simply as *a tight split in* G. Notice that, unless M is a clique, the partition into an independent set and clique is unique. If M is a clique, then there can be at most *one* non–probe, since the non–probes form an independent set.

Clearly, each maximal clique D_i of H is a tight split in G. Let $C_i \subseteq D_i$ and $S_i \subseteq D_i$ be the set of probes and non–probes in D_i respectively. Then (C_i, S_i) is a tight split in G.

Lemma 1

$$\forall_{1 \le i \le t}\, C_i \neq \varnothing \ \wedge\ \forall_{1 \le i < t}\, C_i \neq C_{i+1}$$

Proof. Since the embedding is crammed. □

Remark 2. It is conceivable that, if we could list all the (maximal) induced tight splits in polynomial time, i.e., if the number of these were polynomial, that this could simplify our job. However, even for interval graphs, the number of maximal induced tight splits can be exponential as shown by a collection of $\frac{n-1}{2}$ disjoint edges and a 'central' vertex c which is adjacent to all other vertices.[2] It is easy to see that this graph is an interval graph with at least $2^{\frac{n-1}{2}} + \frac{n-1}{2}$ maximal induced tight splits.

3 The Shape of the Separators

Our main tool to recognize whether a graph G is an unpartitioned PIG are the *inclusion minimal separators* in G. In this section we analyze the structure of the minimal separators in an unpartitioned PIG. Assume throughout that G is connected.

Definition 6. *A set* $\Omega \subseteq V$ *is an* x, y-separator *for non–adjacent vertices* x *and* y *not in* Ω *if* x *and* y *are in different components of* $G - \Omega$. *An* x, y*-separator* Ω *is* a minimal x, y-separator *if no proper subset of* Ω *is an* x, y*-separator. A*

[2] Sometimes this graph is called a *windmill*.

subset Ω *is a* minimal separator *if there exist a pair of non–adjacent vertices* x *and* y *such that* Ω *is a minimal* x, y*-separator.*

Remark 3. A weakly chordal graph, and therefore also a probe interval graph, has $O(n^2)$ minimal separators and a complete list of them can be obtained in polynomial time [4].

Theorem 3. *Assume* G *is a* PIG *equipped with a crammed embedding* H*. Let* $\mathcal{D} = [D_1, \ldots, D_t]$ *be a consecutive clique arrangement for* H*. Let* Ω *be a minimal separator in* G*. Then*

1. *there exists an index* i *such that* $\Omega = D_i \cap D_{i+1}$*, or*
2. *there exist indices* $i \leq j$ *such that* $\Omega = C_i + \cdots + C_j$*, or*
3. *there exist indices* $i \leq j$ *such that* $\Omega = C_i + \cdots + C_j + (S_{i-1} \cap S_{j+1})$*.*

Proof. Assume Ω is a minimal x, y-separator for $x, y \in \mathbb{N}$. The other cases are similar. If $N(x) \subseteq N(y)$ or $N(y) \subseteq N(x)$ then the only minimal x, y-separator is either $N(x)$ or $N(y)$. (Of course, all common neighbors of x and y must be in Ω.) In this case Ω agrees with the second format.

Assume x is in consecutive subsets $D_a, \ldots, D_b$ with $a \leq b$ and y is in subsets $D_p, \ldots, D_q$ with $p \leq q$. Assume neither $N(x)$ nor $N(y)$ is contained in Ω and assume $a < p$. Let $x' \notin \Omega$ be a vertex in the *last* C_k that contains such a vertex with $a \leq k \leq \min(p, b)$ and let $y' \notin \Omega$ be a vertex in the *first* set C_ℓ that contains such a vertex with $\max(p, b) \leq \ell \leq q$. If $\ell = k+1$ then $\Omega = D_k \cap D_{k+1}$, since this is a separator that contains the common neighbors of x' and y'. In this case Ω agrees with the first format.

Let $x^* \in S_k - \Omega$, possibly $x^* = x$, be a vertex with an *inclusion maximal* neighborhood in $D_{k+1} + \cdots + D_t$, i.e., we take $x^* \in S_k - \Omega$ such that $|N(x^*) \cap (D_{k+1} + \cdots + D_t)|$ is maximal. Notice that x^* is adjacent at least to all vertices of $C_k + \cdots + C_b$, but not to y'.

Clearly, $C_i \subseteq \Omega$ for $k < i \leq b$. Also, $C_j \subseteq \Omega$ for $p \leq j < \ell$. Notice that x and x^* are in the same component of $G - \Omega$. Hence, by induction we may assume $x^* = x$.

We may assume also that there is no path from x' to any vertex in C_{b+1} outside Ω otherwise we can apply induction showing that all minimal x', y-separators are of a prescribed form. Hence $\Omega = C_{k+1} + \cdots + C_b + (S_k \cap S_{b+1})$, i.e., Ω is in agreement with the third format.

The other cases, i.e., other than $x, y \in \mathbb{N}$, are similar. □

Remark 4. The Theorem 3's Type 1 is the only minimal separator that could have no probes, i.e., is an independent set. If G is an interval graph then all minimal separators are of Type 1 and, in this case, they are cliques.

Corollary 1. *A* PIG *has* $O(n^2)$ *minimal separators. See also Remark 3.*

Definition 7. *An* inclusion minimal separator, an IMS*,* Ω *is a set of vertices such that* $G - \Omega$ *is disconnected and every vertex of* Ω *has at least one neighbor in* every *component of* $G - \Omega$*.*

Lemma 2. *Every graph* G *which is not a clique has an* IMS*, and such an* IMS *can be found in polynomial time.*

Proof. For example, consider the neighborhood of a vertex that is not adjacent to all other vertices. Repeatedly, remove vertices from this neighborhood that do not have neighbors in all components. □

Remark 5. In our algorithm we do not use Lemma 2. Instead we make a list of all the minimal separators and collect those that are inclusion minimal.

4 Arrangement of an IMS and Its Components

Assume G is a PIG equipped with a crammed embedding H and let $\mathcal{D}$ be a consecutive clique arrangement for H. Let Ω be an IMS of G. This separator can occur at a number of places in $\mathcal{D}$.

Definition 8. *An* occurrence *of* Ω *in* $\mathcal{D}$ *is a maximal interval* $i \leq j$ *such that* Ω *is of one of the types mentioned in Theorem 3.*

Remark 6. A separator Ω can occur at various places in $\mathcal{D}$. If G is an interval graph, then this can happen only with Type 1.

Notice that different occurrences of the same minimal separator Ω are disjoint and that there is at least one probe vertex $\notin \Omega$ between them. (If there is no probe vertex between 'two' occurrences of Ω then this is considered as *one* occurrence.) Clearly, if a component of $G - \Omega$ has more than one vertex then it must contain a probe vertex. Also notice that every D_i *between* two occurrences of Ω must *contain* Ω by definition of a consecutive clique arrangement.

Lemma 3. *If* Ω *occurs more than once in* $\mathcal{D}$ *then* Ω *induces a tight split.*

Proof. Since $\mathcal{D}$ is a consecutive clique arrangement of H. □

If Ω occurs *more than once*, then consider the *last* occurrence. Since the embedding is crammed we may assume that there is at least one probe to the right of this occurrence. A similar argument shows that there is at least one probe $\notin \Omega$ to the left of the *first* occurrence of Ω.

Since Ω is inclusion minimal, every vertex of Ω has at least one neighbor in every component. Hence, if a component contains only one vertex x, then it must be adjacent to all of Ω and if Ω has at least one non–probe then x must be a probe. *A fortiori*, if Ω has at least one non–probe, then every component of $G - \Omega$ must have a probe.

Theorem 4. *Assume* Ω *is not a tight split. Then* G *has at most two components that contain a probe. Hence there are at most two components with at least two vertices. If, furthermore, there are components with a single vertex then* Ω *must be of Theorem 3's Type 2 and these singleton components are non–probes.*

Proof. Assume Ω not a tight split, hence it is of Theorem 3's Type 2 or Type 3. Then it occurs only once in $\mathcal{D}$ by Lemma 3. If a component has at least two vertices then it must contain a probe p. Clearly p can appear only on one side of Ω. If there are three components with a probe then these probes are non–adjacent. Hence one of these probes, say p_1 'separates' Ω from another probe p_2. The component of p_2 must have a non–probe adjacent to a vertex of Ω, but this is impossible unless p_1 and p_2 are in one component.

Assume Ω is of Type 3 with at least one non–probe. A singleton component cannot be a probe, since it can appear only on one side of Ω and then it cannot be adjacent to all vertices of Ω. But if a component consists of a single *non–*probe, then this cannot be adjacent to any non–probe of Ω. Since Ω is inclusion minimal, this is a contradiction. □

Theorem 5. *Assume Ω is not a tight split. Then there exists a polynomial time algorithm to decide if Ω is of Theorem 3's Type 2 or Type 3. The algorithm produces the unique set of non–probes.*

Proof. By Theorem 4 we may assume that there are exactly two components in $G - \Omega$ with at least a probe. Let A be one of these two components. For a maximal clique C of Ω, let $\mathcal{N}_A(C) = \{x \in A \mid C \subseteq N(x)\}$. If there is a non–probe in Ω, then there is at least one maximal clique C such that $\mathcal{N}_A(C) = \varnothing$, namely, a maximal clique that is *not* 'the one that is closest to A'.[3]

Consider the complement $\overline{\Omega}$ of Ω. The tight split that is the intersection of all sets of Ω is a clique component (the non–probes), and a set of isolated vertices in $\overline{\Omega}$. We may try all clique components N of $\overline{\Omega}$ and choose the one such that $\mathcal{N}_A(C - N) \neq \varnothing$ for *every* clique C of Ω. It is easy to verify that N must be unique. □

5 Determination of End–Cliques

In this section we determine the tight splits at the end of an IMS Ω. In case Ω is of Type 1, we let these ends coincide with Ω itself.

Definition 9. *Let Ω be an* IMS. *For each maximal clique C of Ω and for each component A of $G - \Omega$ define $\mathcal{N}_A(C) = \{x \in A \mid C \subseteq N(x)\}$.*

Theorem 6. *Assume that for every component A and for every maximal clique C of Ω, $\mathcal{N}_A(C) \neq \varnothing$ then $\Omega \subseteq \mathbb{P}$ or Ω is contained in some (at least one) tight split set of $\mathcal{D}$.*

Proof. Let A be a component that contains a probe. Let N be the set of non-probes of Ω. If $\Omega - N$ has more than one maximal clique in some occurrence of Ω in $\mathcal{D}$, then consider a clique C which is not the closest to A. Then $N = \varnothing$, otherwise $\mathcal{N}_A(C + N) = \varnothing$. Hence, either $N = \varnothing$ or $\Omega - N$ has only one maximal clique. This maximal clique must appear somewhere in $\mathcal{D}$ together with N. □

[3] Since Ω is not a tight split, there are at least two maximal cliques.

5.1 The Thinning Procedure

Assume $\exists_A \exists_C \mathcal{N}_A(C) = \varnothing$. for some maximal clique C of Ω and some component A of $G - \Omega$. Then A has more than one vertex and Ω has at least one non–probe. Hence Ω is of Type 1 or Type 3. Consider the complement $\overline{\Omega}$ of Ω. Then the set of non-probes forms a clique component in $\overline{\Omega}$.

If Ω is a tight split then we *guess* a clique component in $\overline{\Omega}$ as a set of non–probes N.[4] Otherwise choose the unique clique component N such that $\mathcal{N}_A(C - N) \neq \varnothing$ for *every* maximal clique C in Ω. The set N is the set of non–probes in Ω and $\Omega - N$ is an interval graph.

Consider the maximal clique C_A of $\Omega - N$ such that $\mathcal{N}_A(C_A)$ is inclusion *maximal*. If there is more than one possible choice, then use another component B with at least two vertices and choose C_A such that $\mathcal{N}_B(C_A)$ is inclusion minimal.

Assume $C_A \neq C_B$. Take a vertex $x \in \mathcal{N}_A(C_B)$. Clearly x is a non-probe. Assume $N(x) \cap \mathcal{N}_A(C_A) \neq \varnothing$. Then define

$$\Delta_A = \{z \in \mathcal{N}_A(C_A) \mid N(z) \cap (\Omega - N - C_A) \neq \varnothing\}$$

Otherwise, define $\Delta_A = \mathcal{N}_A(C_A)$.

Lemma 4. $\Delta_A \subseteq \mathbb{N}$.

Proof. In the first case this is obvious. In the second case, let $D_i, \ldots, D_j$ be an occurrence of Ω and assume that A has a probe on the left side of this occurrence. If Δ_A contains a probe z, then $z + C_A$ is a clique. Hence C_A is contained in D_{i-1}. A non–probe of A that has no neighbors in $\Omega - C_A$, is not in D_i, since the embedding is crammed and, hence, the number of edges is minimal. □

Define

$$\Omega_A = C_A + N + \Delta_A$$

Define Ω_B similarly.

Assume $C_A = C_B$. If Ω has a non–probe x such that $N(x) \cap \mathcal{N}_A(C_A) \neq \varnothing$ then Ω is of type 1. Define $\Omega_A = \Omega_B = \Omega$. Otherwise, $\mathcal{N}_A(C_A)$ contains only non–probes. Define $\Omega_A = \Omega + \mathcal{N}_A(C_A)$. Similarly define Ω_B.

Assume $\forall_A \forall_C \mathcal{N}_A(C) \neq \varnothing$. By Theorem 6 either $\Omega \subseteq \mathbb{P}$ or Ω is contained in a tight split set of $\mathcal{D}$. We first guess that Ω has a non–probe, i.e., we guess a clique component in $\overline{\Omega}$. Like above determine C_A and C_B. In this case we must have $C_A = C_B$, since Ω is contained in a split set of $\mathcal{D}$. Let $\Omega_A = \Omega_B = \Omega$.

Finally assume that $\Omega \subseteq \mathbb{P}$. Then Ω is an interval graph. Determine the end–cliques C_A and C_B. If $C_A \neq C_B$, then determine Ω_A and Ω_B as above.

If $\Omega \subseteq \mathbb{P}$ is a clique then let $\Omega_A = \Omega_B = \Omega$.

[4] In this case we try all possible clique components in $\overline{\Omega}$.

6 Recognition of PIGs

In this section, we offer the following algorithm for the recognition of PIGs. We describe a procedure $\mathsf{PIG}(\Omega_1, W, \Omega_2)$ where $W \subseteq V$ is a subset of vertices and $\Omega_i \subseteq W$, $i = 1, 2$, are tight splits, possibly empty. The procedure determines whether $G[W]$ is a probe interval graph that can be embedded with Ω_1 and Ω_2 as the tight split sets or the Type 1 separators at the two ends of $\mathcal{D}$.

In the algorithm we fix two vertices at maximum distance in G. During the course of the algorithm described below these end–points are thought of as being completely connected to the two end–splits Ω_1 and Ω_2 respectively. If there is an embedding, the procedure returns the possible partitions into probes and non–probes of Ω_1 and Ω_2. The algorithm tries all possible pairs of fixed end–points until it finds an embedding or finds that the graph is not a probe interval graph.

The recursive calls in the description below are replaced by memoization, or dynamic programming. In this way every minimal separator of the graph is visited only a linear number of times. Thus the algorithm can be implemented to run in polynomial time.

Procedure $\mathsf{PIG}(\Omega_1, W, \Omega_2)$**;** Try to find a minimal separator in G that separates Ω_1 and Ω_2. Find a minimal separator Ω in G that is contained in W such that $\Omega_1 - \Omega$ is with one fixed end–point in one component of $G - \Omega$ and $\Omega_2 - \Omega$ is with the other fixed end–point in another component.[5] If there is no minimal separator for Ω_1 and Ω_2, then Ω_1 and Ω_2 must be *consecutive* separators of Theorem 3's Type 1, i.e., W is a tight split set of $\mathcal{D}$, or $W = \Omega_1 \supseteq \Omega_2$, or $W = \Omega_2 \supseteq \Omega_1$. Take all possible partitions of W and restrict these to Ω_1 and Ω_2.

Otherwise, let Ω be an IMS in W separating Ω_1 and Ω_2.

Assume $\exists_A \exists_C \mathcal{N}_A(C) = \varnothing$**.** If Ω is a tight split, then *guess* a non–empty set of non–probes, otherwise determine the unique set of non–probes. Determine C_A, C_B, Ω_A, and Ω_B.

CALL $\mathsf{PIG}(\Omega_A, A + \Omega_A, \Omega_1)$;

CALL $\mathsf{PIG}(\Omega_B, B + \Omega_B, \Omega_2)$;[6]

If $C_A \neq C_B$, then use the algorithm of [16] to test if $[\Omega_A + \Omega + \Omega_B]$ is a partitioned probe interval graph with an embedding with Ω_A on one side and Ω_B on the other.

If Ω is a tight split, then for each other component Q, the partition is induced by the non–probes of Ω. Use the algorithm of [16] to test of $[\Omega + Q + \Omega]$ is a partitioned probe interval graph and can be embedded with Ω on both sides.

Assume $\forall_A \forall_C \mathcal{N}_A(C) \neq \varnothing$**.** By Theorem 6 $\Omega \subseteq \mathbb{P}$ or Ω is contained in a tight split set of $\mathcal{D}$. Assume first that Ω has a non-probe and guess a non–probe set N. Determine $\Omega_A = \Omega_B$.

[5] It is possible that $\Omega_1 = \Omega_2$. Then $\Omega \supseteq \Omega_1$ is just any minimal separator of G contained in W.

[6] Using memoization techniques or dynamic programming these calls are replaced by a table look–up every secondary visit.

CALL $\mathsf{PIG}(\Omega_A, A + \Omega_A, \Omega_1)$;
CALL $\mathsf{PIG}(\Omega_B, B + \Omega_B, \Omega_2)$;
For other components Q use the algorithm of [16] to check if there is an embedding with Ω on both sides.
If $\Omega \subseteq \mathbb{P}$, then Ω is an interval graph. Determine the end–cliques C_A and C_B, and Ω_A and Ω_B. If $C_A \neq C_B$, then use the algorithm of [16] to test if $[\Omega_A + \Omega + \Omega_B]$ is a partitioned probe interval graph with an embedding with Ω_A on one side and Ω_B on the other.
CALL $\mathsf{PIG}(\Omega_A, A + \Omega_A, \Omega_1)$;
CALL $\mathsf{PIG}(\Omega_B, B + \Omega_B, \Omega_2)$;
The only other components are singletons which are non–probes.
If $C_A = C_B$, then $\Omega_A = \Omega_B = \Omega$, and the same recursive calls as above are made. In this case, if there are other components Q, also
CALL $\mathsf{PIG}(\Omega, Q, \Omega)$

end.

Theorem 7. *There exists a polynomial time algorithm to check if a graph is an unpartitioned* PIG*.*

Proof. A CALL $\mathsf{PIG}(\varnothing, V, \varnothing)$ does the job. Using memoization, the algorithm can be implemented such that every minimal separator of G is visited at most n times. A CALL $\mathsf{PIG}(\Omega_1, W, \Omega_2)$ returns with at most n possible partitions for Ω_1 and for Ω_2. By Remark 3, a weakly chordal graph has $O(n^2)$ minimal separators and this proves that the algorithm can be implemented to run in polynomial time. □

7 Concluding Remarks

In this paper we presented the first polynomial time recognition algorithm for unpartitioned probe interval graphs. It remains an open problem to find a more efficient recognition algorithm. For the partitioned case, we expect that a *simple linear time algorithm* is in demand *and* we expect that it is attainable. As shown in Remark 2, the number of induced tight splits could well be exponential, however, we conjecture that the number of those that are *minimal separators* is only linear. Finding those and ordering them linearly, using a PQ–tree algorithm, seems a possible approach at the present.

For the unpartitioned case, many more intriguing questions remain. One of those is to find a list of forbidden induced subgraphs (if it exists).

We started the investigation of other probe graph classes, such as (unpartitioned) probe cographs, splitgraphs, and classes with few P_4's. For these classes we were able to design polynomial time recognition algorithms. For the case of trees, Sheng gives characterizations by forbidden induced subgraphs, for both the partitioned and the unpartitioned case [10, 18]. From an algorithmic and graph–theoretical point of view it would be interesting to research the recognition problem for other classes such as probe permutation graphs.

Another line of research is taken on where the set of non–probes engages another shape than that of an independent set. Some research in this direction was done in [1], for the class of probe chordal graphs. In this paper the set of non–probes is not assumed to be a stable set.

Research into graph classes that are "close to" well-behaving classes, such as chordal graphs with small clique number or a small number of edges, or that are in some way close to interval graphs (like AT–free graphs) have proved often to be of great practical significance in a wide area of applications. This we judge is a proper justification for further study into probe graph classes.

Acknowledgment

We are grateful to Ross McConnell and Dieter Kratsch for extensive communication on the topic.

References

1. A. Berry, M. C. Golumbic, and M. Lipshteyn, Recognizing and triangulating chordal probe graphs. Research Report LIMOS/RR–03–08, 4^{th} July 2003.
2. A. Berry, M. C. Golumbic, and M. Lipshteyn, Two tricks to triangulate chordal probe graphs in polynomial time, *Proceedings* 15^{th} *ACM–SIAM Symposium on Discrete Algorithms* (2004), pp. 962–969.
3. K. S. Booth and G. S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ–tree algorithms, *Journal of Computer and System Sciences* **13** (1976), pp. 335–379.
4. V. Bouchitté and I. Todinca, Treewidth and minimum fill-in of weakly triangulated graphs, *Proceedings* 16^{th} *STACS'99*, Springer Verlag, LNCS 1563 (1999), pp. 197–206.
5. A. Brändstadt, V. Le, and J. P. Spinrad, *Graph classes: A survey* SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1999.
6. M. Chudnovsky, P. Seymour, N. Robertson, and R. Thomas, The strong perfect graph theorem. Manuscript 2002.
7. S. Földes and P. L. Hammer, Split graphs, *Congressus Numerantium* **19** (1977), pp. 311–315.
8. P. C. Gilmore and A. J. Hoffman, A characterization of comparability graphs and of interval graphs, *Canad. J. Math.* **16** (1964), pp. 539–548.
9. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
10. M. C. Golumbic and A. N. Trenk, *Tolerance graphs*, Cambridge University Press, 2003.
11. M. C. Golumbic and M. Lipshsteyn, Chordal probe graphs, *Proceedings* 29^{th} *Workshop on Graph-Theoretic Concepts in Computer Science, WG'2003*, Springer Verlag, LNCS 2880 (2003), pp. 249–260.
12. P. L. Hammer and B. Simeone, The splittance of a graph, *Combinatorica* **1** (1981), pp. 275–284.
13. R. B. Hayward, Weakly triangulated graphs, *J. Combin. Theory B* **39** (1985), pp. 200–208.

14. R. B. Hayward, C. Hoáng, and F. Maffray, Optimizing weakly triangulated graphs, *Graphs and Combinatorics* **5** (1990), pp. 33–35.
15. J. L. Johnson and J. Spinrad, A polynomial time recognition algorithm for probe interval graphs, *Proceedings* 12^{th} *ACM–SIAM Symposium on Discrete Algorithms* (2001), pp. 477–486.
16. R. M. McConnell and J. Spinrad, Construction of probe interval graphs, *Proceedings* 13^{th} *ACM–SIAM Symposium on Discrete Algorithms* (2002), pp. 866–875.
17. F.R. McMorris, C. Wang, and P. Zhang, On probe interval graphs, *Discrete Applied Mathematics* **88** (1998), pp. 315–324.
18. L. Sheng, Cycle free probe interval graphs, *Congressus Numerantium* **140** (1999), pp. 33–42.
19. P. Zhang, Probe interval graph and its application to physical mapping of DNA, Manuscript 1994.
20. P. Zhang, E. A. Schon, S. .G. Fisher, E. Cayanis, J. Weiss, S. Kistler, P. E. Bourne, An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA, *CABIOS* **10** (1994), pp. 309–317.

Centrality Measures Based on Current Flow*

Ulrik Brandes and Daniel Fleischer**

Department of Computer & Information Science, University of Konstanz
Daniel.Fleischer@uni-konstanz.de

Abstract. We consider variations of two well-known centrality measures, betweenness and closeness, with a different model of information spread. Rather than along shortest paths only, it is assumed that information spreads efficiently like an electrical current. We prove that the current-flow variant of closeness centrality is identical with another known measure, information centrality, and give improved algorithms for computing both measures exactly. Since running times and space requirements are prohibitive for large networks, we also present a randomized approximation scheme for current-flow betweenness.

1 Introduction

Centrality measures are an important tool in network analysis [6]. In social, biological, communication, and transportation networks alike, it is important to know the relative structural prominence of nodes or links to identify key elements in the network. The structure of a network is represented by a graph, so we will speak of vertices and edges in the following.

In social network analysis [22], the two most frequently used measures are vertex betweenness and vertex closeness centrality. They are based on the assumption that information (or whatever else is the content of linkages) is transmitted along shortest paths. While betweenness centrality measures the degree to which a vertex is between pairs of other vertices, i.e. on shortest paths connecting them, closeness is just the inverse of the average distance to other vertices.

A common criticism for shortest-paths based measures is that they do not take into account spread along non-shortest paths, and are thus not appropriate in cases where link content distribution is governed by other rules [4]. A betweenness measure based on network flow has been proposed in [10], and recently a variation of betweenness based on the flow of electrical current has raised considerable attention [18].

We here generalize closeness in the latter spirit and proof that the resulting measure is exactly what is already known under the name of information centrality. Despite its wide recognition, information centrality is not frequently utilized because its foundations are not very intuitive and therefore hard to understand

* Research partially supported by DFG under grant Br∼2158/1-2.
** Corresponding author.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 533–544, 2005.

by substantively oriented social scientists. Our new derivation thus provides an intuition that builds on well-known concepts and should therefore find easier reception.

Moreover, we give improved algorithms for computing current-flow based measures and describe a probabilistic approach for approximating them in large networks. The performance of the latter algorithm is evaluated on real-world and random instances.

2 Preliminaries

In this section, we recall basic definitions and facts about electrical networks (see, e.g., [3]). Throughout the paper, we only consider graphs $G = (V, E)$ that are simple, undirected, connected and have $n \geq 3$ vertices. An *electrical network* $N = (G; c)$ is such a graph together with positive edge weights $c : E \to \mathbb{R}_{>0}$ indicating the *conductance* or *strength* of an edge. Equivalently, the network can be defined in terms of positive edge weights $r : E \to \mathbb{R}_{>0}$ indicating the *resistance* or *length* of an edge, where conductance and resistance are related by $c(e) = 1/r(e)$ for all $e \in E$.

We are interested in how current flows through an electrical network. A vector $b : V \to \mathbb{R}$ called *supply* defines where current externally enters and leaves the network. A vertex $v \in V$ with $b(v) \neq 0$ is called an *outlet*; it is called a *source*, if $b(v) > 0$, and a *sink* otherwise. Since there should be as much current entering the network as leaving it, $\sum_{v \in V} b(v) = 0$ is required. Actually, we will only consider the case in which a unit current enters the network at a single source $s \in V$ and leaves it at a single sink $t \in V \setminus \{s\}$, i.e. we consider unit *st-supplies*

$$b_{st}(v) = \begin{cases} 1 & v = s, \\ -1 & v = t, \\ 0 & \text{otherwise} . \end{cases}$$

To account for the directionality of flow, each edge is given an arbitrary orientation. While the actual choice of orientation is of no importance, we denote by $\overrightarrow{e}$ the directed edge corresponding to the orientation of $e \in E$, and by $\overrightarrow{E}$ the set of all oriented edges.

Definition 1. *Let $N = (G; c)$ be an electrical network with supply b. A vector $x : \overrightarrow{E} \to \mathbb{R}$ is called* (electrical) current*, if it satisfies*

1. Kirchhoff's Current Law (KCL)

$$\sum_{(v,w) \in \overrightarrow{E}} x(v, w) - \sum_{(u,v) \in \overrightarrow{E}} x(u, v) = b(v) \qquad \text{for all } v \in V ,$$

2. Kirchhoff's Potential Law (KPL)

$$\sum_{i=1}^{k} x(\overrightarrow{e_i}) = 0 \qquad \text{for every cycle } e_1, \ldots, e_k \text{ in } G .$$

Lemma 1. *For an electrical network $N = (G; c)$ and any supply b, there is a unique current $x : \overrightarrow{E} \to \mathbb{R}$.*

A value $x(\overrightarrow{e}) > 0$ is interpreted as current flowing in the direction of $\overrightarrow{e}$, whereas $x(\overrightarrow{e}) < 0$ denotes current flowing against the direction of $\overrightarrow{e}$. For an st-supply, the corresponding current is called an st-current and denoted by x_{st}.

Currents are related to *potential differences* (or *voltages*) $\hat{p} : \overrightarrow{E} \to \mathbb{R}$ by *Ohm's Law*, $\hat{p}(\overrightarrow{e}) = x(\overrightarrow{e})/c(e)$ for all $e \in E$. A vector $p : V \to \mathbb{R}$ is said to assign *absolute potentials* if $\hat{p}(v, w) = p(v) - p(w)$ for all $(v, w) \in \overrightarrow{E}$.

Lemma 2. *Let $N = (G; c)$ be an electrical network with supply b. For any fixed vertex $v_1 \in V$ and constant $p_1 \in \mathbb{R}$, there are unique absolute potentials $p : V \to \mathbb{R}$ with $p(v_1) = p_1$.*

Again, we use $\hat{p}_{st}$ and p_{st} to indicate that the potential differences and absolute potentials are based on an st-supply. Potentials are easily computed from a given current and vice versa.

Absolute potentials can be computed directly using the *Laplacian* matrix $L = L(N)$ of $N = (G; c)$ defined by

$$L_{vw} = \begin{cases} \sum_{e\,:\,v \in e} c(e) & \text{if } v = w \\ -c(e) & \text{if } e = \{v, w\} \\ 0 & \text{otherwise} \end{cases}$$

for all $v, w \in V$. Note that the rows of L correspond to the left-hand side of KCL.

Lemma 3. *The absolute potentials of an electrical network $N = (G; c)$ with supply b are exactly the solutions of $Lp = b$.*

Since G is connected, the rank of L is $n - 1$ with a kernel spanned by $\mathbf{1} = (1, \ldots, 1)^{\mathrm{T}}$. This implies that any two assignments of absolute potentials differ only by an additive constant. Let there be a fixed vertex ordering $v_1, \ldots, v_n$ defining matrices and vectors. For brevity, we sometimes use i as an index instead of v_i. A way to choose an absolute potential is to fix, say, $p(v_1) = 0$, so that we obtain a restricted system

$$\widetilde{L}\widetilde{p} = \widetilde{b}\ ,$$

where $\widetilde{L} \in \mathbb{R}^{n-1 \times n-1}$ is the matrix obtained from L by omitting the row and column of v_1, and $\widetilde{p}$ and $\widetilde{b}$ are obtained from p and b by omitting the entry of v_1. Since $\widetilde{L}$ is positive definite, and in particular regular, we get

$$p = \underbrace{\begin{pmatrix} 0 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \widetilde{L}^{-1} \end{pmatrix}}_{=:\,C} \cdot b\ . \tag{1}$$

Matrix C will play a crucial role in computing centralities.

3 Current-Flow Measures of Centrality

Two of the most widely used centrality measures are based on a model of non-splitting information transmission along shortest paths. Note that in the following, distances may well be defined in terms of an edge length (or resistance) $r : V \to \mathbb{R}_{>0}$.

(Shortest-path) betweenness centrality [1, 9] $c_B : V \to \mathbb{R}_{\geq 0}$ is defined by

$$c_B(v) = \frac{1}{n_B} \sum_{s,t \in V} \frac{\sigma_{st}(v)}{\sigma_{s,t}}$$

where $\sigma_{s,t}$ denotes the number of shortest paths from s to t, $\sigma_{st}(v)$ denotes the number of shortest paths from s to t with v as an *inner vertex*, and $n_B = (n-1)(n-2)$ is a normalizing constant ($n_B = n(n-1)$ if v may also be a start or end vertex). It thus measures the degree to which a vertex is participating in the communication between pairs of other vertices.

(Shortest-path) closeness centrality [2] $c_C : V \to \mathbb{R}_{>0}$ is defined by

$$c_C(v) = \frac{n_C}{\sum_{t \neq v} d_G(v,t)}$$

where $d_G(v,w)$ denotes the length of a shortest path between v and w and $n_C = n - 1$ is a normalizing constant. It thus measures the degree to which a vertex is close to other vertices (on average).

Both measures assume that information (or whatever else is being modeled) flows along shortest paths, and does not split. We next describe two alternative measures that build on the same intuition, but let information flow and split like current in an electrical network.

3.1 Current-Flow Betweenness Centrality

In electrical networks, the analog of the fraction of shortest st-paths passing through a vertex (or an edge) is the fraction of a unit st-current flowing through that vertex (or edge). Given a supply b, we therefore define the *throughput* of a vertex $v \in V$ to be

$$\tau(v) = \frac{1}{2} \left(-|b(v)| + \sum_{e:v \in e} |x(\overrightarrow{e})| \right) ,$$

where the term $-|b(v)|$ accounts for the fact that only inner vertices are considered in the definition of shortest-path betweenness centrality. To include start and end vertex, it should be replaced by $+|b(v)|$. Accordingly, the throughput of an edge $e \in E$ is defined as

$$\tau(e) = |x(\overrightarrow{e})| .$$

Let τ_{st} denote the throughput in case of an st-current.

Definition 2 ([18]). *Let $N = (G; c)$ be an electrical network.* Current-flow betweenness centrality $c_{CB} : V \to \mathbb{R}_{\geq 0}$ *is defined by*

$$c_{CB}(v) = \frac{1}{n_B} \sum_{s,t \in V} \tau_{st}(v) \qquad \text{for all } v \in V \ ,$$

where $n_B = (n-1)(n-2)$.

Current-flow betweenness is well-defined because of Lemma 1. For the following reason, it is also called *random-walk betweenness*. A simple random st-walk is a random walk that starts at s, ends in t and continues at vertex $v \neq t$ by picking an incident edge $e \in E$ with probability $c(e)/\sum_{e':v \in e'} c(e')$. Then, given an st-current, the amount of current flowing through a particular edge $\overrightarrow{e}$ equals the expected difference of the number of times that the simple random st-walk passes edge $\overrightarrow{e}$ along and against its orientation (see, e.g., [3]).

3.2 Current-Flow Closeness Centrality

Similar to the above variation of betweenness centrality, we utilize the analog of shortest-path distance in electrical networks to introduce a variant of closeness centrality.

Definition 3. *Let $N = (G; c)$ be an electrical network.* Current-flow closeness centrality $c_{CC} : V \to \mathbb{R}_{>0}$ *is defined by*

$$c_{CC}(s) = \frac{n_C}{\sum\limits_{t \neq s} p_{st}(s) - p_{st}(t)} \qquad \text{for all } s \in V \ .$$

Current-flow closeness centrality is well-defined, because by Lemma 2 any two absolute potentials differ only by an additive constant. Since we only consider unit st-currents, the term $p_{st}(s) - p_{st}(t)$ corresponds to the *effective resistance*, which can be interpreted as an alternative measure of distance between s and t.

Though not derived in the same fashion, it turns out that current-flow closeness has actually been considered before. *Information centrality* $c_I : V \to \mathbb{R}_{>0}$ is defined by

$$c_I(s)^{-1} = nC^I_{ss} + \mathrm{trace}(C^I) - \frac{2}{n} \ , \tag{2}$$

where $C^I = (L+J)^{-1}$ with Laplacian L and $J = \mathbf{1}\mathbf{1}^{\mathrm{T}}$ [20]. Information centrality is often referred to, but not frequently used; most likely because its underlying intuition is not widely understood.

Theorem 1. *Current-flow closeness centrality equals information centrality.*

Proof. We first note that Eq. (2) can be rewritten in terms of matrix elements only,

$$c_I(s)^{-1} = \sum_{t \in V} C^I_{ss} + C^I_{tt} - C^I_{st} - C^I_{ts} \ . \tag{3}$$

On the other hand, current-flow closeness can be rewritten into the same form, though with matrix C introduced in Eq. (1),

$$\begin{aligned} c_{CC}(s)^{-1} &= \sum_{t \neq s} p_{st}(s) - p_{st}(t) \;=\; \sum_{t \neq s} C_{ss} - C_{st} - (C_{ts} - C_{tt}) \\ &= \sum_{t \in V} C_{ss} + C_{tt} - C_{st} - C_{ts} \;. \end{aligned}$$

We show that these terms are actually equal using a matrix D with $C^I = C + D$ that contributes zero to the common summation scheme.

For $i = 1, \ldots, n$ let $\boldsymbol{e_i} = (0, \ldots, 0, 1, 0, \ldots, 0)^{\mathrm{T}}$, with 1 in the ith position, and let $C_{\bullet i}$ and $C^I_{\bullet i}$ denote the ith column of the corresponding matrix. The columns of C are uniquely determined by

$$LC_{\bullet i} = \boldsymbol{e_i} - \boldsymbol{e_1} \quad \text{and} \quad C_{1i} = 0$$

and those of C^I satisfy

$$(L + J)\, C^I_{\bullet i} = \boldsymbol{e_i} \tag{4}$$

by definition. Projecting Eq. (4) onto the kernel of L, i.e. multiplying both sides with $\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ from the left, yields

$$JC^I_{\bullet i} = (\mathbf{1}^{\mathrm{T}} C^I_{\bullet i})\, \mathbf{1} = \frac{1}{n}\mathbf{1} \;.$$

Eq. (4) is therefore equivalent to

$$LC^I_{\bullet i} = \boldsymbol{e_i} - \frac{1}{n}\mathbf{1} \quad \text{and} \quad \mathbf{1}^{\mathrm{T}} C^I_{\bullet i} = \frac{1}{n} \;.$$

Now let q be a vector with $Lq = L(C^I_{\bullet i} - C_{\bullet i}) = \boldsymbol{e_1} - \frac{1}{n}\mathbf{1}$. Then we have $C^I_{\bullet i} = C_{\bullet i} + q + d_i \mathbf{1}$ for some constants d_i (choosing q such that $q_1 = 0$ yields $d_i = C^I_{1i}$). In matrix notation we thus obtain $C^I = C + D$ with

$$D = \begin{pmatrix} q_1 + d_1 & q_1 + d_2 & \cdots \\ q_2 + d_1 & q_2 + d_2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \;.$$

It is easily verified that D contributes zero when subjected to the summation scheme of (3). □

4 Improved Exact Computation

For comparison note that shortest-path betweenness and closeness can be computed in $\mathcal{O}(nm + n^2 \log n)$ time and $\mathcal{O}(n + m)$ space using an efficient implementation of Dijkstra's algorithm [5].

For current-flow betweenness centrality, matrix C defined in Eq. (1) is determined by inverting the reduced Laplacian. Since $p_{st} = Cb_{st}$ and $x(v,w) = (p(v) - p(w)) \cdot c(\{v,w\})$, we can use the incidence matrix $B = B(N) \in \mathbb{R}^{n \times m}$, defined by

$$B_{ve} = \begin{cases} c(e) & \text{if } \overrightarrow{e} = (v,w) \text{ for some } w \\ -c(e) & \text{if } \overrightarrow{e} = (u,v) \text{ for some } u \\ 0 & \text{otherwise} , \end{cases}$$

to compute st-currents $x_{st} = BCb_{st}$. From the entries of *current-flow matrix* $F = BC$ the centrality scores are then determined via

$$\begin{aligned} c_{CB}(v) &= \frac{1}{n_B} \sum_{s,t \in V} \tau_{st}(v) \\ &= \frac{1}{n_B} \sum_{s,t \in V} \frac{1}{2} \left(-|b_{st}(v)| + \sum_{e\,:\,v \in e} |x_{st}(\overrightarrow{e})| \right) \\ &= \frac{1}{2-n} + \frac{1}{n_B} \sum_{s,t \in V} \sum_{e\,:\,v \in e} \frac{1}{2} |F_{es} - F_{et}| \\ &= \frac{1}{2-n} + \frac{1}{n_B} \sum_{s<t \in V} \sum_{e\,:\,v \in e} |F_{es} - F_{et}| , \end{aligned}$$

where $v_i < v_j$ if and only if $i < j$ (recall that we assume a fixed vertex ordering). The total time to compute current-flow betweenness is thus in $\mathcal{O}(I(n-1) + mn^2)$ [18], where $I(n) \in \mathcal{O}(n^3)$ is the time required to compute the inverse of an $n \times n$-matrix. Note that $I(n) \in \Omega(n^2 \log n)$ for arbitrary real matrices.

This can be improved as follows (see Alg. 1).

Theorem 2. *Current-flow betweenness can be computed in* $\mathcal{O}(I(n-1)+mn \log n)$ *time.*

Proof. We refer to Alg. 1. We can compute $c_{CB}(v)$ by summing up only the *inflows*, i.e. positive current on an edge directed to v or negative current on an edge leaving v, as follows. Note that for every non-outlet the inflow is equal to the outflow by KCL. We will later take care of the outlets. The *total inflow* τ_{in} into v equals

$$\begin{aligned} \tau_{\text{in}}(v) &= \frac{1}{n_B} \sum_{(v,w) \in \overrightarrow{E}} \sum_{\substack{s<t: \\ F_{es} < F_{et}}} |F_{es} - F_{et}| + \frac{1}{n_B} \sum_{(w,v) \in \overrightarrow{E}} \sum_{\substack{s<t: \\ F_{es} > F_{et}}} |F_{es} - F_{et}| \\ &= \frac{1}{n_B} \sum_{(v,w) \in \overrightarrow{E}} \sum_{i=1}^{n} (i - \text{pos}(\{v,w\}, v_i)) \cdot F_{ev_i} \\ &\quad + \frac{1}{n_B} \sum_{(w,v) \in \overrightarrow{E}} \sum_{i=1}^{n} (n + 1 - i - \text{pos}(\{w,v\}, v_i)) \cdot F_{ev_i} . \end{aligned}$$

Algorithm 1: Current-flow betweenness

Input: electrical network $N = (G; c)$ with vertices $v_1, \ldots, v_n$
Output: current-flow betweenness $c_{CB} : V \to \mathbb{R}_{\geq 0}$

begin
  $c_{CB} \leftarrow \mathbf{0}$
  $C \leftarrow \begin{pmatrix} 0 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \widetilde{L}^{-1} \end{pmatrix}$
  for $e \in E$ **do**
    $F_{e\bullet} \leftarrow (BC)_{e\bullet}$
1.1 sort row $F_{e\bullet}$ in non-increasing order
    $\mathrm{pos}(e, v) \leftarrow$ rank of F_{ev} in sorted row $F_{e\bullet}$
    for $i = 1, \ldots, n$ **do**
      increase $c_{CB}(\mathrm{source}(\overrightarrow{e}))$ by $(i - \mathrm{pos}(e, v_i)) \cdot F_{ev_i}$
      increase $c_{CB}(\mathrm{target}(\overrightarrow{e}))$ by $(n + 1 - i - \mathrm{pos}(e, v_i)) \cdot F_{ev_i}$
  for $i = 1, \ldots, n$ **do**
1.2 $c_{CB}(v_i) \leftarrow (c_{CB}(v_i) - i + 1) \cdot 2/n_B$
end

Inflows include the vanishing unit current whenever v is the sink. In the summation over all pairs $s < t$ this will be the case $i-1$ times, namely when $v = v_i$. Note that the inflow of the source is always zero. Subtracting the vanishing currents from the total inflow yields half of the current-flow betweenness. The relation

$$c_{CB}(v_i) = 2(\tau_{\mathrm{in}}(v_i) - i + 1) \ ,$$

is accounted for in Line 1.2 of the algorithm.

The computational bottleneck after determining C by matrix inversion is the sorting of rows in Line 1.1, which takes $\mathcal{O}(mn \log n)$ time. Note that F is computed by multiplying C with an incidence matrix, so that it takes only $\mathcal{O}(mn)$ time. □

Information centrality can be computed by determining matrix C^I defined in the previous section and evaluating Eq. (2) [20]. The total running time is thus $\mathcal{O}(I(n) + n)$.

Using the new interpretation as current-flow closeness centrality, we see that it can also be determined from C rather than C^I (see Alg. 2). Thus sparseness is preserved and only one matrix inversion is required to compute both closeness and betweenness, which corresponds nicely to the fact that shortest-path betweenness and closeness can be computed during the same traversals.

A straightforward approach for matrix inversion uses Gaussian elimination, leading to a computation time of $\mathcal{O}(n^3)$. For networks from many application areas, however, sparse matrix techniques are appropriate. Since $\widetilde{L}$ is symmetric and positive definite, the conjugate gradient method (CGM) can be used with an incomplete LU-decomposition as a preconditioner. This yields a running time of

$\mathcal{O}(mn\sqrt{\kappa})$, where κ is the condition number of $\widetilde{L}$ times its approximate inverse obtained by applying the preconditioner. A rough estimate for the condition number is $\kappa \in \Theta(n)$ [12], leading to a running time of $\mathcal{O}(mn^{1.5})$ which is faster than the subsequent summation before its improvement to $\mathcal{O}(mn \log n)$.

The inverse of $\widetilde{L}$ can be computed column-by-column as needed in Line 2.1 of the algorithm for closeness centrality. Its memory requirement is in $\mathcal{O}(m)$.

For betweenness centrality, $\mathcal{O}(n^2)$ memory is required in the worst case. Here it is the current-flow matrix F that is processed row-by-row, implying that columns of $\widetilde{L}^{-1}$ corresponding to vertices u and w with $\{u, w\} \in E$ are needed simultaneously. Therefore, the column $v \in V$ needs to be determined only when the first row $F_{e\bullet}$ with $v \in e$ is encountered, and it can be dropped from memory when the last such row has been processed.

To reduce the memory requirements of Alg. 1, we therefore seek an ordering that minimizes the maximal number of columns that have to be kept in memory at the same time. That is, we would like to determine a one-to-one mapping $\pi : V \to \{1, \ldots, n\}$ where

$$\delta(\pi) = \max_{1 \le i \le n} |\{u \in V \,:\, \exists\, w \in V, \{uw\} \in E \;\text{ with } \pi(u) \le i < \pi(w)\}| \le n$$

is minimum. Unfortunately, this is an $\mathcal{NP}$-hard problem known as *vertex separation* [17], or, equivalently [15], *minimum pathwidth*.

Heuristically, we can find a good ordering π^* by using algorithms for bandwidth- and envelope-reduction of matrices, since the bandwidth (of the Laplacian matrix of N ordered by π^*) is an upper bound for $\delta(\pi)$. Algorithm 1 is easily modified to use any precomputed ordering. The proven reverse Cuthill-McKee heuristic [7]) does not increase the asymptotic running time, while it reduces the memory requirement to $\mathcal{O}(\delta(\pi^*)n)$. Note that it can also be employed in the inversion of $\widetilde{L}$.

Algorithm 2: Current-flow closeness

Input: electrical network $N = (G; c)$
Output: current-flow closeness $c_{CC} : V \to \mathbb{R}_{>0}$

```
begin
    c_CC ← 0
    for v ∈ V do
2.1     C_•v ← ( 0  0^T ; 0  L̃^{-1} )_•v
        for w ∈ V do
            increase c_CC(v) by C_vv − 2C_wv
            increase c_CC(w) by C_vv
    for v ∈ V do
        c_CC(v) ← 1/c_CC(v)
end
```

5 Probabilistic Approximation

In large networks, both running time and space requirements of the algorithm for current-flow betweenness are prohibitive. Note that shortest-path closeness can be approximated quickly [21].

We show that a similar approach can be used to reduce not only the running time, but also the space requirements of (approximate) current-flow betweenness computations. For large data sets, this is often even more important.

The basic idea is that the betweenness of a vertex, i.e. the throughput over all *st*-currents, can be approximated using a small fraction of all pairs $s \neq t \in V$. A fully polynomial randomized approximation scheme is given in Alg. 3.

Theorem 3. *There is a randomized algorithm that, in* $\mathcal{O}(\frac{1}{\varepsilon^2} m\sqrt{\kappa}\log n)$ *time and* $\mathcal{O}(m)$ *space, approximates current-flow betweenness to within an absolute error of* ε *with high probability.*

Proof. Let $X_v^{(1)}, \ldots, X_v^{(k)}$ be independent random variables that return $\tau_{st}(v)$, for a pair $s \neq t \in V$, picked uniformly at random. With $c^* = n(n-1)/n_B$,

$$E\left(\frac{c^*}{k}\sum_{i=1}^{k} X_v^{(i)}\right) = c^* E(X_v^{(1)}) = \frac{1}{n_B}\sum_{s\in V}\sum_{t\neq s}\tau_{st}(v) = c_{CB}(v)\ ,$$

i.e. the scaled expected throughput of k *st*-currents is equal to the current-flow betweenness. Since $0 \leq \tau_{st}(v) \leq 1$, Hoeffding's bound [14] gives

$$\mathbb{P}\left(\left|\frac{c^*}{k}\sum_{i=1}^{k} X_i^v - c_{CB}(v)\right| \geq \varepsilon\right) \leq 2\exp\left(-2(\varepsilon/c^*)^2 k\right) \leq \frac{2}{n^{2\ell}}$$

when choosing $k = \ell \cdot \lceil (c^*/\varepsilon)^2 \log n \rceil$ pairs for arbitrary ℓ.

For each selected pair $s \neq t \in V$, the restricted system in Line 3.1 of Alg. 3 can be solved in $\mathcal{O}(m\sqrt{\kappa})$ time and $\mathcal{O}(m)$ space using CGM. □

Algorithm 3: Randomized approximation scheme for current-flow betweenness

Input: electrical network $N = (G; c)$, threshold $\varepsilon > 0$, constant ℓ
Output: current-flow betweenness approximation $c'_{CB} : V \to \mathbb{R}_{\geq 0}$

begin
  $c'_{CB} \leftarrow \mathbf{0}$ and $k \leftarrow \ell \cdot \lceil (c^*/\varepsilon)^2 \log n \rceil$
  for *i=1,... k* **do**
    select $s \neq t \in V$ uniformly at random and solve $\widetilde{L}\widetilde{p} = \widetilde{b}_{st}$
3.1 **for** $v \in V \setminus \{s, t\}$ **do**
      for $e = \{v, w\} \in E$ **do** increase $c'_{CB}(v)$ by $c(e) \cdot |\widetilde{p}(v) - \widetilde{p}(w)| \cdot c^*/2k$
end

6 Discussion

Current-flow betweenness and closeness are variants of (shortest-path) betweenness and closeness centrality for an alternative model of information spreading. In particular, we introduced current-flow closeness and proved that it is equal to information centrality, the original definition of which is rather unintuitive.

There is one and only one path between each pair of vertices in a tree, and the length of this path equals its resistance. We thus have the following result.

Theorem 4. *The two shortest-path and current-flow measures agree on trees.*

Corollary 1. *Betweenness and closeness can be computed in $\mathcal{O}(n)$ time and space on trees.*

Proof. A bottom-up followed by a top-down traversal similar to [19]. □

Finally, we want to remark that there is a straightforward extension of shortest-path betweenness to edges (simply replace the numerators by the number of shortest st-paths that use the edge) [1]. A similar extension of current-flow betweenness, that can be computed by slight modification of Alg. 1, is given by

$$c_{CB}(e) = \frac{1}{n_B} \sum_{s \neq t \in V} \tau_{st}(e) \qquad \text{for all } e \in E \ .$$

References

1. Anthonisse, J.M.: The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, Amsterdam (1971)
2. Beauchamp, M.A.: An improved index of centrality. Behavioral Science **10** (1965) 161–163
3. Bollobás, B.: Modern Graph Theory. Springer (1998)
4. Borgatti, S.P.: Centrality and Network Flow. Social Networks (to appear)
5. Brandes, U.: A faster algorithm for betweenness centrality. Journal of Mathematical Sociology **25** (2001) 163–177
6. Brandes, U., Erlebach, T., Eds.: Network Analysis. Springer LNCS (to appear)
7. Cuthill, E.H., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. Proceedings of the 24th ACM National Conference (1969) 157–172
8. Diaz, J., Petit, J., Serna, M.: A Survey of Graph Layout Problems. ACM Comput. Surv. **34** (2002) 313–356
9. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry **40** (1977) 35–41
10. Freeman, L.C., Borgatti, S.P., White, D.R.: Centrality in valued graphs: A measure of betweenness based on network flow. Social Networks **13** (1991) 141–154
11. Godsil, C., Royle, G.: Algebraic Graph Theory. Springer (2001)
12. Golub, G.H., van Loan, C.F.: Matrix Computations. Johns Hopkins University Press (1983)
13. Hackbusch, W.: Iterative Solution of Large Sparse Systems of Equations. Springer (1994)

14. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58** (1963) 13-30
15. Kinnersley, N.G.: The vertex separation number of a graph equals its path width. Information Processing Letters **42** (1992) 345–350
16. Kirchhoff, G.: Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der Linearen Vertheilung galvanischer Ströme geführt wird. Ann. Phys. Chem. **72** (1847) 497–508
17. Lengauer, T.: Black-white pebbles and graph separation. Acta Informatica **16** (1981) 465–475
18. Newman, M.E.J.: A Measure of betweenness centrality based on random walks. http://arxiv.org/abs/cond-mat/0309045 (2003)
19. Rosenthal, A., Pino, J.A.: A generalized algorithm for centrality problems on trees. Journal of the ACM **36** (1989) 349–381
20. Stephenson, K.A., Zelen, M.: Rethinking centrality: methods and examples. Social Networks **11** (1989) 1–37
21. Wang, J., Eppstein, D.: Fast approximation of centrality. Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (2001) 228–229
22. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press (1994)

A Experimental Evaluation

We provide empirical evidence that the proposed algorithms for (approximate) computation of current-flow betweenness is practical. It has been implemented in Java using the yFiles[1] graph data structure and the JMP[2] linear algebra package. All experiments were performed on a regular PC with 2.4 GHz clock-speed and 3 GB main memory. Constant $\ell = 1$ for the approximation. See Fig. 1.

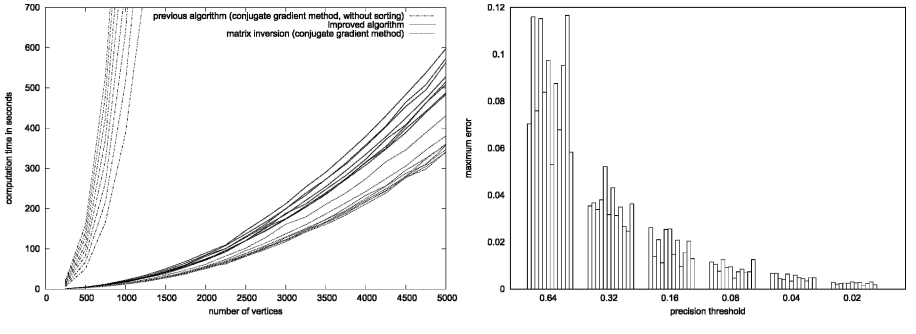


Fig. 1. Comparison of total running time for current-flow betweenness on random graphs with average degree $6, 8, \ldots, 20$ and maximum error of approximation on 6 random and 6 AS graphs with approximately 9000 vertices and 20000 edges each

[1] www.yworks.de
[2] www.math.uib.no/~bjornoh/jmp/index2.html

Varieties of Codes and Kraft Inequality*

Fabio Burderi and Antonio Restivo

Dipartimento di Matematica ed Applicazioni,
Università degli studi di Palermo,
Via Archirafi 34, 90123 Palermo, Italy
{Burderi, Restivo}@math.unipa.it

Abstract. Decipherability conditions for codes are investigated by using the approach of Guzmán, who introduced in [7] the notion of variety of codes and established a connection between classes of codes and varieties of monoids. The class of Uniquely Decipherable (UD) codes is a special case of variety of codes, corresponding to the variety of all monoids.

It is well known that the Kraft inequality is a necessary condition for UD codes, but it is not sufficient, in the sense that there exist codes that are not UD and that satisfy the Kraft inequality. The main result of the present paper states that, given a variety $\mathcal{V}$ of codes, if all the elements of $\mathcal{V}$ satisfy the Kraft inequality, then $\mathcal{V}$ is the variety of UD codes. Thus, in terms of varieties, Kraft inequality characterizes UD codes.

1 Introduction

The theory of *uniquely decipherable* (UD) codes, born in the context of information theory, has been later developed, as an independent subject, using both combinatorial and algebraic methods. This theory is now a part of theoretical computer science and is strongly related to combinatorics on words, automata theory and formal languages (cf [1], and also [6], [10]).

UD codes are formal languages with special combinatorial properties, which are exploited in information processing. A set C of words is a UD code if any word over the same alphabet admits at most one factorization in elements of C. Hence, when a source message is encoded using a UD code, the deciphering process will recover the original message in its entirety.

Unique decipherability imposes some quantitative constraints on the size of code words, settled by the famous *Kraft inequality*, that keeps the words of a UD code from getting too short. This inequality is at the basis of well known limitations on the efficiency of the encoding process. Remark that Kraft inequality is, in a strict sense, a necessary but not sufficient condition for unique decipherability. Indeed there exist sets of words that satisfy Kraft inequality and that are not UD codes.

* Partially supported by Italian MURST Project of National Relevance "Linguaggi Formali e Automi: Metodi, Modelli e Applicazioni".

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 545–556, 2005.

Motivated by special problems in information transmission, decipherability conditions weaker than UD have been introduced. Indeed, there are situations when it is not desirable or necessary to recover the whole original message, but only part of the information embedded in it. In these cases something weaker than unique decipherability is the appropriate property for a code to have. One of these weaker concepts is that of *multiset decipherability*, MSD for short, introduced by Lempel in [9]. As stressed by Lempel *"in certain applications it is desired to communicate a description of a sequence of events where the information of interest is which of the set of possible events have occurred, including multiplicity, but where the order of occurrence is irrelevant"*. MSD codes are such that, given a finite message, every possible parsing of the message into code words must yield the same multiset of code words. In [7] Guzmán introduced the notion of a *set decipherable* (SD) code. In set decipherability, it is the set of code words that is the relevant information, so the order and the multiplicity of words are immaterial. The decidability of multiset decipherability and set decipherability, and other related properties, have been studied in [8] and [2].

A very general approach to the study of decipherability conditions weaker than UD has been initiated by Guzmán in [7], introducing the notion of *variety* of codes. The approach is based on the concept of decipherability of a code in a monoid and then on a correspondence between varieties of monoids and some classes of codes, called varieties of codes. The classes of UD, MSD and SD codes appears respectively, in such approach, as very special cases of varieties of codes. Moreover Guzmán in [7] shows that there exists an infinite hierarchy of varieties of codes different from those of UD, MSD and SD codes, respectively.

Although the study of decipherability conditions weaker than UD originates in concrete problems of information transmission, it gives rise to several interesting questions in the areas of combinatorics on words, formal languages and the theory of semigroups.

In this paper we investigate the quantitative constraints that the new decipherability conditions impose on the length of code words. In particular we approach the problem to characterize those varieties of codes where the Kraft inequality is satisfied. Lempel [9] firstly posed the question whether there exist MSD codes whose Kraft sum exceeds unity. A positive answer was given in [11]. The resulting shorter average length of such codes is then a welcome trade-off for the relaxed decipherability conditions. The problem is here approached for an arbitrary variety of codes. The main result of the paper states, generalizing [11], that, *given a variety $\mathcal{V}$ of codes, if all the codes in $\mathcal{V}$ satisfy the Kraft inequality, then $\mathcal{V}$ is the variety of UD codes.* Thus we can assert that, in terms of varieties, Kraft inequality characterizes UD codes. From the point of view of information theory, the result leaves open the possibility that there may exist situations in which codes, that are not UD, can provide greater efficiency, in terms of word lengths, than UD codes.

In order to prove the main theorem, we develop some complementary results concerning codes which are *maximal in a variety*. We extend to varieties some well known relationships, between maximality and completeness, that hold for

UD codes. In particular, we derive that, given two varieties $\mathcal{V}_1$ and $\mathcal{V}_2$ of codes, such that $\mathcal{V}_1 \subseteq \mathcal{V}_2 \subseteq SD$, if a code C is maximal in $\mathcal{V}_1$, then it is also maximal in $\mathcal{V}_2$. This extends a result of [11], originally conjectured by Lempel [9], asserting that no MSD code contains a maximal UD code as a proper subcode.

An important tool in the proof of our main result is a construction, introduced by Ehrenfeucht and Rozenberg in [5] (cf. also [1]), for embedding a regular UD code in a complete and regular UD code. Here we extend such a construction to an arbitrary variety of codes.

The paper is organized as follows. In Section 2 we report some basic definitions and results on UD codes, that are useful in the sequel. The notion of a variety of codes is introduced in Section 3. The relationships between maximality and completeness in a variety of codes, and the extension of the construction of Ehrenfeucht and Rozenberg to an arbitrary variety of codes are given in Section 4. Section 5 contains the proof of the main result. Some open problems are finally posed in the last section of the paper.

2 Uniquely Decipherable Codes

Let A be a finite alphabet. Let A^* denote the free monoid generated by A, i.e. the set of words over the alphabet A, and let $A^+ = A^* \backslash \{\varepsilon\}$.

A code C over A is a subset of A^+. The words of C are called *code words*, the elements of C^* *messages*, where C^* denotes the submonoid of A^* generated by C, i.e. the set of words obtained concatenating elements of C.

A code C is said to be *uniquely decipherable* (UD) if every message has an unique factorization into codewords, i.e. the equality

$$x_1 x_2 \cdots x_n = y_1 y_2 \cdots y_m,$$

$x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m \in C$, implies $n = m$ and $x_1 = y_1, \ldots, x_n = y_n$.

The theory of UD codes has been widely developed, and it is closely related also to problems in automata theory, combinatorics on words, formal languages and semigroup theory. A complete treatment of such theory can be found in [1].

The notions of *maximality* and *completeness* play an important role in this theory. A UD code over the alphabet A is *maximal (full)* if it is not a proper subset of another UD code over A. In order to introduce the notion of completeness we need the following definitions. If u and v are words of A^*, we say that u is *factor* of v if there exist words $s, t \in A^*$ such that $v = sut$. Let $F(v)$ denote the set of factors of the word v, and for any subset $L \subseteq A^*$, let $F(L)$ denote the set of factors of words in L. A subset C of A^* is *complete* if any word of A^* is factor of some word of C^*. In our notation, C is complete if and only if $F(C^*) = A^*$.

Recall that a subset $L \subseteq A^*$ is *regular* if it is recognized by a finite state automaton. The following result of Schutzenberger (cf [1]) relates completeness and maximality of UD codes.

Theorem 1. *A maximal UD code is complete. Conversely a regular and complete UD code is maximal.*

For any code C over A, with $card(A) = d$, the *Kraft sum* for C is given by

$$\mathcal{K}(C) = \sum_{c \in C} d^{-|c|}$$

where $|c|$ is the length of the word c.

It is well known (cf [1]) that every UD code C satisfies

$$\mathcal{K}(C) \leq 1.$$

This is known as the *Kraft Inequality* and it keeps the words of a UD code from getting too short.

Remark 1. Let us remark that Kraft Inequality provides only a necessary condition for unique decipherability. However it is not sufficient, as shown by very simple examples. Consider, for instance the code $C = \{ab, ba, a\}$. C is not UD. Indeed the message aba has two factorizations in elements of C: $(a)(ba)$ and $(ab)(a)$. However $\mathcal{K}(C) = 1$.

The Kraft sum is also related to the completeness of the code, as shown by the following theorem (cf [1]).

Theorem 2. *If C is a regular and complete code, then $\mathcal{K}(C) \geq 1$.*

Next theorem ([3], cf also [1]) provides a stronger relationship between unique decipherability, completeness and the Kraft sum.

Theorem 3. *Let C be a regular code. Any two among the following three conditions imply the third:*

- *C is UD*
- *$\mathcal{K}(C) = 1$*
- *C is complete*

3 Varieties of Codes

When a source message is encoded using a UD code, the deciphering process will recover the original message in its entirety. There are situations, however, when it is not necessary to recover the whole message but only part of the original information embedded in it. In these cases, something weaker than unique decipherability is the appropriate property for a code to have.

The investigation on decipherability conditions weaker than UD was initiated in [9] by Lempel, who introduced the notion of *multiset decipherable* (MSD) codes. Here the information of interest is the multiset of codewords used in the

encoding process so that the order in which transmitted words are received is immaterial. In a more formal way, a code C is a MSD code if the equality

$$x_1x_2\cdots x_n = y_1y_2\cdots y_m,$$

$x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m \in C$, implies the equality of the two multisets $\{x_1, x_2, \ldots, x_n\}$ and $\{y_1, y_2, \ldots, y_m\}$.

In [7] Guzmán considers also the notion of *set decipherable* (SD) codes. In this case the original message is recovered up to commutativity and actual count of occurrences, i.e. two factorizations of the same message yield the same *set* of codewords. Denote by $\mathcal{UD}$, $\mathcal{MSD}$ and $\mathcal{SD}$ the classes of UD, MSD and SD codes, respectively. It is clear that

$$\mathcal{UD} \subseteq \mathcal{MSD} \subseteq \mathcal{SD}$$

and has been shown that the two inclusions are strict. The code

$$C_1 = \{110, 101, 11011, 01110101\}$$

shows that the first inclusion is strict. In fact the message

$$(110)(11011)(101)(01110101) = (11011)(01110101)(110)(101)$$

has two distinct factorizations into codewords and in [9] is shown to be MSD. The code

$$C_2 = \{0, 010, 11011, 101101\}$$

shows the strictness of the second inclusion. In fact the message

$$(0)(101101)(101101)(11011)(010) = (010)(11011)(0)(11011)(101101)(0)$$

has two distinct factorizations with the same set of codewords, but distinct multisets of codewords and in [7] is shown to be SD.

In the same paper [7] Guzmán introduces a very general concept of decipherability using *varieties* of monoids. Unique decipherability, multiset decipherability and set decipherability then appear as very special cases of such general concept.

The intuitive idea in introducing decipherability conditions weaker than UD is the following: since, in certain coding applications, distinct sequences of words can carry the "same" information, one has to specify a set of possible identities such that the information of interest is invariant with respect to transformations induced by such identities. This idea can be formalized by introducing the notion of *decipherability in monoids* and then considering classes of monoids satisfying certain identities, i.e. *varieties* of monoids.

Let C be a code and let M be a monoid. We say that C is *decipherable* in M if every map $f : C \to M$ extends to a (unique) homomorphism $\overline{f} : C^* \to M$.

The following proposition can be easily proved (cf [7]).

Proposition 1. *Let C be a code. One has:*

- *C is UD if and only if C is decipherable in every monoid.*
- *C is MSD if and only if C is decipherable in every commutative monoid.*
- *C is SD if and only if C is decipherable in every semilattice i.e. a monoid that is commutative and idempotent.*

Let $\mathcal{N}$ be a class of monoids. Denote by $\mathcal{C}(\mathcal{N})$ the class of codes C which are decipherable in every $M \in \mathcal{N}$. Conversely, let $\mathcal{K}$ be a class of codes. Denote by $\mathcal{M}(\mathcal{K})$ the class of monoids M such that every $C \in \mathcal{K}$ is decipherable in M.

Previous definitions lead in a natural way to take into account classes of monoids having special algebraic properties, i.e. *varieties* of monoids (cf [4]). Indeed we have the following proposition(cf Prop 1.3 of [7]).

Proposition 2. *If $\mathcal{K}$ is a class of codes, then $\mathcal{M}(\mathcal{K})$ is a variety of monoids.*

We can now introduce our basic definition.

A class $\mathcal{V}$ of codes is a *variety of codes* if $\mathcal{V} = \mathcal{C}(\mathcal{M}(\mathcal{V}))$.

Remark that not any class of codes is a variety, and that the previous definition establishes a correspondence between varieties of codes and varieties of monoids.

From Proposition 1, one can derive that:

- $\mathcal{UD}$ is the variety of codes corresponding to the variety of all monoids.
- $\mathcal{MSD}$ is the variety of codes corresponding to the variety of commutative monoids.
- $\mathcal{SD}$ is the variety of codes corresponding to the variety of semilattices.

Remark 2. Observe that, by definition, $\mathcal{UD}$ is the smallest variety of codes, i.e., for any variety $\mathcal{V}$ of codes, $\mathcal{UD} \subseteq \mathcal{V}$. Moreover the classification of decipherability conditions of codes in terms of varieties is not trivial. Indeed in [7] it is shown that there exist an *infinite* hierarchy of varieties $\mathcal{V}$ of codes such that $\mathcal{MSD} \subset \mathcal{V}$.

Remark 3. Let C_1 and C_2 be codes and let $\psi : C_1^* \to C_2^*$ be an isomorphism such that $\psi(C_1) = C_2$. Then, for every monoid M, C_1 is decipherable in M if and only if C_2 is decipherable in M. Therefore C_1 and C_2 belong to the same variety.

Next lemma will be useful in the sequel. In order to state the lemma we need some supplementary definitions.

Let X, Y be codes such that $X^+ \cap Y^+ = \emptyset$ and denote by Z their union: $Z = X \cup Y$. Consider a word $w \in Z^+$.

An expression of the type $z_1 z_2 \cdots z_n$ is said to be an (X, Y)-*factorization* of w if

- $w = z_1 z_2 \cdots z_n$
- $z_i \in X^+ \cup Y^+, \ for\ i = 1, \ldots, n$
- if $n > 1$, $z_i \in X^+ \Leftrightarrow z_{i+1} \in Y^+, \ for\ i = 1, \ldots, n-1$

Remark 4. (X, Y)-factorization of a word w may be not unique. The fact that (X, Y)-factorization is not unique is independent from the fact that X and Y are UD. Consider, for instance, $X = \{0\}$, $Y = \{01, 10\}$ and the word $w = 010$. One has: $w = (0)(10) = (01)(0)$ thought X and Y are UD. Otherwise, consider, for instance, $X = \{00, 000\}$ and $Y = \{11, 111\}$. All words $w \in (X \cup Y)^+$ have a unique (X, Y)-factorization, though X and Y are not UD.

Lemma 1. *Let $\mathcal{V}$ be a variety of codes. Let $X, Y \in \mathcal{V}$ be codes such that $X^+ \cap Y^+ = \emptyset$, and consider their union $Z = X \cup Y$. If every $w \in Z^+$ admits a unique (X, Y)-factorization, then $Z \in \mathcal{V}$.*

Proof. Let $\mathcal{N}$ be the variety of monoids associated to the variety $\mathcal{V}$ of codes. We have to show that for all monoids $M \in \mathcal{N}$, Z is decipherable in M. Let $M \in \mathcal{N}$, and let $f : Z \to M$ be a map from Z to M. Denote by g_1 and g_2 the restrictions of f to X and Y respectively: $g_1 := f|_X$, $g_2 := f|_Y$. Since X, Y are decipherable in M, g_1 extends to $\overline{g_1} : X^* \to M$ and g_2 extends to $\overline{g_2} : Y^* \to M$. Let $w \in Z^+$ and suppose, without loss of generality, that its unique (X, Y)-factorization is $w = x_1 y_1 \cdots x_n y_n$. Then, putting $\overline{f}(w) := \overline{g_1}(x_1)\overline{g_2}(y_1) \cdots \overline{g_1}(x_n)\overline{g_2}(y_n)$, we get the unique homomorphism extending f and so Z is decipherable in M.

4 Maximality and Completeness

Let $\mathcal{V}$ be a variety of codes and consider a code $C \in \mathcal{V}$ over the alphabet A. C is called a $\mathcal{V}$-code. C is a *maximal* $\mathcal{V}$-code if it is no proper subset of another $\mathcal{V}$-code over the same alphabet.

The following theorem generalizes to an arbitrary variety of codes a result stated for UD codes (cf Theorem 1).

Theorem 4. *Let $\mathcal{V}$ be a variety of codes and let C be a code in the variety $\mathcal{V}$. If C is a maximal $\mathcal{V}$-code, then it is complete.*

Proof. Let C be a $\mathcal{V}$-code over the alphabet A, with $card(A) \geq 2$ (the case $card(A) < 2$ is trivial). We will prove that, if C is not complete, then there exists a word $w \in A^* \setminus C$ such that $C \cup \{w\}$ is a $\mathcal{V}$-code. Indeed, if C is not complete, there exists a word $v \in A^*$ such that v does not belong to $F(C^*)$. Let a be the first letter of v and let $b \in A$ such that $b \neq a$. Consider the word $w = vab^{|v|}$. By construction, w is *unbordered*, i.e. no proper prefix of w is a suffix of w. Since v does not belong to $F(C^*)$, we have that also w does not belong to $F(C^*)$.

Let us first remark that $C^+ \cap \{w\}^+ = \emptyset$ and that the one word set $\{w\}$ is trivially a UD code (and then it is a $\mathcal{V}$-code). We now prove that every word $t \in (C \cup \{w\})^*$ admits a unique $(C, \{w\})$-factorization. Indeed, since w is unbordered, we can uniquely distinguish all occurrences of w in t, i.e. t has a unique factorization of the form

$$t = u_1 w u_2 w \cdots w u_n,$$

with $n \geq 1$ and $u_i \in C^*$, for $i = 1, \ldots, n$. From this factorization we obtain a unique $(C, \{w\})$-factorization of t. Therefore, by Lemma 1, $C \cup \{w\}$ is a code in the variety $\mathcal{V}$. This concludes the proof.

The converse of previous theorem does not hold in general. We need, as supplementary hypothesis, that the code C is *regular* (as in the case of Theorem 1) and, moreover, that the variety $\mathcal{V}$ is included in the variety $\mathcal{SD}$.

Theorem 5. *Let $\mathcal{V}$ be a variety of codes included in the variety $\mathcal{SD}$, i.e. $\mathcal{V} \subseteq \mathcal{SD}$, and let $C \in \mathcal{V}$ be a regular code. If C is complete, then it is a maximal $\mathcal{V}$-code.*

As a consequence of Theorem 4 and Theorem 5 we obtain the following corollary.

Corollary 1. *Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be varieties of codes such that $\mathcal{V}_1 \subseteq \mathcal{V}_2 \subseteq \mathcal{SD}$, and let $C \in \mathcal{V}_1$ be a regular code. If C is maximal in the variety $\mathcal{V}_1$, then it is maximal in the variety $\mathcal{V}_2$.*

This corollary extends to arbitrary varieties a result of [11], originally conjectured by Lempel [9], asserting that no MSD code contains a maximal UD code as a proper subcode.

The proof of Theorem 5 is based on the following lemma, due to Schutzenberger (cf the proof of Theorem 7.4 in [6]).

Lemma 2. *Let $C \subseteq A^*$ be a regular and complete code. Then, for any word $w \in A^*$ there exist a word $v \in C^*$ and a positive integer m such that $(vwv)^m \in C^*$.*

Proof. Since C is a regular set, C^* is a regular set too. Let

$$\mathcal{A} = (A, Q, \delta, i, F)$$

be a finite state automaton recognizing C^*. For any set of states $S \subseteq Q$ and for any word $u \in A^*$, denote by Su the set $\{\delta(q, u); q \in S\}$ of states reached by paths having label u and starting at any state of S. Let $n = \inf card(Qu)$ with u ranging over A^*, and choose u such that $n = card(Qu)$. Since C is complete, we have $xuy = v \in C^*$ for some $x, y \in A^*$. Since $Qxu \subseteq Qu$, it follows that $card(Qv) \leq card(Qu)$. Thus, by minimality, $card(Qv) = n$. Let $P = Qv$. Since $Pv = Qvv \subseteq Qv = P$, it follows from the minimality of n that $Pv = P$ and thus v defines a permutation of P. Thus, replacing v by a suitable power of v, we may assume that $pv = p$ for all $p \in P$. Consider now a word $w \in A^*$ and let $z = vwv$. Again we have $Qz \subseteq Qv$, and thus $Qz = P = Pz$. Thus again, for some power z^m, $m \geq 1$, we have $pz^m = p$ for all $p \in P$. To prove that

$$z^m = (vwv)^m \in C^*,$$

it suffices to show that $qz^m = qv$ for all $q \in Q$. Since $qvv = qv$, it follows that $qz = qvwv = qvvwv = qvz$ and therefore that $qz^m = qvz^m$. Since $Qv = P$, we have that $qvz^m = qv$. Thus $qz^m = qv$ as required. This complete the proof.

Proof **of Theorem 5.** Let $C \subseteq A^*$ be a regular and complete code in the variety $\mathcal{V} \subseteq \mathcal{SD}$. We have to prove that, for any $w \in A^* \setminus C$, $C \cup \{w\}$ does not belong to the variety $\mathcal{V}$. By Lemma 2, for any word $w \in A^*$ there exist words $u, v \in C^*$ and a positive integer m such that $(vwv)^m = u$. This means that we have the equality $(vwv)^m = u$ between elements of $(C \cup \{w\})^*$, where the word w appears only on the left side of the equality. As a consequence, $C \cup \{w\} \notin \mathcal{SD}$, and therefore $C \cup \{w\}$ does not belong to the variety $\mathcal{V}$. This concludes the proof.

The following theorem, which has an independent interest, is essential for the proof of the main result in next section.

Theorem 6. *Let $\mathcal{V}$ be a variety of codes and let $C \in \mathcal{V}$ be a regular code. Then there exists a regular and complete code Y such that $C \subseteq Y$ and $Y \in \mathcal{V}$.*

Proof. In the proof we make use of a construction, introduced by Ehrenfeucht and Rozenberg in [5] (cf also [1]), for embedding a regular UD code in a complete and regular UD code. The kernel of the proof is to show that the construction works also for an arbitrary variety.

If C is complete there is nothing to prove. Assume that C is not complete. Then, as in the proof of Theorem 4, there exists an *unbordered* word w such that $w \notin F(C^*)$. Let

$$U := (C^*)^c \cap (A^* w A^*)^c,$$

where L^c denotes the complement of a set $L \subseteq A^*$. Consider now the code

$$T := w(Uw)^* = \{wu_1wu_2 \cdots u_nw \mid n \geq 0,\ u_i \notin C^*,\ w \notin F(u_i),\ i = 1, \ldots, n\}$$

and the code $Y := C \cup T$. We will prove that:

1) $T \in \mathcal{UD}$
2) Y is regular and complete
3) $Y \in \mathcal{V}$

The proofs of 1) and 2) are very close to Ehrenfeucht and Rozenberg's proof.

1) Since w is unbordered, any word $s \in T^*$ has a unique representation of the form $s = wv_1wv_2 \cdots wv_kw$, with $w \notin F(v_i)$, for $i = 1, \ldots, k$, that is, we can uniquely distinguish all occurrences of w in s. By construction, for $i = 1, \ldots, n$, either $v_i = \epsilon$ (the empty word), or $v_i \in U$. The values of i such that $v_i = \epsilon$ corresponds to the parsing lines in the factorization of s in elements of T and this proves that s admits a unique factorization in elements of T, i.e. that T is a UD code.

2) By construction Y is obtained by regular operations on regular languages so it is regular. Let us now prove that Y is complete. Actually we show that, for any word $u \in A^*$, $wuw \in Y^*$. Since w is unbordered, the word u has a *unique* representation of the form

$$u = v_1wv_2w \cdots wv_n,$$

with $n \geq 1$ and $w \notin F(v_i)$, for $i = 1, \ldots, n$. Let $v_{i_1}, \ldots, v_{i_k}$, with $k \geq 0, 1 \leq i_j \leq n$ be the elements, among the v_i's, belonging to C^*. Then the word wuw can be (uniquely) factorized as follows:

$$wuw=(wv_1w\cdots wv_{i_1-1}w)v_{i_1}(wv_{i_1+1}w\cdots wv_{i_2-1}w)v_{i_2}\cdots v_{i_k}(wv_{i_k+1}w\cdots wv_nw).$$

The elements in the factorization between parenthesis belong to T and the other ones belong to C^*. Therefore $wuw \in (C \cup T)^* = Y^*$.

3) We first prove that all words $u \in Y^+ = (C \cup T)^+$ admit a unique (C, T)-factorization.

As before u has a *unique* representation of the form

$$u = v_1wv_2w\cdots wv_n,$$

with $n \geq 1$ and $w \notin F(v_i)$, for $i = 1, \ldots, n$. Let $v_{i_1}, \ldots, v_{i_k}$, with $1 \leq i_j \leq n$, $1 \leq j \leq k \leq n$ be the elements, among the v_i's, belonging to C^*. Since $u \in (C \cup T)^+$ and by definition of T we have $i_1 = 1$ and $i_k = n$. Then the word u can be uniquely factorized as follow:

$$u = v_1(wv_2\cdots wv_{i_2-1}w)v_{i_2}\cdots v_{i_{k-1}}(wv_{i_{k-1}+1}w\cdots v_{i_k-1}w)v_n.$$

The values of j such that $v_{i_j} \neq \varepsilon$ give the elements in the (C, T)-factorization of u belonging to C^+, the elements between parenthesis, or their concatenations, give the elements belonging to T^+.

We have then proved that all words $u \in Y^+ = (C \cup T)^+$ admit a unique (C, T)-factorization. Now, since $T \in \mathcal{UD}$ and $\mathcal{UD} \subseteq \mathcal{V}$, by Lemma 1, we have that $Y \in \mathcal{V}$.

5 Main Result

In [9] Lempel conjectured that every MSD code satisfies Kraft inequality. However, in [11] it is shown that there exist MSD codes C such that $\mathcal{K}(C) > 1$. An interesting question is to characterize those varieties $\mathcal{V}$ of codes where the Kraft inequality is satisfied. The answer is given by the following theorem.

Theorem 7. *Let $\mathcal{V}$ be a variety of codes. If $\mathcal{K}(C) \leq 1$ for every $C \in \mathcal{V}$, then $\mathcal{V} = \mathcal{UD}$.*

Proof. Assume, by contradiction, that $\mathcal{V} \neq \mathcal{UD}$. Then there exists a code $C \in \mathcal{V}$ such that $C \notin \mathcal{UD}$.

Suppose first that C is finite. Then C is regular and, by Theorem 6, there exists a regular and complete code $Y \in \mathcal{V}$ such that $C \subseteq Y$. By hypothesis, $\mathcal{K}(Y) \leq 1$. By Theorem 2, $\mathcal{K}(Y) \geq 1$, hence $\mathcal{K}(Y) = 1$. As a consequence of Theorem 3, we have that $Y \in \mathcal{UD}$. Since C is a subset of Y, one has that $C \in \mathcal{UD}$, a contradiction.

Suppose now that C is not finite. Since $C \notin \mathcal{UD}$, then exists a word $w \in C^+$ having two distinct factorizations:

$$w = c_1\cdots c_s = c_{s+1}\cdots c_n \ , c_i \in C \ , i = 1, \ldots, n.$$

Put $C' := \{c_1, c_2, \ldots, c_n\}$. One has that $C' \in \mathcal{V}$, $C' \notin \mathcal{UD}$ and C' is finite. So we are brought back to the previous case (C finite) and the proof is concluded.

Remark 5. It is well known that the Kraft inequality is a necessary condition for UD codes, but it is not sufficient (cf Remark 1). However, by previous theorem, we can assert that, in terms of varieties, Kraft inequality characterizes UD codes.

From the proof of Theorem 6 we also derive the following theorem which states that, even if the Kraft sum of a code $C \in \mathcal{V}$, with $\mathcal{V} \neq \mathcal{UD}$, can exceed the unity, however there exists a complete codes in $\mathcal{V}$ whose Kraft sum is arbitrarily close to the unity.

Theorem 8. *Let $\mathcal{V}$ be a variety of codes such that $\mathcal{V} \neq \mathcal{UD}$. Then, for any $\varepsilon > 0$, there exists a complete (and regular) code $Y \in \mathcal{V}$ such that*

$$1 < \mathcal{K}(Y) < 1 + \varepsilon.$$

Proof. Let C be a finite code over the alphabet A, with $card(A) = d$, such that $C \in \mathcal{V}$ and $C \notin \mathcal{UD}$. Let $C = \{c_1, \ldots, c_n\}$ and let $m(C)$ denotes the minimal length of words in C. For any $\varepsilon > 0$, let r be a positive integer such that

$$nd^{-rm(C)} < \varepsilon.$$

Given a word $u = a_1 \cdots a_t$, $a_i \in A$, $i = 1, \ldots, t$, denote by $u^{(r)} = a_1^r \cdots a_t^r$ be the word obtained from u by substituting each letter a occurring in u with the word a^r, i.e. the letter a repeated r times. By construction, $|u^{(r)}| = r|u|$. Consider now the code

$$C^{(r)} = \{c_1^{(r)}, \ldots, c_n^{(r)}\}.$$

It is easy to verify that the mapping $\psi_r : C \to C^{(r)}$ defines an isomorphism between the submonoids generated by C and $C^{(r)}$, respectively. Therefore, by Remark 3, one has that $C^{(r)} \in \mathcal{V}$. Moreover one has:

$$\mathcal{K}(C^{(r)}) = \sum_{c \in C^{(r)}} d^{-|c|} = \sum_{c \in C} d^{-r|c|} \leq \sum_{c \in C} d^{-rm(C)} = nd^{-rm(C)} < \varepsilon.$$

As in the proof of Theorem 6, consider the code T corresponding to the code $C^{(r)} \in \mathcal{V}$ and the code $Y = T \cup C^{(r)}$, which is again a code in the variety $\mathcal{V}$. Since Y is complete and it is not UD, as a consequence of Theorem 3 one has that $\mathcal{K}(Y) > 1$. Moreover, since T is UD, one has $\mathcal{K}(T) \leq 1$, and then:

$$1 < \mathcal{K}(Y) = \mathcal{K}(T \cup C^{(r)}) = \mathcal{K}(T) + \mathcal{K}(C^{(r)}) < 1 + \varepsilon.$$

This concludes the proof.

6 Open Problems and Concluding Remarks

There are some problems, which are still open, concerning varieties of codes and Kraft inequality. A first natural question is, given a variety $\mathcal{V}$ of codes, whether

there exists a constant $\Gamma_{\mathcal{V}}$ such that all the codes $C \in \mathcal{V}$ satisfy a *generalized Kraft inequality* $\mathcal{K}(C) \leq \Gamma_{\mathcal{V}}$. Such inequalities would express, in a quantitative way, a trade-off between the relaxed decipherability conditions and the efficiency of the encoding process. Tight upper bounds to the Kraft sum are known for the extremal varieties: indeed $\Gamma_{\mathcal{V}} = 1$ for $\mathcal{V} = \mathcal{UD}$, which is the smallest variety of codes, and, trivially, $\Gamma_{\mathcal{V}} = \infty$ if $\mathcal{V}$ is the variety of *all* codes. For the intermediate cases the problem is open.

Let us further remark that, by Theorem 8, in variety $\mathcal{V} \neq \mathcal{UD}$, the Kraft sum can exceed the unity, but if $\mathcal{V} \subseteq \mathcal{SD}$, by Theorem 5 we can have codes, maximal in the variety $\mathcal{V}$, such that their Kraft sum is arbitrarily close to the unity. As a consequence, whereas in the variety $\mathcal{UD}$ all maximal codes have the same Kraft sum (which is equal to the unity), on the contrary, given a variety $\mathcal{V} \neq \mathcal{UD}$ such that $\mathcal{V} \subseteq \mathcal{SD}$, distinct maximal elements in the variety have, in general, different Kraft sums. This provides a further characterization of the variety $\mathcal{UD}$, in terms of the Kraft sum.

References

1. Berstel, J., Perrin, D.: The Theory of Codes, Academic Press, New York, 1985.
2. Blanchet-Sadri, F, Morgan, C.: Multiset and set decipherable codes, Computers and Mathematics with Applications **41** (2001) 1257-1262.
3. Boe, J.M., De Luca, A., Restivo, A.: Minimal complete sets of words, Theoretical Computer Science **12** (1980) 325-332.
4. Burris, S., Sankappanavar, H.P.: A Course in Universal Algebra, Springer, New York, 1981.
5. Ehrefeucht, A., Rozenberg, G.: Each regular code is included in a maximal regular code, RAIRO Inform. Theor. Appl. **20** (1986) 89-96.
6. Eilenberg, S.: Automata, Languages and Machines, Vol.A, Academic Press, New York, 1974.
7. Guzmán, F.: Decipherability of codes, Journal of Pure and Applied Algebra **141** (1999) 13-35.
8. Head, T., Weber, A.: Deciding multiset decipherability, IEEE Trans. Inform. Theory **41** (1995) 291-297.
9. Lempel, A.: On multiset decipherable codes, IEEE Trans. Inform. Theory **32** (1986) 714-716.
10. Lothaire, M.: Algebraic Combinatorics on Words, Cambridge University Press, 2002.
11. Restivo, A.: A note on multiset decipherable code, IEEE Trans. Inform. Theory **35** (1989) 662-663.

Improving the Alphabet-Size in High Noise, Almost Optimal Rate List Decodable Codes

Eran Rom and Amnon Ta-Shma

Computer Science Department, Tel-Aviv University, 69978 Tel-Aviv, Israel
{eranrom, amnon}@post.tau.ac.il

Abstract. We revisit the construction of high noise, almost optimal rate list decodable code of Guruswami [1]. Guruswami showed that if one can explicitly construct optimal extractors then one can build an explicit $(1-\epsilon, O(\frac{1}{\epsilon}))$ list decodable codes of rate $\Omega(\frac{\epsilon}{log\frac{1}{\epsilon}})$ and alphabet size $2^{O(\frac{1}{\epsilon}\cdot log\frac{1}{\epsilon})}$. We show that if one replaces the expander component in the construction with an unbalanced disperser, then one can dramatically improve the alphabet size to $2^{O(log^2\frac{1}{\epsilon})}$ while keeping all other parameters the same.

1 Introduction

Error correcting codes were built to deal with the task of correcting errors in transmission over noisy channels. Formally, an $(N, n, d)_q$ [1] error correcting code over alphabet Σ, where $|\Sigma| = q$, is a subset $C \subseteq \Sigma^N$ of cardinality q^n in which every two elements are distinct in at least d coordinates. n is called the dimension of the code, N the block length of the code, and d the distance of the code. If C is a linear subspace of $[\mathbb{F}_q]^N$, where Σ is associated with some finite field $\mathbb{F}_q$ we say that C is a linear code, and denote it $[N, n, d]_q$ code. From the definition we see that one can uniquely identify a codeword in which at most $\frac{d-1}{2}$ errors occurred during transmission. Moreover, since two codewords from Σ^N can differ in at most N coordinates, the largest number of errors from which unique decoding is possible is $N/2$.

This motivates the list decoding problem, first defined in [2]. In list decoding we give up unique decoding, allowing potentially more than $N/2$ errors, and require that there are only few possible codewords having some modest agreement with any received word. Formally, we say that an $(N, n, d)_q$ code C is (p, K)-list decodable, if for every $r \in \Sigma^N$, $|\{c \in C | \Delta(r, c) \leq pN\}| \leq K$, where $\Delta(x, y)$ is the number of coordinates in which x and y differ. That is, the number of codewords which agree with r on at least $(1-p)N$ coordinates is smaller than K. We call the ratio n/N the rate of the code, and p the error rate.

[1] We will use n to denote the dimension of a code to avoid confusion with with the min–entropy parameter of extractors and dispersers, for which k is usually reserved.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 557–568, 2005.

We can demonstrate the difference between unique decoding and list decoding with Reed-Solomon codes. Reed-Solomon codes are linear $[N, n, N-n+1]_q$ codes, defined for every q such that $\mathbb{F}_q$ is a finite field, and $n \leq N \leq q$. Every $(N, n, d)_q$ code is $(\sqrt{1-d/N}, qN)$-list decodable (see, [3], Lecture 8). For Reed-Solomon codes there exists an efficient list decoding algorithm [4]. Thus, unique decoding is possible with at most $N/2$ errors, while by [4] list decoding is possible with up to $N - \sqrt{Nn}$ errors, and the number of all possible decodings is small.

We focus on the high noise regime, where $p = 1 - \epsilon$, for $\epsilon > 0$ being very small. A simple probabilistic argument shows that $(1-\epsilon, O(\frac{1}{\epsilon}))$-list decodable codes with $rate = \Omega(\epsilon)$, and $|\Sigma| = O(\frac{1}{\epsilon^2})$ exist. Also the rate must be $O(\epsilon)$, and $|\Sigma| = \Omega(\frac{1}{\epsilon})$. Until recently, the best known explicit constructions only achieved rate of ϵ^2. Recently, Guruswami in [1], used an expander based construction to give the first explicit construction of rate $\Omega(\frac{\epsilon}{\log^{O(1)} \frac{1}{\epsilon}})$. However, the alphabet size and the decoding list size in this construction are huge.

Although Guruswami's result suffers the drawbacks of huge decoding list size and huge alphabet size it is interesting as it improves our understanding of the relationships between extractors, expanders and codes. Specifically, it gives motivation for explicitly constructing better extractors which will yield better codes.

1.1 Our Results

The relationship that [1] has found between strong extractors[2] and high noise list decodable codes is given in the following theorem:

Theorem 1 (Old connection between strong extractors and L.D.C.). *Let $K = K(N)$ be arbitrary. If a family of $(K, 1/4)$-strong extractors $f : [N] \times [D] \to [M]$ with degree $D = O(\log N)$ and entropy loss $O(1)$ can be explicitly constructed, then one can explicitly construct $(1-\epsilon, O(1/\epsilon))$-list decodable codes of rate $\Omega(\frac{\epsilon}{log(1/\epsilon)})$ over an alphabet size $2^{O(\epsilon^{-1} log(1/\epsilon))}$*

We show:

Theorem 2 (New connection between strong extractors and L.D.C.). *Let $K = K(N)$ be arbitrary. If a family of $(K, 1/4)$-strong extractors $f : [N] \times [D] \to [M]$ with degree $D = O(\log N)$ and entropy loss $O(1)$ can be explicitly constructed, then one can explicitly construct $(1-\epsilon, O(1/\epsilon))$-list decodable codes of rate $\Omega(\frac{\epsilon}{\log(1/\epsilon)})$ over an alphabet size $2^{2^{polyloglog(\frac{1}{\epsilon})}}$.*[3]

Note that all parameters are the same as in [1], except the significantly smaller alphabet size. Using the best explicit construction of strong extractors we have today [1] has shown:

[2] The definition of strong extractors and dispersers is given in Sect. 1.2.

[3] If we further assume an optimal disperser then the alphabet size can be reduced to $2^{O(\log^2 \frac{1}{\epsilon})}$.

Theorem 3 (Old connection between explicit strong extractors and L.D.C.). *For every constant $\epsilon > 0$, there is a polynomial time constructible family of $(1-\epsilon, 2^{O(\sqrt{n \log n})})$-list decodable codes of rate $\Omega(\frac{\epsilon}{polylog(1/\epsilon)})$ over an alphabet of size $2^{O(\epsilon^{-1}\log(1/\epsilon))}$, where n is the dimension of the code.*

Again, we show that the alphabet size can be improved:

Theorem 4 (New connection between explicit strong extractors and L.D.C.). *For every constant $\epsilon > 0$, there is a polynomial time constructible family of $(1-\epsilon, 2^{O(\sqrt{n \log n})})$-list decodable codes of rate $\Omega(\frac{\epsilon}{polylog(1/\epsilon)})$ over an alphabet of size $2^{2^{polyloglog(\frac{1}{\epsilon})}}$, where n is the dimension of the code.*

1.2 The Technique

In order to understand our technical contribution we first need to understand what Guruswami did in [1].

Introducing the Basic Objects

Strong Extractors. An extractor is a function which extracts randomness from a weak random source. A weak random source is a distribution which might be far form uniform but still has some randomness in it. A standard measure for the amount of randomness contained in a source is its min–entropy. A distribution X over $\{0,1\}^n$ has k min–entropy, denoted $H_\infty(X) = k$, if $\forall x,\ X(x) \leq 2^{-k}$. If $H_\infty(X) = k$ we say that X has k bits of min–entropy. An example of a weak random source is a uniform distribution over some subset of 2^k elements from $\{0,1\}^n$.

A simple fact is that randomness extraction from a weak source cannot be done without additional randomness independent of the source. This leads to the following definition:

Definition 1. *$f : [N] \times [D] \to [M]$ is a (K, ζ_{ext})-strong extractor if for every X distributed over $[N]$ with $H_\infty(X) \geq \log K$, the distribution $y \circ f(x,y)$ is ζ_{ext}-close to $U_{[D]\times[M]}$, where x is drawn from X and y is taken uniformly at random from $[D]$. The entropy loss of the strong extractor is $\frac{K}{M}$. ζ_{ext} is called the extractor error. The strong extractor is explicit if $f(x,y)$ can be computed in time polynomial in the input length, i.e., polynomial in $\log N + \log D$.*

That is the extractor gets an input from some unknown distribution X that is guaranteed to have at least $\log K$ min–entropy and uses some additional $\log D$ truly random bits, called the seed of the extractor, to extract $\log M$ random bits that together with the seed are close to uniform. An extractor (not necessarily strong) is one where we only require that the $\log M$ output bits are close to uniform.

Dispersers. A disperser is the one-sided variant of an extractor. Instead of requiring that the output is ϵ-close to the uniform distribution, we require that the disperser's output covers at least a $1-\epsilon$ fraction of the target set.

Definition 2. *$g : [L] \times [T] \to [D]$ is a (H, ζ_{disp})-disperser if for every $X \subseteq [L]$ with $|X| \geq H$ we have $|\{g(l,j) | l \in X, j \in [T]\}| \geq (1-\zeta_{disp})D$. The entropy loss of the disperser is $\frac{HT}{D}$. The disperser is explicit if $g(x,y)$ can be computed in time polynomial in the input length, i.e., polynomial in $\log L + \log T$.*

In definition 1 we defined a *strong* extractor, while in definition 2 we defined a (not necessarily strong) disperser. This is due to the way we use these objects later on.

Extractor Codes. [5] observed a simple connection between strong extractors and list decodable codes. Given a strong extractor $f : [N] \times [D] \to [M]$, we define a code $C : [N] \to [M]^D$ as follows: $\forall x \in [N]$, $C(x)_i = f(x,i)$. By definition the rate of the code is $\frac{\log N}{D \log M}$. The connection is summarized by the following lemma:

Lemma 1. *If $f : [N] \times [D] \to [M]$ is a (K, ζ_{ext})-strong extractor then the extractor code $C(x)$ is $(1-(\frac{1}{M}+\zeta_{ext}), K)$-list decodable code.*

Also observed by [5] is that extractor codes meet a property stronger than list decoding, known as list recovering [6]. List recovering deals with the situation where the i^{th} symbol of the received word is only known to be in some set $S_i \subseteq \Sigma$. The goal is to find a code $C \subseteq \Sigma^N$ such that for every given $S_1, \ldots S_N \subseteq \Sigma$ describing a received word, there are not to many codewords $C(x)$ with $C(x)_i \in S_i$ for many indices i. List decoding is the case where all sets S_i are of size 1.

Error Amplification Using Expanders. The technique of code amplification using expanders was introduced in [7], where it is used to amplify Justesen code. Justesen code rate vanishes as the error rate grows. [7] take Justesen code of constant error rate and amplify it using an expander to get a code with large distance and constant rate over a large alphabet. Looking back, the amplification in [7] can be done using any *disperser* (a good expander is just a special case).

Here is how the amplification is done: Assume $C : \Sigma^n \to \Sigma^D$ is a (p,K)-list decodable code. Let $g : [L] \times [T] \to [D]$ be a (H, ζ_{disp})-disperser. Define a code $C_g : \Sigma^n \to [\Sigma^T]^L$. For $x \in \Sigma^n$ let $C(x)$ be its encoding using C. Given $C(x) \in \Sigma^D$, we put its symbols along the output of g, such that the $i'th$ symbol of the codeword is matched with the $i'th$ output element of g. We now look at the input elements in $[L]$, each such element has T neighbors each matched with a symbol from Σ. For each input element we collect the symbols of its neighbors and get a symbol in Σ^T. Altogether, we get a code $C_g : \Sigma^n \to [\Sigma^T]^L$. A simple argument shows:

Lemma 2. *If $\zeta_{disp} \leq p$, then $C_g(x)$ is $(1-\frac{H}{L}, K)$-list decodable.*

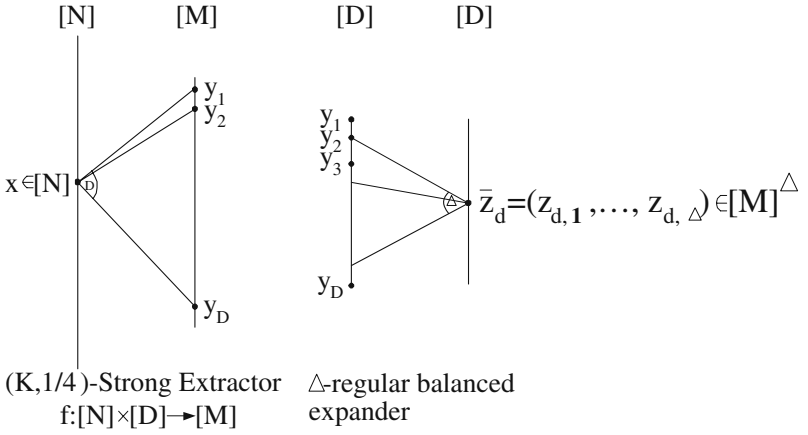


Fig. 1. The strong extractor on the left has a constant error, giving an extractor code with $\frac{3}{4} - \frac{1}{M}$ error rate. The code produced by the extractor is then amplified using the balanced expander on the right. The balanced expander used is equivalent to some $(\epsilon D, \zeta_{disp})$-disperser $g : [D] \times [\Delta] \to [D]$

Guruswami's Construction. Lemma 1 shows that an extractor code already has a good error rate. However, the lower bound of $\Omega(\frac{1}{\zeta_{ext}^2})$ on the extractor degree (see Sect. 2.1) implies an upper bound of $O(\zeta_{ext}^2)$ on the extractor code's rate. To overcome this obstacle, Guruswami uses a strong extractor with a constant error and amplifies it using a balanced expander, see figure 1. As shown in Section 3, the construction takes advantage of the list recovering property of extractor codes.

Our Improvement. What we do is replace the expander component in [1] with a good *unbalanced* disperser. Studying the problem we discover that what is needed is a disperser for the high min–entropy rate, that has optimal entropy loss and a surprisingly small degree. Fortunately, an explicit construction of such a graph was given recently [8]. Using such a graph, our improvement over the construction in [1] can be made explicit. For every code built using Guruswami's scheme, the expander component can be replaced with the explicit disperser and improve the alphabet size. As the disperser is explicit, the decoding scheme mentioned in [1] and the time it takes does not change.

2 Preliminaries

We need the following standard definitions: A probability distribution X over Ω is a function $X : \Omega \to [0, 1]$, such that $\sum_{x \in \Omega} X(x) = 1$. U_n is the uniform distribution over $\{0, 1\}^n$. The statistical distance between two probability distributions X, Y distributed over Ω, denoted $|X - Y|$, is $\frac{1}{2} \sum_{x \in \Omega} |X(x) - Y(x)| = max_{S \subseteq \Omega} |X(S) - Y(S)|$. X, Y are ϵ-close if $|X - Y| \leq \epsilon$.

2.1 Bounds of the Parameters Achievable for Extractors and Dispersers

Ta-Shma and Radhakrishnan [9] show that a (K, ζ_{ext})-strong extractor $f : [N] \times [D] \to [M]$ must have degree $D = \Omega(\frac{1}{\zeta_{ext}^2} \log \frac{N}{K})$, and entropy loss $\frac{K}{M} = O(\frac{1}{\zeta_{ext}^2})$. Also shown in [9] are matching implicit upper bounds. The degree lower bound gives the minimal true randomness needed for extracting randomness from a weak source. The entropy loss lower bound gives the amount of randomness lost by the process. [9] also give matching lower bounds and non-explicit upper bounds for dispersers. Specifically, a (H, ζ_{disp})-disperser $g : [L] \times [T] \to [D]$ must have $T = \Omega(\frac{1}{\zeta_{disp}} \log \frac{L}{H})$, and entropy loss $\frac{HT}{D} = \Omega(log \frac{1}{\zeta_{disp}})$.

2.2 The Mixing Property

An important property of extractors is mixing (see, [10], Chap 9). We introduce some notation. For $x \in [N]$ we define $\Gamma_f(x)$ to be the ordered neighbors of x. Formally,

$$\Gamma_f(x) = \{(i, f(x, i) | i \in [D]\} \ . \tag{1}$$

The mixing property says that:

Fact 5 *If $f : [N] \times [D] \to [M]$ is a (K, ζ_{ext})-strong extractor, then for every $S \subseteq [D] \times [M]$, there are at most K elements $x \in [N]$ satisfying*

$$\frac{|\Gamma_f(x) \cap S|}{D} - \frac{|S|}{D \cdot M} \geq \zeta_{ext} \ . \tag{2}$$

3 A Better Connection Between Strong Extractors and L.D.C.

The connection shown below is basically Guruswami's, except that Guruswami uses a balanced, good expander and we use a slightly unbalanced good disperser. Let: $f : [N] \times [D] \to [M]$ be a (K, ζ_{ext})-strong extractor, and let $g : [L] \times [T] \to [D]$ be a (H, ζ_{disp})-disperser. We define the code $C_{f,g} : [N] \to [M^T]^L$ as follows:

1. Given $x \in [N]$, denote by $\overline{y} = (y_1, \ldots, y_D) \in [M]^D$ where $y_i = f(x, i)$.
2. Put the symbols $(y_1, \ldots, y_D) \in [M]^D$ along g's range $[D]$. Each element $\ell \in [L]$ has T neighbors in $[D]$. Collect from each neighbor the symbol that was put along it. I.e., for each $\ell \in [L]$ define $\overline{w}_\ell = (w_{\ell,1}, \ldots, w_{\ell,T}) \in [M]^T$, where $w_{\ell,t} = y_{g(\ell,t)}$.
3. The encoding of x is defined to be

$$C_{f,g}(x) = (\overline{w}_1, \ldots, \overline{w}_L) \ . \tag{3}$$

See figure 2 for an illustration of the construction. We claim:

Lemma 3. *If the extractor f and the disperser g are as above, and if $M \cdot D \geq \frac{L \cdot T}{1 - \zeta_{ext} - \zeta_{disp}}$, then $C_{f,g}$ is $(1 - \frac{H}{L}, K)$-list decodable.*

An eye on figure 2 might be helpful while reading the proof.

Proof. Let $z = (\overline{z}_1, \ldots, \overline{z}_L) \in [M^T]^L$ be an arbitrary word in $[M^T]^L$. From z we build a set S as follows. For each $1 \leq \ell \leq L$, we look at $\overline{z}_\ell = z_{\ell,1}, \ldots, z_{\ell,T}$ and we build the set $S_\ell \subseteq [D] \times [M]$ by:

$$S_\ell = \{(g(\ell,t), z_{\ell,t}) | 1 \leq t \leq T\} \ . \tag{4}$$

S_ℓ represents what $\overline{z}_\ell$ thinks $y_1, \ldots, y_D$ are in locations $g(\ell,1), \ldots, g(\ell,T)$. We define the set S of $z = (\overline{z}_1, \ldots, \overline{z}_L)$ to be $\bigcup_{\ell=1}^{L} S_\ell$.

Suppose a codeword $C_{f,g}(x) \in [M^T]^L$ agrees with z on a set $\mathcal{H} \subseteq [L]$ of size at least H. Now, if $\ell \in \mathcal{H}$ then $(i, f(x,i)) \in S_\ell$ for every $i \in [D]$ such that i is a neighbor of l in g (because the l^{th} coordinate is the concatenation of all the symbols along the neighbors of ℓ in g). Since g is a (H, ζ_{disp})-disperser, the set of neighbors of $\mathcal{H}$ has at least $(1-\zeta_{disp})D$ elements. Hence, $|\Gamma_f(x) \cap S| \geq (1-\zeta_{disp})D$. Noting that $|S| \leq L \cdot T$, and using the assumption $M \cdot D \geq \frac{L \cdot T}{1-\zeta_{ext}-\zeta_{disp}}$, we see that $\frac{|S|}{MD} \leq 1 - \zeta_{ext} - \zeta_{disp}$ and together

$$\frac{|\Gamma_f(x) \cap S|}{D} - \frac{|S|}{MD} \geq \zeta_{ext} \ . \tag{5}$$

By Fact 5 we conclude that there are at most K x's for which $C_{f,g}(x)$ agrees with z on at least H coordinates. Hence the code is $(1-\frac{H}{L}, K)$-list decodable. □

3.1 What Makes the Difference

First, let us have a second look at Guruswami's construction. A strong extractor gives a list decodable code that can correct $1-\alpha$ noise with α^2 penalty in the rate, and so we do not lose much when α is a constant. Indeed, on the left of figure 2 we use a strong extractor for a constant error rate.

We are then left with the task of amplifying the error. For that Guruswami uses a balanced expander. The property that we need from the expander, is that every set (of relatively small cardinality H) on the right hand side (of figure 2) sees almost all of the vertices on the left hand side as its neighbors (more precisely $1-\zeta_{disp}$ of them).

Taking a balanced expander does the job, but at the cost of enlarging the disperser degree T. This is because H vertices can have at most HT neighbors, and so if H is small and HT is almost D, we must have a disperser with a large degree T. On the other hand, if we take a larger right hand side L (such that H is roughly D) we can use a much smaller degree T and still have the same property.

To see how the parameters behave, we notice that the size of the alphabet is determined by T (and so we get a much smaller alphabet size) and the rate is determined by $L \cdot T$ and L should be H/ϵ (to provide the necessary amplification). The fact that L is larger does not translate to an inferior rate, because T is smaller and so $L \cdot T$ stays exactly as in Guruswami's construction. We thus keep the rate while dramatically reduce the alphabet size.

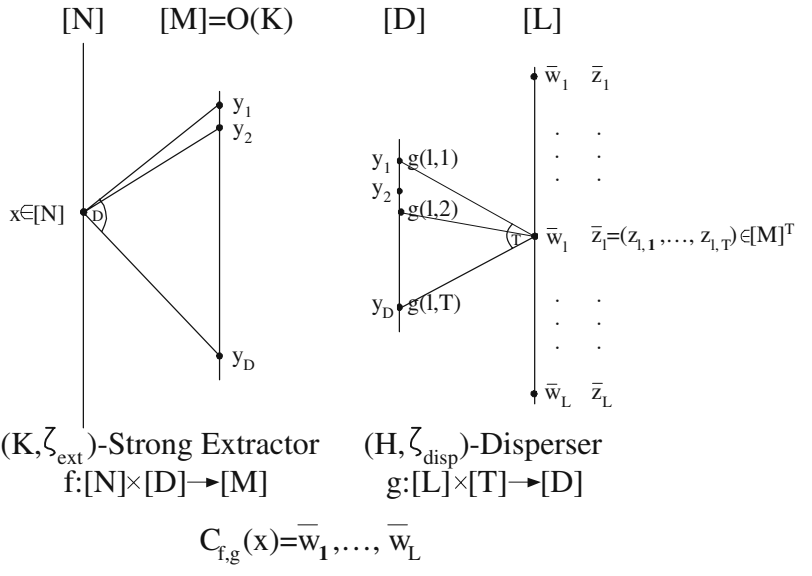


Fig. 2. The neighbors of $x \in [N]$ on the left: $(y_1, \ldots, y_D)$ are "put" along the disperser's output $[D]$, defining for each $l \in [L]$ an ordered vector of its neighbors $\overline{w}_l = (w_{l,1}, \ldots, w_{l,T}) \in [M]^T$. The vector $\overline{w}_l$ is the l^{th} symbol in the codeword $C_{f,g}(x)$. $z = (\overline{z}_1, \ldots, \overline{z}_L)$ is an arbitrary word in $[M^T]^L$. $\overline{z}_l \in [M]^T$ is the l^{th} coordinate of z and $g(l,1), \ldots, g(l,T)$ are the neighbors of $l \in [L]$ in g. Each neighbor $g(l,t)$ of l is associated with $z_{l,t} \in [M]$, the t^{th} element of $\overline{z}_l$. The pairs $\{(g(l,t), z_{l,t})\}_{t=1,\ldots,T}$ form the set $S_l \subset [D] \times [M]$. An agreement between $\overline{z}_l$ and $\overline{w}_l$ implies that $y_t = z_{l,t}$ for all $1 \leq t \leq T$, i.e. agreement on t out of the D symbols $(y_1, \ldots, y_D)$. An agreement between z and $C_{f,g}(x)$ on H coordinates will thus translate to an agreement on $(1 - \zeta_{disp})D$ of the symbols $(y_1, \ldots, y_D)$

For this to work we need a good disperser that works for the high min–entropy setting (where H is very close to L) and tiny degree (the optimal is $O(\log(L/H))$. Luckily, the recent Zig-Zag construction [8] explicitly constructs such a graph.

3.2 Analyzing the Parameters

We now find out the parameters of the extractor and disperser to be used in the construction, so as to get a $(1 - \epsilon, O(\frac{1}{\epsilon}))$-list decodable code. These parameters must not violate the lower bounds of the extractor and disperser, and the condition of Lemma 3. Since the components' lowers bounds have matching non-explicit upper bounds, the parameters we find give a non-explicit construction for the desired code.

The Constraints. First, we write down all the constraints. The bounds we give are both lower bounds, and achievable by non-explicit constructions. We have:

$$D = \Omega\left(\frac{1}{\zeta_{ext}^2} \cdot \log\frac{N}{K}\right) \quad . \tag{6}$$

$$M = O\left(K\zeta_{ext}^2\right) \quad . \tag{7}$$

$$T = \Omega\left(\frac{1}{\zeta_{disp}} \cdot \log\frac{L}{H}\right) \quad . \tag{8}$$

$$D = O\left(\frac{HT}{\log\frac{1}{\zeta_{disp}}}\right) \quad . \tag{9}$$

$$M \cdot D \geq \frac{L \cdot T}{1 - \zeta_{ext} - \zeta_{disp}} \quad . \tag{10}$$

The first two equations are the degree and entropy loss of the extractor, the third and fourth are the degree and entropy loss of the disperser, and the fifth is the construction bound that guarantees that the set S is small in $[D] \times [M]$.

A Specific Choice of Parameters. We now choose parameters. We first set $\zeta_{ext}, \zeta_{disp}$ to be small constants, say we set both to be $\frac{1}{4}$. In order to get a $(1-\epsilon, O(\frac{1}{\epsilon}))$-list decodable code we set K to be $K = \Theta(\frac{1}{\epsilon})$. With these choices we have $D = \Theta(\log(N))$, and $M = \Theta(K) = \Theta(\frac{1}{\epsilon})$. We also set $\frac{H}{L} = \epsilon$. This implies that $T = \Theta(\log(\frac{L}{H})) = \Theta(\log\frac{1}{\epsilon})$. To satisfy Equation (9) we need to take $H = \Theta(\frac{D}{T}) = \Theta(\frac{\log(N)}{\log\frac{1}{\epsilon}})$ which implies that $L = \frac{H}{\epsilon} = \Theta(\frac{\log(N)}{\epsilon \cdot \log(\frac{1}{\epsilon})})$. Finally, we check Equation (10). We see that $M \cdot D = \Theta(\frac{\log(N)}{\epsilon})$ and $L \cdot T = \Theta(\frac{\log(N)}{\epsilon})$, so with the proper choice of constants the equation holds. We let $N = 2^n$ and $\epsilon > 0$ be our basic parameters. We summarize all other parameters as functions in n and ϵ. We have,

$$K = \Theta\left(\frac{1}{\epsilon}\right) \quad . \tag{11}$$

$$D = \Theta\left(n\right) \quad . \tag{12}$$

$$M = \Theta\left(\frac{1}{\epsilon}\right) \quad . \tag{13}$$

$$L = \Theta\left(\frac{n}{\epsilon \cdot \log(\frac{1}{\epsilon})}\right) \quad . \tag{14}$$

$$H = \Theta\left(\frac{n}{\log(\frac{1}{\epsilon})}\right) \quad . \tag{15}$$

$$T = \Theta\left(\log\frac{1}{\epsilon}\right) \quad . \tag{16}$$

Thus, $rate = \frac{\log N}{L \cdot T \log M}$ is $\Theta\left(\frac{\epsilon}{\log(\frac{1}{\epsilon})}\right)$, and the alphabet size $|\Sigma| = M^T$ is $2^{O(\log^2(\frac{1}{\epsilon}))}$. This proves that using the best implicit disperses one gets the parameters stated in footnote of Theorem 2.

4 Explicit Constructions

We now make the construction explicit by plugging in explicit disperser and explicit strong extractor. Naturally, the parameters deteriorate. As before, we set the extractor and disperser errors to be constants, say $\zeta_{ext} = \zeta_{disp} = \frac{1}{4}$. We note that (10) now becomes, $L \cdot T \leq \frac{1}{2} \cdot M \cdot D$

4.1 Using Explicit High Min–entropy Optimal Loss Disperser

As suggested by the parameters chosen in 3.2, the construction requires a high min–entropy disperser with optimal entropy loss. Such a disperser is given by [8] in the extractors' analogue of the Zig-Zag construction. Specifically:

Fact 6 ([8]) *For every L and $\epsilon > \frac{1}{\sqrt{L}}$, there exists an explicit construction of $(\epsilon L, \frac{1}{4})$-disperser $g : [L] \times [T] \to [D]$, with $T = 2^{polyloglog(\frac{1}{\epsilon})}$, and entropy loss $\frac{\epsilon L \cdot T}{D} = O(1)$.*

We now prove Theorem 2:

Proof. The disperser from Fact 6 has $\Theta(1)$ entropy loss. Let C_1, C_2 be the constants which bound this entropy loss from below and from above accordingly. Let C_3 be the constant behind the extractor degree $O(\cdot)$ notation from the assumption, where $D = O(\log N)$.

Let $\epsilon > 0$, and $T = 2^{polyloglog(\frac{1}{\epsilon})}$ as in Fact 6. We let $M = \frac{2 \cdot C_2}{\epsilon}$, $N > 2^{\frac{2T}{\epsilon \cdot C_3 (C_1 + C_2)}}$, $D = C_3 \log N$, $L = \frac{C_1 + C_2}{2} \cdot \frac{D}{\epsilon T}$ and $K = \Theta(M)$. By the choice of N, we have that $\epsilon > \frac{1}{\sqrt{L}}$. Thus, by Fact 6 there is an explicit construction of $(\epsilon L, \frac{1}{4})$-disperser $g : [L] \times [T] \to [D]$ and by the assumption there is a $(K, 1/4)$-strong extractor $f : [N] \times [D] \to [M]$. Finally, by the choice of M, $LT \leq \frac{1}{2} MD$, and we satisfy constraint (10). Lemma (3) now gives the desired list decodable code. □

4.2 Using an Explicit Extractor

As mentioned in [1], and shown in Sect. 5, in order to keep the rate strictly greater than zero, we need an extractor with degree $D = O(\log N)$. The best explicit construction to date of a strong extractor, which achieves the required degree is due to [11].

Fact 7 ([11]). *For Every $m = m(n)$, $k = k(n)$ and $\zeta = \zeta(n)$ such that $3m\sqrt{n \log(n/\zeta)} \leq k \leq n$, there is an explicit family of (k, ζ)-strong extractors $E_n : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d = \log n + O(\log \frac{m}{\zeta})$.*

Denoting $N = 2^n$, $K = 2^k$, $D = 2^d$, and $M = 2^m$, Plugging the above extractor in the construction, we prove Theorem 4:

Proof. Let $\epsilon > 0$, $T = 2^{polyloglog(\frac{1}{\epsilon})}$ as in Fact 6. Let C_1, C_2 be as in the proof of Theorem 2. Let C_3 be the constant behind the extractor degree $O(\cdot)$ notation from Fact 7, where $d = \log n + O(\log \frac{m}{\zeta})$.

Let $M = \frac{2 \cdot C_2}{\epsilon}$, $N > 2^{\frac{2T}{\epsilon \cdot (C_1+C_2) \cdot 2^{2 \cdot C_3} \cdot (\log M)^{C_3}}}$, such that $2^{\sqrt{\frac{\log N}{\log \log N}}} > M^6$. Let $K = M^6 \sqrt{\log N \log \log N}$, $D = 2^{2 \cdot C_3} \log N (\log M)^{C_3}$ and $L = \frac{C_1+C_2}{2} \cdot \frac{D}{\epsilon T}$. By Fact 7 and by the choice of N, K and M there is an explicit $(K, \frac{1}{4})$-strong extractor $f : [N] \times [D] \to [M]$. By the choice of N, $\frac{1}{\sqrt{L}} < \epsilon$ and there is an explicit $(\epsilon L, \frac{1}{4})$ disperser $g : [L] \times [T] \to [D]$. Finally, by the choice of M we have $LT \leq \frac{1}{2} MD$, and constraint (10) is satisfied. Applying Lemma 3 gives the desired code.

Encoding an element $x \in [N]$ is consisted of finding $\Gamma_f(x)$, and finding $\Gamma_g(l) = \{g(l, j) | j \in [T]\}$ for each $l \in [L]$. Thus, the explicitness of the code is straightforward from the explicitness of the disperser g and the extractor f above. □

5 On the Optimality of the Parameters Choice

We now take a closer look at the parameters. Specifically, we show that the parameters chosen in Sect. 3.2, which give good but sub optimal rate and alphabet size w.r.t. the non explicit construction, are the best possible in the above construction. The following claim summarizes the various relations between the parameters and their optimality.

Lemma 4. *In the construction given in Sect. 3, for any choice of parameters satisfying error rate of* $1 - \epsilon$ *the following holds:*

1. *Decoding list size of* $O(\frac{1}{\epsilon})$ *and rate bounded away from zero implies that the rate and alphabet size cannot be better (up to constant factor) than those in Sect. 3.2.*
2. *Decoding list size of* $O(\frac{1}{\epsilon})$ *implies disperser and extractor with optimal entropy loss (namely,* $\frac{HT}{D} = O(1)$ *and* $\frac{K}{M} = O(1)$*).*
3. *An almost optimal rate of* $O(\frac{\epsilon}{\log \frac{1}{\epsilon}})$ *implies extractor's degree* $D = O(\log N)$*.*
4. *Almost optimal rate of* $O(\frac{\epsilon}{\log \frac{1}{\epsilon}})$ *and* $|\Sigma| = 2^{O(\log^2 \frac{1}{\epsilon})}$ *imply an optimal entropy loss disperser (namely,* $\frac{H \cdot T}{D} = O(1)$*).*

For lack of space we give only the proof of 1.

Proof. Having an error rate of $1-\epsilon$, we have $H = \epsilon L$. We first show that M must be $\Theta(\frac{1}{\epsilon})$: Decoding list of size $\frac{1}{\epsilon}$ implies that $K = \frac{1}{\epsilon}$, and by (7) $M = O(K) = O(\frac{1}{\epsilon})$. By (10) $M \geq \frac{L \cdot T}{D}$. Since $L = \frac{H}{\epsilon}$, and since (9) implies $T = \Omega(\frac{D}{H})$, we have $M \geq \frac{L \cdot T}{D} = \Omega(\frac{1}{\epsilon})$. Altogether $M = \Theta(\frac{1}{\epsilon})$. By the construction, $rate = \frac{\log N}{L \cdot T \log M}$, (10) implies $L \cdot T \leq M \cdot D$, and so $rate \geq \frac{\log N}{M \cdot D \log M} = \Theta(\frac{\epsilon \log N}{D \log \frac{1}{\epsilon}})$. Thus, in order to bound the rate away from zero, we must take $D = O(\log N)$, which gives $rate = \Omega(\frac{\epsilon}{\log \frac{1}{\epsilon}})$. As for the alphabet size $|\Sigma| = M^T$. $M = \Omega(\frac{1}{\epsilon})$, and by (8) $T = \Omega(\log \frac{1}{\epsilon})$, thus, $|\Sigma| = (\frac{1}{\epsilon})^{\Omega(\log \frac{1}{\epsilon})}$ □

Acknowledgements

We would like to thank the anonymous referees for numerous comments which improved and clarified the final version of this paper a lot.

References

1. Guruswami, V.: Better extractors for better codes? In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM Press (2004) 436–444
2. Elias, P.: List decoding for noisy channels. In: 1957-IRE WESCON Convention Record, Pt. 2. (1957) 94–104
3. Sudan, M.: Lecture Notes on Algorithmic Introduction to Coding Theory. (http://theory.lcs.mit.edu/∼madhu/FT01/scribe/overall.ps)
4. Guruswami, V., Sudan, M.: Improved decoding of reed-solomon and algebraic-geometric codes. In: Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society (1998) 28
5. Ta-Shma, A., Zuckerman, D.: Extractor codes. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing. (2001) 193–199
6. Guruswami, V., Indyk, P.: Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, ACM Press (2002) 812–821
7. Alon, N., Bruck, J., Naor, J., Naor, M., Roth, R.: Construction of asymptotically good, low-rate error-correcting codes through pseudo-random graphs. IEEE Transactions on Information Theory **38** (1992) 509–516
8. Reingold, O., Vadhan, S., Wigderson, A.: Entropy waves, the zig-zag product, and new constant-degree expanders and extractors. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science. (2000)
9. Radhakrishnan, J., Ta-Shma, A.: Bounds for dispersers, extractors, and depth-two superconcentrators. SIAM Journal on Discrete Mathematics **13** (2000) 2–24
10. Alon, N., Spencer, J.H., Erdős, P.: The Probabilistic Method. Wiley–Interscience Series, John Wiley & Sons, Inc., New York (1992)
11. Ta-Shma, A., Zuckerman, D., Safra, S.: Extractors from Reed-Muller codes. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science. (2001) 638–647

The Power of Commuting with Finite Sets of Words

Michal Kunc*

Department of Mathematics, Masaryk University,
Janáčkovo nám. 2a, 662 95 Brno, Czech Republic
kunc@math.muni.cz
http://www.math.muni.cz/~kunc/

Abstract. We show that one can construct a finite language L such that the largest language commuting with L is not recursively enumerable. This gives a negative answer to the question raised by Conway in 1971 and also strongly disproves Conway's conjecture on context-freeness of maximal solutions of systems of semi-linear inequalities.

1 Introduction

In this paper we address the question whether the largest solution of any language equation of the form $XL = LX$ is regular provided L is a finite or regular language. It is known that in several algebraic structures related to algebras of formal languages, two elements commute if and only if they can be generated by the same element. For instance, two words commute if and only if they are powers of the same word (due to the defect theorem) and such a characterization was proved valid also for polynomials and formal series in non-commuting variables over a field (in particular, for multisets of words) by Bergman [2] and Cohn [4], respectively. But in the case of languages the situation is completely different.

Systems of language equations and inequalities were intensively studied especially in connection with context-free languages since these languages can be elegantly described as components of least solutions of systems of explicit polynomial equations, i.e. equations with the operations of union and concatenation. Much less attention was devoted to implicit language equations and particularly to their maximal solutions. Such issues were first addressed by Conway [5], who observed that inequalities of the form $E \subseteq L$, where E is a regular function of variables and L is a regular language, possess only finitely many maximal solutions, all of them are regular and computable. In particular, this leads to an algorithm for calculating best approximations of a given regular language by other given languages.

* Supported by the project MSM 143100009 of the Ministry of Education of the Czech Republic.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 569–580, 2005.

Regular solutions of systems of inequalities generalizing regular grammars were studied for example by Leiss [11]. Baader and Küsters [1] used largest solutions of systems of linear equations, i.e. equations of the form

$$K_0 \cup K_1X_1 \cup \cdots \cup K_nX_n = L_0 \cup L_1X_1 \cup \cdots \cup L_nX_n \ ,$$

where $K_0, \ldots, K_n, L_0, \ldots, L_n$ are regular languages, for dealing with unification of concept descriptions; they proved that the largest solution of each such system is regular and its computation is an ExpTime-complete problem. In [9] well quasi-orders of free monoids were used by the author to prove that all maximal solutions are regular for a large class of systems of inequalities where all constants are languages recognizable by finite simple semigroups. An attempt to initiate development of a unified theory of general language equations has been made by Okhotin; in particular, he considered maximal solutions of standard systems of equations defining context-free languages and related classes [14] and proved that recursive (recursively enumerable, co-recursively enumerable) languages are exactly languages definable as unique (smallest, largest) solutions of systems of implicit language inequalities using concatenation and all Boolean operations [13] and that each such language can be encoded even in a single explicit equation and precisely defined by a system of two explicit equations [15].

The problem of regularity of the largest language commuting with a given regular language was formulated already by Conway [5–p. 55 and 124] in 1971. There are actually two variants of the problem depending on whether we allow the resulting language to contain the empty word or not. The largest solution of the equation $XL = LX$ is denoted $\mathcal{C}(L)$ and its largest solution without the empty word is denoted $\mathcal{C}_+(L)$. The languages $\mathcal{C}(L)$ and $\mathcal{C}_+(L)$ are in general different and no direct relation between them (except for the obvious inclusion $\mathcal{C}_+(L) \subseteq \mathcal{C}(L)$) has been found yet. The problem was recently studied in several articles (e.g. [16, 3, 6]), but affirmative answers were given only for regular prefix codes [16] and at most ternary sets of words [6]. Moreover, it even remained an open problem whether the largest language commuting with a given finite set of words is recursive. On the other hand, as observed in [6], the complements of languages $\mathcal{C}(L)$ and $\mathcal{C}_+(L)$ are recursively enumerable provided the language L is recursive (actually, this is a special case of a general result about systems of language equations due to Okhotin [13]). A summary of known results concerning commutation of languages and some examples can be found in the recent survey [8].

In this paper we give the most negative possible answer to Conway's problem by showing that there exists a finite language L such that $\mathcal{C}(L)$ is not recursively enumerable. More precisely, we show that the complement of the language computed by an arbitrary Minsky machine can be encoded into a solution of a commutation equation. This contrasts with the fact that the largest solution of the inequality $XK \subseteq LX$ is regular provided the language L is regular, as demonstrated in [9]. We formulate our results for the case of languages $\mathcal{C}_+(L)$ too and further we show that for a regular language L, the difference $\mathcal{C}(L) \setminus \mathcal{C}_+(L)$ also need not be recursively enumerable, which answers a question posed in [7]. In

addition, our results disprove Conway's conjecture [5–p. 129] stating that every maximal solution of a system of so-called semi-linear inequalities is context-free.

The aim of this extended abstract is to demonstrate the ideas used in constructing a finite language L such that $\mathcal{C}(L)$ is not recursively enumerable. In Section 3 we deal with a simpler situation where L is only required to be a star-free language. First, we give an example of a star-free language L such that $\mathcal{C}(L)$ is non-regular and then we describe how the construction can be improved to show that $\mathcal{C}(L)$ even need not be recursively enumerable. Let us mention that this is in accord with the results obtained in [9]: complicated cases arise for star-free languages (or equivalently, languages recognizable by aperiodic monoids), whereas maximal solutions of such equations over languages recognizable by finite groups are always regular. Section 4 is devoted to the case of finite languages; by encoding the example from the beginning of Section 3 into finitely many words, we show that the language $\mathcal{C}(L)$ can be non-regular even for a finite language L. The techniques of Sections 3 and 4 can then be combined to obtain a finite set of words L such that $\mathcal{C}(L)$ is not recursively enumerable. The final construction, which is not included in this presentation, as well as detailed proofs of the results can be found in the full version of the paper [10].

Basic notions employed in our considerations are recalled in the following section. For a more comprehensive introduction to formal languages the reader is referred to [17].

2 Preliminaries

We denote the sets of positive and non-negative integers by $\mathbb{N}$ and $\mathbb{N}_0$, respectively. Throughout the paper we consider a finite alphabet A. As usual, we write A^+ for the set of all non-empty finite words over A, and A^* for the set obtained from A^+ by adding the empty word ε. A word $u \in A^*$ is called a *factor* of $v \in A^*$ if $v = wu\hat{w}$ for some words $w, \hat{w} \in A^*$; it is called a *prefix* (*suffix*) of v if $v = uw$ ($v = wu$, respectively) for some $w \in A^*$.

Languages over the alphabet A are arbitrary subsets of A^* and we say that a language $L \subseteq A^*$ is ε-free if $\varepsilon \notin L$. The basic operation on languages is concatenation defined by the rule $K \cdot L = \{uv \mid u \in K,\ v \in L\}$ and we use the standard notation $L^+ = \bigcup_{m \in \mathbb{N}} L^m$ and $L^* = L^+ \cup \{\varepsilon\}$. *Regular* languages are languages definable by finite automata, or equivalently, by rational expressions. The basic tool for proving non-regularity of languages is the well-known pumping lemma (see e.g. [17]). A language $L \subseteq A^*$ is called *star-free* if it can be obtained from finite languages using the operations of finite union, complementation and concatenation; in particular, for every $B \subseteq A$, the languages B^+ and B^* are star-free.

For every language L over A we denote by $\mathcal{C}(L)$ the largest language over A which commutes with L and by $\mathcal{C}_+(L)$ the largest ε-free language over A which commutes with L. Such languages $\mathcal{C}(L)$ and $\mathcal{C}_+(L)$ certainly exist for every language L since the union of arbitrarily many languages commuting with L commutes with L as well. We clearly always have $\mathcal{C}_+(L) \subseteq \mathcal{C}(L)$, $L^* \subseteq \mathcal{C}(L)$ and

$L^+ \subseteq \mathcal{C}_+(L)$. Further, the languages $\mathcal{C}(L)$ and $\mathcal{C}_+(L)$ are easily seen to be closed under concatenation and so they form a submonoid and a subsemigroup of the free monoid A^*, respectively. Another interesting property of the languages $\mathcal{C}(L)$ and $\mathcal{C}_+(L)$ is that they remain unchanged when we replace L with its closure under concatenation, i.e. $\mathcal{C}(L) = \mathcal{C}(L^+)$ and $\mathcal{C}_+(L) = \mathcal{C}_+(L^+)$.

3 Star-Free Languages

The aim of this section is to construct a star-free language L such that the largest solution of the equation $XL = LX$ is not recursively enumerable. This is achieved by encoding an arbitrary Minsky machine $\mathcal{M}$ into a star-free language L in such a way that $\mathcal{C}(L) \cap uv^*w = \{uv^nw \mid n \notin \mathcal{L}(\mathcal{M})\}$, where u, v and w are certain words and $\mathcal{L}(\mathcal{M}) \subseteq \mathbb{N}_0$ is the set computed by the machine $\mathcal{M}$. Because the construction is rather technical, let us first present it in a simplified form which shows that the largest language commuting with a given star-free language need not be regular.

Example 1. Let us take the alphabet $A = \{a, b, c, e, \hat{e}, f, \hat{f}, g, \hat{g}\}$. We consider auxiliary languages

$$M = efga^+ba^* \cup ga^*ba^*\hat{g}\hat{f} \cup a^*ba^*\hat{g}\hat{f}\hat{e} \cup fga^*ba^*\hat{g} \ ,$$
$$L_0 = (A \setminus \{c\})^*b(A \setminus \{c\})^* \setminus (\{efg, fg, g, \varepsilon\} \cdot a^*ba^* \cdot \{\varepsilon, \hat{g}, \hat{g}\hat{f}, \hat{g}\hat{f}\hat{e}\})$$

and define a star-free language

$$L = \{c, ef, ga, e, fg, \hat{f}\hat{e}, a\hat{g}, \hat{e}, \hat{g}\hat{f}, fgba\hat{g}\} \cup A^*bA^*bA^* \cup L_0 \cup cM \cup Mc \ .$$

With the aim of verifying that the language $\mathcal{C}(L)$ is not regular, let us first observe that $fga^mba^n\hat{g}\hat{f} \notin \mathcal{C}(L)$ for every $m, n \in \mathbb{N}_0$. Indeed, assuming the converse, we obtain $c^2fga^mba^n\hat{g}\hat{f} \in L^2 \cdot \mathcal{C}(L) = \mathcal{C}(L) \cdot L^2$, which is a contradiction because this word has no suffix belonging to L^2.

Now we are going to show that $efga^mba^n \notin \mathcal{C}(L)$ for every $m, n \in \mathbb{N}_0$ satisfying $m < n$. We proceed by induction on m. If $m = 0$ then the converse of this fact implies $cefgba^n \in L \cdot \mathcal{C}(L) = \mathcal{C}(L) \cdot L$, which is impossible as no suffix of the word $cefgba^n$ lies in L. Let $m \geq 1$ and suppose we already know that $efga^{m-1}ba^{n-1} \notin \mathcal{C}(L)$ holds. By way of contradiction, assume $efga^mba^n \in \mathcal{C}(L)$. Then $efga^mba^n\hat{g}\hat{f} \in \mathcal{C}(L) \cdot L = L \cdot \mathcal{C}(L)$. Since there are just two prefixes of $efga^mba^n\hat{g}\hat{f}$ which belong to L, we have either $fga^mba^n\hat{g}\hat{f} \in \mathcal{C}(L)$ or $ga^mba^n\hat{g}\hat{f} \in \mathcal{C}(L)$. Due to our initial observation, the former case is false. From the latter one we deduce $ga^mba^n\hat{g}\hat{f}\hat{e} \in \mathcal{C}(L) \cdot L = L \cdot \mathcal{C}(L)$. This immediately gives $a^{m-1}ba^n\hat{g}\hat{f}\hat{e} \in \mathcal{C}(L)$ and therefore $fga^{m-1}ba^n\hat{g}\hat{f}\hat{e} \in L \cdot \mathcal{C}(L) = \mathcal{C}(L) \cdot L$. The word $fga^{m-1}ba^n\hat{g}\hat{f}\hat{e}$ has exactly two suffixes from L and repeating the previous argument we get $fga^{m-1}ba^n\hat{g} \in \mathcal{C}(L)$. Consequently $efga^{m-1}ba^n\hat{g} \in L \cdot \mathcal{C}(L) = \mathcal{C}(L) \cdot L$, and since we have $n \geq 2$, this finally leads to $efga^{m-1}ba^{n-1} \in \mathcal{C}(L)$, contradicting our assumption.

On the other hand, the language

$$K = L^* \cup \{efga^nba^n, ga^nba^n\hat{g}\hat{f}, a^{n-1}ba^n\hat{g}\hat{f}\hat{e}, fga^{n-1}ba^n\hat{g} \mid n \in \mathbb{N}\}$$

commutes with L because one can easily calculate that both products KL and LK are equal to the language

$$L^+ \cup \{efga^nba^n\hat{g}\hat{f}, ga^nba^n\hat{g}\hat{f}\hat{e}, fga^{n-1}ba^n\hat{g}\hat{f}\hat{e}, efga^{n-1}ba^n\hat{g} \mid n \in \mathbb{N}\} \ ;$$

notice that this holds due to the fact $efgba\hat{g} \in L^2$. Therefore the word $efga^nba^n$ belongs to $\mathcal{C}(L)$ for all $n \in \mathbb{N}$.

Altogether, we have demonstrated both $efga^nba^n \in \mathcal{C}(L)$ for $n \in \mathbb{N}$ and $efga^mba^n \notin \mathcal{C}(L)$ for $m < n$, hence the language $\mathcal{C}(L)$ cannot be regular due to the pumping lemma.

Notice that in the previous example we have in fact encoded into the language L two counters (as powers of a) and the operation of simultaneous decrementation of both counters together with testing whether both counters are equal to zero. The following construction of a language L such that $\mathcal{C}(L)$ is not recursively enumerable is based essentially on the same idea.

Theorem 1. *There exists a star-free language L such that*

(i) the largest language commuting with L is not recursively enumerable.
(ii) the difference between the largest language commuting with L and the largest ε-free language commuting with L is not recursively enumerable.

Sketch of the proof. Let $\mathcal{M}$ be a Minsky machine [12] which computes a non-recursive set of non-negative integers. The machine consists of two counters and a finite set of states Q, which is a disjoint union

$$Q = T_1 \cup T_2 \cup I_1 \cup I_2 \cup D_1 \cup D_2 \cup \{1\} \ ,$$

where 1 is the terminal state. We assume that the initial state 0 of $\mathcal{M}$ belongs to I_1. A configuration of the machine is a triple (i, m, n), where $i \in Q$ and $m, n \in \mathbb{N}_0$ are the values stored in the counters. The step performed by the machine in a given state is determined by the instruction associated with this state:

- From the state $i \in T_k$, $k \in \{1, 2\}$, the machine goes to the state $\tau_0(i)$ if the counter number k is empty and to the state $\tau_1(i)$ otherwise, where $\tau_0(i) \neq i$ and $\tau_1(i) \neq i$ are distinct states.
- When the machine is in the state $i \in I_k$ (or $i \in D_k$), it increments (decrements, respectively) the counter number k and goes to the state $\tau(i) \neq i$.
- When the machine reaches the state 1, the computation stops.

Initial configurations of $\mathcal{M}$ are configurations of the form $(0, 0, n)$, $n \in \mathbb{N}_0$. The machine computes the set $\mathcal{L}(\mathcal{M}) \subseteq \mathbb{N}_0$ of all numbers n such that the machine stops in the configuration $(1, 0, n)$ starting from some initial configuration. Since

we have chosen $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M})$ is not recursive, its complement $\mathbb{N}_0 \setminus \mathcal{L}(\mathcal{M})$ is not recursively enumerable. In addition, let us assume that the initial state 0 of $\mathcal{M}$ cannot be reached from the other states and if for some $i, j \in Q$ either $\tau(i) = j$ or $\tau_0(i) = j$ or $\tau_1(i) = j$, then neither $\tau(j) = i$ nor $\tau_0(j) = i$ nor $\tau_1(j) = i$, i.e. given the two states involved in one step of a computation, it is uniquely determined which of these steps is the original one and which is the resulting one.

Consider the alphabet

$$A = \{a, b, c\} \cup \{d_i \mid i \in Q\} \cup \{e_i, f_i, g_i \mid i \in I_1 \cup I_2 \cup D_1 \cup D_2\} .$$

Every configuration (i, m, n) of the machine will be represented by the word $a^{m+1}ba^{n+1}d_i^2$. If a configuration is reachable from some initial configuration, then the corresponding word should not belong to $\mathcal{C}(L)$. Since in our encoding using commutation of languages we have no means of directing a computation, the same will hold also for all configurations from which some configuration reachable from an initial configuration is eventually obtained.

In order to construct the language L, we introduce several auxiliary languages first. For each state $i \in Q$, we consider a language M_i which can be used to move occurrences of the letter d_i from one side of a word to the other and in this way enables us to link steps of a computation modifying different counters: let $M_0' = a^+aba^+d_0^2$ and for $i \in Q \setminus \{0\}$ let

$$M_i' = d_i^2a^+ba^+ \cup d_ia^+ba^+d_i \cup a^+ba^+d_i^2 ;$$

then define $M_i = cM_i' \cup M_i'c$ for every $i \in Q$.

The following languages N_i describe words appearing during manipulations corresponding to performing the instruction associated with the state i; in the case of $i \in I_1 \cup I_2 \cup D_1 \cup D_2$, there are also four short words whose addition and removal transforms one of these words into another.

For $i \in T_1 \cup T_2$ let

$$N_i' = d_iaba^+d_{\tau_0(i)} \cup d_ia^+aba^+d_{\tau_1(i)}$$

and define

$$N_i = cN_i' \cup N_i'c \cup A^*d_ia^+aba^+d_{\tau_0(i)}A^* \cup A^*d_iaba^+d_{\tau_1(i)}A^* .$$

For $i \in I_1 \cup I_2$ let

$$N_i' = g_ia^+ba^+d_i \cup e_if_ig_ia^+ba^+ \cup f_ig_ia^+ba^+d_{\tau(i)}$$

and define

$$\begin{aligned} N_i = {} & cN_i' \cup N_i'c \cup \{e_if_i, g_ia, e_i, f_ig_i\} \\ & \cup f_ig_ia^+ba^+d_i^2 \cup g_ia^+ba^+d_{\tau(i)}^2 \cup ba^+d_i^2 . \end{aligned}$$

And dually, for $i \in D_1 \cup D_2$ let

$$N_i' = g_ia^+ba^+d_{\tau(i)} \cup e_if_ig_ia^+ba^+ \cup f_ig_ia^+ba^+d_i$$

and define

$$\begin{aligned} N_i = {} & cN_i' \cup N_i'c \cup \{e_if_i, g_ia, e_i, f_ig_i\} \\ & \cup f_ig_ia^+ba^+d_{\tau(i)}^2 \cup g_ia^+ba^+d_i^2 \cup ba^+d_{\tau(i)}^2 \ . \end{aligned}$$

The next language describes possible consecutive states:

$$\begin{aligned} L' = {} & \bigcup \{\{d_i^2, d_i, \varepsilon\} \cdot a^*ba^* \cdot \{\varepsilon, d_i, d_i^2\} \mid i \in Q,\ i \neq 0\} \\ & \cup \bigcup \{\{d_i^2, d_i, \varepsilon\} \cdot a^*ba^* \cdot \{\varepsilon, d_{\tau_1(i)}, d_{\tau_1(i)}^2, d_{\tau_0(i)}, d_{\tau_0(i)}^2\} \mid i \in T_1\} \\ & \cup \bigcup \{\{d_{\tau_0(i)}^2, d_{\tau_0(i)}, d_{\tau_1(i)}^2, d_{\tau_1(i)}, \varepsilon\} \cdot a^*ba^* \cdot \{\varepsilon, d_i, d_i^2\} \mid i \in T_2\} \\ & \cup \bigcup \{\{e_if_ig_i, f_ig_i, g_i, \varepsilon\} \cdot a^*ba^* \cdot \{\varepsilon, d_{\tau(i)}, d_{\tau(i)}^2, d_i, d_i^2\} \mid i \in I_1 \cup D_1\} \\ & \cup \bigcup \{\{d_i^2, d_i, d_{\tau(i)}^2, d_{\tau(i)}, \varepsilon\} \cdot a^*ba^* \cdot \{\varepsilon, g_i, g_if_i, g_if_ie_i\} \mid i \in I_2 \cup D_2\} \ . \end{aligned}$$

Finally we define a star-free language

$$\begin{aligned} L = {} & \{d_i \mid i \in Q\} \cup \{c\} \cup A^*bA^*bA^* \\ & \cup (A \setminus \{c\})^*b(A \setminus \{c\})^* \setminus L' \\ & \cup \bigcup \{M_i \mid i \in Q\} \\ & \cup \bigcup \{N_i \mid i \in T_1 \cup I_1 \cup D_1\} \\ & \cup \bigcup \{\mathrm{rev}(N_i) \mid i \in T_2 \cup I_2 \cup D_2\} \ , \end{aligned}$$

where $\mathrm{rev}(N_i)$ denotes the reverse of the language N_i.

Let us briefly sketch out how computations of the Minsky machine are simulated by manipulations of words from A^+. If some word u obtained during our manipulations is multiplied from some side (say from the left) by a word $v \in L$, one usually gets a word from L with just few exceptions corresponding to correct computations of the machine. Such exceptional products vu then usually have only one suffix v' belonging to L whose removal does produce a word u' which can potentially belong to $\mathcal{C}(L)$. In this way we achieve that u lies in $\mathcal{C}(L)$ if and only if u' lies in $\mathcal{C}(L)$. Therefore every computation of the machine $\mathcal{M}$ preserves the properties that the word corresponding to a configuration belongs, or does not belong, to $\mathcal{C}(L)$.

For instance, starting from a word $u = a^{m+1}ba^{n+1}d_i^2$ corresponding to the state $i \in I_1$, we can multiply this word by $g_ia \in L$ from the left and then by removing the word $d_i \in L$ from the right we get the word $u' = g_ia^{m+2}ba^{n+1}d_i$, which is in $\mathcal{C}(L)$ if and only if u is in $\mathcal{C}(L)$. Now the first counter is already incremented in the word u' and the occurrence of the letter g_i on the left ensures that the following manipulations of u' either return us back to u (if we multiply by d_i from the right) or continue with simulating the step of $\mathcal{M}$ performed in the state i by adding the word e_if_i to the left, removing the remaining occurrence

of d_i on the right, and then multiplying by $d_{\tau(i)}$ from the right, eventually reaching the word $a^{m+2}ba^{n+1}d_{\tau(i)}^2$.

The language L' describes which words are allowed to occur during these manipulations. Each of these words consists of one occurrence of b in the middle surrounded by several occurrences of a and a block of letters corresponding to some state of $\mathcal{M}$ on each side; notice that on each side of a word all letters have the same indices. Those words which do not correspond to a correct computation of $\mathcal{M}$, but can be obtained from such a word by concatenating with a word from L, are included in L.

On the other hand, concatenating with the word $c \in L$ can be used to show that the only eligible results of removing a suffix or a prefix belonging to L are words from the languages M_i' and N_i'. In particular, the letter c serves for proving that all words corresponding to initial configurations do not belong to $\mathcal{C}(L)$, which is consequently true also for all configurations reachable from initial configurations; this is achieved by not including the words from the language $aba^+d_0^2$ into M_0'.

Restating the previously described ideas formally, in order to show that both sets $\mathcal{C}(L)$ and $\mathcal{C}(L) \setminus \mathcal{C}_+(L)$ are not recursively enumerable, it is enough to verify that the equivalences

$$n \notin \mathcal{L}(\mathcal{M}) \iff aba^{n+1}d_1^2 \in \mathcal{C}(L) \iff aba^{n+1}d_1^2 \in \mathcal{C}(L) \setminus \mathcal{C}_+(L) \qquad (1)$$

hold for every non-negative integer $n \in \mathbb{N}_0$.

First notice that if some word of the form $aba^{n+1}d_1^2$ belongs to $\mathcal{C}_+(L)$, then

$$aba^{n+1}d_1^2c \in \mathcal{C}_+(L) \cdot L = L \cdot \mathcal{C}_+(L) \ ,$$

which is impossible since $aba^{n+1}d_1^2c \notin LA^+$. Therefore the second condition of (1) is equivalent to the third one.

The verification of the equivalence of the first two conditions of (1) proceeds similarly to Example 1. The converse implication is obtained by proving $a^{m+1}ba^{n+1}d_i^2 \notin \mathcal{C}(L)$ for every configuration (i, m, n) reachable from some initial configuration, which is achieved by induction with respect to the length of a run of $\mathcal{M}$ reaching (i, m, n) from an initial configuration.

In order to verify the direct implication, we consider the set C of all configurations (i, m, n) of $\mathcal{M}$ such that there is no computation of $\mathcal{M}$ transforming (i, m, n) to a configuration reachable from some initial configuration. We use this set to construct a language K which commutes with L and contains the word $aba^{n+1}d_1^2$ for all $n \notin \mathcal{L}(\mathcal{M})$. In the definition of K we employ the same auxiliary languages as in the definition of L. For $m, n \in \mathbb{N}_0$, we denote

$$K_{m,n} = (A \setminus \{a\})^* a^{m+1} b a^{n+1} (A \setminus \{a\})^*$$

and define

$$K = L^* \cup \bigcup \{M_i' \cap K_{m,n} \mid (i, m, n) \in C\}$$
$$\cup \bigcup \{N_i' \cap K_{m,n} \mid i \in T_1 \cup D_1,\ (i, m, n) \in C\}$$

$$\cup \bigcup \{N'_i \cap K_{m,n} \mid i \in I_1,\ (\tau(i), m, n) \in C\}$$
$$\cup \bigcup \{\mathrm{rev}(N'_i) \cap K_{m,n} \mid i \in T_2 \cup D_2,\ (i, m, n) \in C\}$$
$$\cup \bigcup \{\mathrm{rev}(N'_i) \cap K_{m,n} \mid i \in I_2,\ (\tau(i), m, n) \in C\}\ .$$

Because the languages K and L can be proved to commute, we have $K \subseteq \mathcal{C}(L)$. And since $aba^{n+1}d_1^2 \in M'_1 \cap K_{0,n} \subseteq K$ holds for every $n \notin \mathcal{L}(\mathcal{M})$, we can immediately deduce $aba^{n+1}d_1^2 \in \mathcal{C}(L)$. □

The following lemma establishes a connection between largest solutions and largest ε-free solutions of commutation equations. Namely, we construct for each language L a language $\hat{L}$ such that the largest language commuting with L can be easily reconstructed from the largest ε-free language commuting with $\hat{L}$. The proof of the lemma consists of a direct verification and therefore it is omitted.

Lemma 1. *Let L be an arbitrary ε-free language over an alphabet A and let $\# \notin A$ be a new letter. Consider the alphabet $\hat{A} = A \cup \{\#\}$ and injective homomorphisms $\lambda, \rho\colon A^* \to \hat{A}^*$ defined by the rules $\lambda(a) = \#a$ and $\rho(a) = a\#$ for every $a \in A$. Then the language*

$$\hat{L} = \lambda(L) \cup \rho(L) \cup \{\#\} \cup \hat{A}^* \cdot \#^2 \cdot \hat{A}^*$$

over $\hat{A}$ satisfies

$$\mathcal{C}_+(\hat{L}) = \lambda(\mathcal{C}(L)) \cdot \# \cup \hat{L}^+\ .$$

In particular, since $\hat{L}^+ \cap \lambda(A^*) \cdot \# = \lambda(L^*) \cdot \#$, this implies

$$\mathcal{C}_+(\hat{L}) \cap \lambda(A^*) \cdot \# = \lambda(\mathcal{C}(L)) \cdot \#\ .$$

Now we apply this lemma to the language L constructed in Theorem 1 in order to obtain a star-free language $\hat{L}$ such that $\mathcal{C}_+(\hat{L})$ is not recursively enumerable.

Theorem 2. *There exists a star-free language $\hat{L}$ such that the largest ε-free language commuting with $\hat{L}$ is not recursively enumerable.*

Proof. Theorem 1 provides us with a star-free language L such that $\mathcal{C}(L)$ is not recursively enumerable. Let $\hat{L}$ be the language obtained from L by means of Lemma 1. Then a word $u \in A^*$ belongs to $\mathcal{C}(L)$ if and only if the word $\lambda(u)\#$ belongs to $\mathcal{C}_+(\hat{L})$. This implies that the language $\mathcal{C}_+(\hat{L})$ is not recursively enumerable. And since the image of any star-free language under an injective homomorphism is star-free, it is clear that $\hat{L}$ is star-free as required. □

4 Finite Languages

In this section we show how the construction described in the previous section can be improved to obtain a finite set of words L such that both languages $\mathcal{C}(L)$

and $\mathcal{C}_+(L)$ are not recursively enumerable. Because the techniques needed in the case of finite languages are more complicated, we present here only a construction demonstrating that $\mathcal{C}(L)$ and $\mathcal{C}_+(L)$ need not be regular.

Theorem 3. *There exists a finite language L such that neither the largest language commuting with L nor the largest ε-free language commuting with L is regular.*

Sketch of the proof. The basic idea of the proof is to encode the language L defined in Example 1 into a finite set of words. We achieve this by introducing a new letter s to be used for encoding states of a finite automaton recognizing the language $cM \cup Mc$ of Example 1. With this aim, consider the alphabet $A = \{a, \hat{a}, b, c, e, \hat{e}, f, \hat{f}, g, \hat{g}, s\}$ and let us denote by B its subset $A \setminus \{s\}$. Further, let $\varphi\colon B^* \to A^*$ be the homomorphism defined by the formula $\varphi(x) = sxs^{17}$ for all $x \in B$.

First we define an auxiliary language

$$L'_9 = \{ef, fg, ga, aa, ab, b\hat{a}, \hat{a}\hat{a}, \hat{a}\hat{g}, \hat{g}\hat{f}, \hat{f}\hat{e}\} \ ,$$

which describes pairs of neighbouring letters in the language

$$\{efg, fg, g, \varepsilon\} \cdot a^+ b\hat{a}^+ \cdot \{\varepsilon, \hat{g}, \hat{g}\hat{f}, \hat{g}\hat{f}\hat{e}\}$$

similar to the one used in Example 1. Let $\hat{L} = s^{\leq 18} \cdot B \cdot s^{\leq 18} \cup s^{\leq 18}$, where $s^{\leq 18} = \{\varepsilon, s, s^2, \ldots, s^{18}\}$, and let the language L be the union of the following languages L_k for $k = 0, \ldots, 14$:

$$\begin{aligned}
L_0 &= \varphi(\{c, ef, ga, e, fg, \hat{f}\hat{e}, \hat{a}\hat{g}, \hat{e}, \hat{g}\hat{f}, fgab\hat{a}^2\hat{g}\}) \ , \\
L_1 &= \{scs^{18}es^{18}fs^{18}gs^{18}as^{18}as^{16}, s^2as^{16}, s^2bs^{15}, s^3\hat{a}s^{15}, s^3\hat{a}s^{17}\} \ , \\
L_2 &= \{scs^{18}gs^{14}, s^4as^{14}, s^4bs^{13}, s^5\hat{a}s^{13}, s^5\hat{g}s^{18}\hat{f}s^{17}\} \ , \\
L_3 &= \{scs^{18}as^{12}, s^6as^{12}, s^6bs^{11}, s^7\hat{a}s^{11}, s^7\hat{g}s^{18}\hat{f}s^{18}\hat{e}s^{17}\} \ , \\
L_4 &= \{scs^{18}fs^{18}gs^{10}, s^8as^{10}, s^8bs^9, s^9\hat{a}s^9, s^9\hat{g}s^{17}\} \ , \\
L_5 &= \{ses^{18}fs^{18}gs^{18}as^{18}as^8, s^{10}as^8, s^{10}bs^7, s^{11}\hat{a}s^7, s^{11}\hat{a}s^{18}cs^{17}\} \ , \\
L_6 &= \{sgs^6, s^{12}as^6, s^{12}bs^5, s^{13}\hat{a}s^5, s^{13}\hat{g}s^{18}\hat{f}s^{18}cs^{17}\} \ , \\
L_7 &= \{sas^4, s^{14}as^4, s^{14}bs^3, s^{15}\hat{a}s^3, s^{15}\hat{g}s^{18}\hat{f}s^{18}\hat{e}s^{18}cs^{17}\} \ , \\
L_8 &= \{sfs^{18}gs^2, s^{16}as^2, s^{16}bs, s^{17}\hat{a}s, s^{17}\hat{g}s^{18}cs^{17}\} \ , \\
L_9 &= \hat{L} \cdot \{xs^{18}y \mid x, y \in B \setminus \{c\},\ xy \notin L'_9\} \cdot \hat{L} \ , \\
L_{10} &= \hat{L} \cdot (B \setminus \{c\}) \cdot s^{18}cs^{18} \cdot (B \setminus \{c\}) \cdot \hat{L} \ , \\
L_{11} &= \hat{L} \cdot s^{19} \cdot \hat{L} \ , \\
L_{12} &= \hat{L} \cdot B \cdot s^{\leq 17} \cdot B \cdot \hat{L} \ , \\
L_{13} &= (s^{\leq 18} \cdot B)^2 \cup (s^{\leq 18} \cdot B)^3 \ , \\
L_{14} &= (B \cdot s^{\leq 18})^2 \cup (B \cdot s^{\leq 18})^3 \ .
\end{aligned}$$

In essence, instead of the words that we used in Example 1, we work here with their φ-images. The languages L_{11} through L_{14} ensure that the language L^+ contains all words which do not belong to $\varphi(B^+) \cup s^{\leq 18}$, i.e. which are neither of the form $sb_1s^{18}b_2s^{18}\cdots s^{18}b_ks^{17}$, for $k \in \mathbb{N}$ and $b_1, \ldots, b_k \in B$, nor s^i, for $0 \leq i \leq 18$. More precisely, every word which possesses a factor belonging to one of the languages L_{11} and L_{12} lies in L^+ because this factor can be multiplied by appropriate words from the languages L_{13} from the left and L_{14} from the right (possibly with some words from L_{11} inserted in order to build blocks of occurrences of s of length more than 18). In this way we include in L^+ all words containing more than 18 consecutive occurrences of s and all words with less than 18 occurrences of s between two letters from B. Similarly, we deal with certain sequences of letters from B: factors corresponding to the pairs described by the set L'_9 are determined by the language L_9 and the language L_{10} serves for dealing with occurrences of c between letters from $B \setminus \{c\}$.

Notice that both languages L_{13} and L_{14} contain only words with at least two occurrences of letters from B. This enables us to apply the same argument that we used in Example 1 where certain words were proved not to belong to $\mathcal{C}(L)$ by concatenating with $c \in L$; the role of this letter is played by the word $\varphi(c)$ in our encoding and the argument works well only if we ensure that multiplication by $\varphi(c)$ on one side of a word u cannot be compensated by removing a word of the same length as $\varphi(c)$ from the other side of u. This restriction on the languages L_{13} and L_{14} is also the reason for introducing the language $\hat{L}$ to both sides of the languages L_9 through L_{12}; this language takes care of those words where the factor under consideration is too close to a margin of the word.

The finite set of words from the definition of the language L in Example 1 is encoded in L_0 simply by taking the φ-images. And for each part of the language $cM \cup Mc$ of Example 1, all of its elements are decomposed into finitely many common segments. Then we ensure that these segments can be put together only in the appropriate order by choosing in the definition of languages L_1 through L_8 several different powers of s; if incompatible segments are concatenated, a block of occurrences of s of length different from 18 is produced and the resulting word cannot belong to $\varphi(B^+)$.

Finally, the reason why the language $\mathcal{C}_+(L)$ is also non-regular is that every word $u \in \mathcal{C}(L) \setminus L^*$ used in the proof is long enough to guarantee that the languages $L \cdot u$ and $u \cdot L$ contain no words from L and so the empty word is not essential in $\mathcal{C}(L)$. □

Theorems 1 and 3 state that there exist a star-free language L such that $\mathcal{C}(L)$ is not recursively enumerable and a finite language L such that $\mathcal{C}(L)$ is not regular, respectively. Actually, the constructions of Theorems 1 and 3 can be combined to prove that even for a finite language L, none of the languages $\mathcal{C}(L)$ and $\mathcal{C}_+(L)$ need to be recursively enumerable:

Theorem 4. *There exists a finite language L such that neither the largest language commuting with L nor the largest ε-free language commuting with L is recursively enumerable.*

Acknowledgement

I am grateful to Ondřej Klíma for several useful comments.

References

1. Baader, F., Küsters, R.: Unification in a description logic with transitive closure of roles. In *Proc. LPAR 2001*, LNCS 2250, Springer (2001) 217–232.
2. Bergman, G.M.: Centralizers in free associative algebras. *Trans. Amer. Math. Soc.* **137** (1969) 327–344.
3. Choffrut, C., Karhumäki, J., Ollinger, N.: The commutation of finite sets: A challenging problem. *Theoret. Comput. Sci.* **273** (2002) 69–79.
4. Cohn, P.M.: Factorization in non-commutative power series rings. *Proc. Cambridge Philos. Soc.* **58** (1962) 452–464.
5. Conway, J.H.: *Regular Algebra and Finite Machines*. Chapman and Hall (1971).
6. Karhumäki, J., Petre, I.: Conway's problem for three-word sets. *Theoret. Comput. Sci.* **289** (2002) 705–725.
7. Karhumäki, J., Petre, I.: The branching point approach to Conway's problem. In *Formal and Natural Computing*, LNCS 2300, Springer (2002) 69–76.
8. Karhumäki, J., Petre, I.: Two problems on commutation of languages. In *Current Trends in Theoretical Computer Science, The Challenge of the New Century, vol. 2*, World Scientific (2004) 477–494.
9. Kunc, M.: Regular solutions of language inequalities and well quasi-orders. *Theoret. Comput. Sci.* (to appear). Extended abstract in *Proc. ICALP 2004*, LNCS 3142, Springer (2004) 870–881.
10. Kunc, M.: The power of commuting with finite sets of words. Manuscript (2004). Available at `http://www.math.muni.cz/~kunc/`
11. Leiss, E.L.: *Language Equations*. Springer (1999).
12. Minsky, M.L.: *Computation: Finite and Infinite Machines*. Prentice-Hall (1967).
13. Okhotin, A.: Decision problems for language equations. Submitted for publication (2003). Available at `http://www.cs.queensu.ca/home/okhotin/`
14. Okhotin, A.: Greatest solutions of language equations. Submitted for publication (2003). Available at `http://www.cs.queensu.ca/home/okhotin/`
15. Okhotin, A.: On computational universality in language equations. In *Proc. MCU 2004*, LNCS 3354, Springer (to appear).
16. Ratoandromanana, B.: Codes et motifs. *RAIRO Inform. Théor. Appl.* **23** (1989) 425–444.
17. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*. Springer (1997).

Exact Quantum Algorithms for the Leader Election Problem

Seiichiro Tani[1,2], Hirotada Kobayashi[2], and Keiji Matsumoto[2,3]

[1] NTT Communication Science Laboratories, NTT Corporation,
3-1 Morinosato-Wakamiya, Atsugi, Kanagawa 243-0198, Japan
tani@theory.brl.ntt.co.jp
[2] Quantum Computation and Information Project, ERATO,
Japan Science and Technology Agency,
5-28-3 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
hirotada@qci.jst.go.jp
[3] Foundations of Information Research Division,
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
keiji@nii.ac.jp

Abstract. It is well-known that no classical algorithm can solve exactly (i.e., in bounded time without error) the leader election problem in anonymous networks. This paper gives two quantum algorithms that, when the parties are connected by quantum communication links, can exactly solve the problem for any network topology in polynomial rounds and polynomial communication/time complexity with respect to the number of parties. Our algorithms work well even in the case where only the upper bound of the number of parties is given.

1 Introduction

Quantum computation and communication are turning out to be much more powerful than the classical equivalents in various computational tasks. Perhaps the most exciting developments in quantum computation would be polynomial-time quantum algorithms for factoring integers and computing discrete logarithms [14], and the most remarkable ones in quantum communication would be quantum key distribution protocols [5, 4] that have been proved to be unconditionally secure [12, 15, 16]. Many other algorithms and protocols have been proposed to show the strength of quantum computation and communication, such as cryptographic results (e.g., [9, 8, 1]) and communication complexity results (e.g., [13, 7, 3]). This paper sheds light on another significant superiority of quantum computing over the classical equivalent in the setting of traditional distributed computing.

The leader election problem is a core problem in traditional distributed computing in the sense that, once it is solved, it becomes possible to efficiently solve many substantial problems in distributed computing (see, e.g., [11]). The goal of the leader election problem is to elect a unique leader from among distributed parties. Obviously, it is possible to deterministically elect the unique

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 581–592, 2005.

leader when each party has a unique identifier, and many classical deterministic algorithms with this assumption have been proposed. As the number of parties grows, however, it becomes difficult to preserve the uniqueness of the identifiers. Thus, other studies have examined the cases wherein each party is anonymous, i.e., each party has the same identifier [2, 10, 17, 18], as an extreme case. In this setting, no classical exact algorithm (i.e., an algorithm that runs in bounded time and solves the problem with zero error) exists for a broad class of network topologies including regular graphs, even if the network topology (and thus the number of parties) is known to each party prior to algorithm invocation [17]. Moreover, to the best of our knowledge, no zero-error probabilistic algorithm is known that works for *any* topology and runs in time/communication expected polynomial in the number of parties. Here, and throughout this paper, we denote by time complexity the maximum number of steps, including steps for the local computation, necessary for each party to execute the protocol, where the maximum is taken over all parties. In synchronous networks, the number of simultaneous message passing is also an important measure. Each turn of simultaneous message passing is referred to as a *round*.

This paper considers the model in which the network is anonymous and consists of quantum links, and proposes two exact quantum algorithms both of which elect a unique leader from among n parties in polynomial time for *any* topology of synchronous networks. Our first algorithm is simple and runs in $O(n^3)$ time. The total communication complexity of this algorithm is $O(n^4)$, but this includes the quantum communication of $O(n^4)$ qubits. To reduce the quantum communication complexity, our second algorithm incurs $O(n^6(\log n)^2)$ time complexity, but demands the quantum communication of only $O(n^2 \log n)$ qubits (plus classical communication of $O(n^6(\log n)^2)$ bits). While our first algorithm needs $\Theta(n^2)$ rounds of quantum communication, our second algorithm needs only one round of quantum communication at the beginning of the protocol to share sufficient amount of entanglement, and after the first round, the protocol performs only local quantum operations and classical communications (LOCCs) of $O(n \log n)$ rounds. Both algorithms are easily modified to support their use in asynchronous networks. Furthermore, both algorithms can be easily modified so that they work well even when each party initially knows only the upper bound of the number of parties. This implies that the exact number of parties can be computed when its upper bound is given. No classical zero-error algorithm exists in such cases for any topology that has a cycle as its subgraph [10].

2 Preliminaries

A *distributed system* (or *network*) is composed of multiple parties and bidirectional classical communication links connecting parties. In a quantum distributed system, every party can perform quantum computation and communication and each adjacent pair of parties has a bidirectional quantum communication link between them. When the parties and links are viewed as nodes and edges, respectively, the topology of the distributed system is expressed by an undirected con-

nected graph, say, $G = (V, E)$. In what follows, we may identify each party/link with its corresponding node/edge in the underlying graph for the system, if it is not confusing. Every party has *ports* corresponding one-to-one to communication links incident to the party. Every port of party l has a unique label i, $(1 \leq i \leq d_l)$, where d_l is the number of parties adjacent to l. More formally, G has a *port numbering*, which is a set σ of functions $\{\sigma[v] \mid v \in V\}$ such that, for each node v of degree d_v, $\sigma[v]$ is a bijection from the set of edges incident to v to $\{1, 2, \ldots, d_v\}$. It is stressed that each function $\sigma[v]$ may be defined independently of the others. Just for ease of explanation, we assume that port i corresponds to the link connected to the ith adjacent party of l. In our model, each party knows the number of its ports and the party can appropriately choose one of its ports whenever it transmits or receives a message.

Initially, every party has local information, such as its internal state, and global information, such as the number of nodes in the system (or its upper bound). Every party runs the same algorithm, which has local and global information as its arguments. If all parties have the same local and global information except for the number of ports the parties have, the system is said to be *anonymous*. This is essentially equivalent to the situation in which every party has the same identifier since we can regard the local/global information of the party as his identifier. If message passing is performed synchronously, such a distributed system is called *synchronous*. The unit interval of synchronization is called a *round* (see [11] for more detailed descriptions).

Next we define the *leader election (LE) problem*. Suppose that there is a distributed system and each party in the system has a variable initialized to 0. The task is to set the variable of exactly one of the parties to 1 and the variables of all the other parties to 0. In the case of anonymous networks, Yamashita and Kameda [17] proved that, if the "symmetricity" (defined in [17]) of the network topology is more than one, LE cannot be solved exactly (more rigorously speaking, there are some port numberings for which LE cannot be solved exactly) by any classical algorithm even if all parties know the topology of the network (and thus the number of nodes). In fact, for a broad class of graphs such as regular graphs, the "symmetricity" is more than one. When the parties initially know only the upper bound of the number of the parties, the result by Itai and Rodeh [10] implies that LE cannot be solved with zero error by any classical algorithm (including the one that may not always halt).

3 Quantum Leader Election Algorithm I

For simplicity, we assume that the network is synchronous and each party knows the number n of parties prior to the algorithm invocation. It is easy to generalize our algorithm to the asynchronous case and to the case where only the upper bound N of the number of parties is given, as will be discussed at the end of this section.

Initially all parties are eligible to become the unique leader. The key to solving the leader election problem in an anonymous network is to break symmetry, i.e., to have exactly one party possess a certain state coressponding to the leader.

First we introduce the concept of *consistent* and *inconsistent* strings. Suppose that each party l has a c-bit string x_l. That is, the n parties share cn-bit string $x = x_1 x_2 \cdots x_n$. For convenience, we may consider that each x_l expresses an integer, and identify string x_l with the integer it expresses. Given a set $E \subseteq \{1, \ldots, n\}$, string x is said to be *consistent* over E if x_l has the same value for all l in E. Otherwise x is said to be *inconsistent* over E. We also say that a cn-qubit pure state $|\psi\rangle = \sum_x \alpha_x |x\rangle$ shared by the n parties is *consistent (inconsistent)* over E if $\alpha_x \neq 0$ only for x that is consistent (inconsistent) over E. Further, for a positive integer m, we denote the state that is of the form of $(|0^m\rangle + |1^m\rangle)/\sqrt{2}$, by the m-cat state.

3.1 The Algorithm

The algorithm repeats one procedure exactly $(n-1)$ times, each of which is called a *phase*. In each phase, the number of eligible parties either decreases or remains the same, but never increases or becomes zero. After $(n-1)$ phases the number of eligible parties becomes one with certainty.

Each phase has a parameter denoted by k, whose value is $(n-i+1)$ in the ith phase. In each phase i, let $E_i \subseteq \{1, \ldots, n\}$ be the set of all ls such that party l is still eligible. First, each eligible party prepares the state $(|0\rangle + |1\rangle)/\sqrt{2}$, while each ineligible party prepares the state $|0\rangle$. Next every party calls Subroutine A, followed by partial measurement. This transforms the system state into either the cat state $(|0^{|E_i|}\rangle + |1^{|E_i|}\rangle)/\sqrt{2}$ shared only by eligible parties, or a state that is inconsistent over E_i. In the former case, each eligible party calls Subroutine B. If k equals $|E_i|$, Subroutine B always succeeds in transforming the $|E_i|$-cat state into a state that is inconsistent over E_i. Now, each eligible party l measures his qubits in the computational basis to obtain (a binary expression of) some integer z_l. Parties then compute the maximum value of z_l over all eligible parties l, by calling Subroutine C. Finally, parties with the maximum value remain eligible, while the other parties become ineligible. More precisely, each party l performs Algorithm I described below with parameters "eligible," n, and d_l. The party who obtains the output "eligible" is the unique leader.

Subroutine A: Subroutine A is essentially for the purpose of checking the consistency of each string that is superposed to a quantum state shared by parties. We use a commute operator "$\circ$" over a set $\mathcal{S} = \{0, 1, *, \times\}$ whose operations are summarized in Table 1. Intuitively, "0" and "1" represent the possible values all eligible parties will have when the string finally turns out to be consistent; "$*$" represents "don't care," which means that the corresponding party has no information about the values any of eligible parties have; and "$\times$" represents "inconsistent," which means that the corresponding party already knows that the string is inconsistent. Subroutine A is precisely described below.

As one can see from the description of Algorithm I, the content of $\mathbf{S}$ is initially "consistent" whenever Subroutine A is called. Therefore, after every party finishes Subroutine A, the state shared by parties in their $\mathbf{R}_0$s is decomposed into a consistent state for which each party has the content "consistent" in his

Algorithm I

Input: a classical variable **status**, integers n, d
Output: a classical variable **status**

1. Prepare one-qubit quantum registers $\mathbf{R}_0$, $\mathbf{R}_1$, and $\mathbf{S}$.
2. For $k := n$ down to 2, do the following:
   2.1 If **status** = "eligible," prepare the states $(|0\rangle + |1\rangle)/\sqrt{2}$ and $|$"consistent"$\rangle$ in $\mathbf{R}_0$ and $\mathbf{S}$, otherwise prepare the states $|0\rangle$ and $|$"consistent"$\rangle$ in $\mathbf{R}_0$ and $\mathbf{S}$.
   2.2 Perform Subroutine A with $\mathbf{R}_0$, $\mathbf{S}$, **status**, n, and d.
   2.3 Measure the qubit in $\mathbf{S}$ in the $\{|$"consistent"$\rangle, |$"inconsistent"$\rangle\}$ basis.
   If it results in $|$"consistent"$\rangle$ and **status** = "eligible," prepare the state $|0\rangle$ in $\mathbf{R}_1$ and perform Subroutine B with $\mathbf{R}_0$, $\mathbf{R}_1$, and k.
   2.4 If **status** = "eligible," measure the qubits in $\mathbf{R}_0$ and $\mathbf{R}_1$ in the $\{|0\rangle, |1\rangle\}$ basis to obtain a nonnegative integer z expressed by the two bits; otherwise let $z := -1$.
   2.5 Perform Subroutine C with z, n, and d to know the maximum value $z_{\max}$ of z over all parties.
   If $z \neq z_{\max}$, let **status** := "ineligible."
3. Output **status**.

Table 1. The definitions of commute operator "$\circ$"

x	y	$x \circ y$	x	y	$x \circ y$	x	y	$x \circ y$	x	y	$x \circ y$
0	0	0	1	0	$\times$	$*$	0	0	$\times$	0	$\times$
0	1	$\times$	1	1	1	$*$	1	1	$\times$	1	$\times$
0	$*$	0	1	$*$	1	$*$	$*$	$*$	$\times$	$*$	$\times$
0	$\times$	$\times$	1	$\times$	$\times$	$*$	$\times$	$\times$	$\times$	$\times$	$\times$

S, and an inconsistent state for which each party has the content "inconsistent" in his **S**. Steps 4 and 5 are performed so that the output quantum registers $\mathbf{R}_0$ and **S** are disentangled from work quantum registers $\mathbf{X}_i^{(t)}$s.

Subroutine B: Suppose k parties are still eligible and share the k-cat state $(|0^k\rangle + |1^k\rangle)/\sqrt{2}$. Subroutine B has purpose of changing the k-cat state to a superposition of inconsistent strings, if k is given. Subroutine B is precisely described as follows, where $\{U_k\}$ and $\{V_k\}$ are two families of unitary operators,

$$U_k = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{-i\frac{\pi}{k}} \\ -e^{i\frac{\pi}{k}} & 1 \end{pmatrix}, V_k = \frac{1}{\sqrt{R_k+1}} \begin{pmatrix} 1/\sqrt{2} & 0 & \sqrt{R_k} & e^{i\frac{\pi}{k}}/\sqrt{2} \\ 1/\sqrt{2} & 0 & -\sqrt{R_k}e^{-i\frac{\pi}{k}} & e^{-i\frac{\pi}{k}}/\sqrt{2} \\ \sqrt{R_k} & 0 & \frac{e^{-i\frac{\pi}{2k}} I_k}{i\sqrt{2}R_{2k}} & -\sqrt{R_k} \\ 0 & \sqrt{R_k+1} & 0 & 0 \end{pmatrix},$$

where R_k and I_k are the real and imaginary parts of $e^{i\frac{\pi}{k}}$, respectively.

The point is that the amplitudes of the states $|00\rangle^{\otimes k}$, $|01\rangle^{\otimes k}$, $|10\rangle^{\otimes k}$, and $|11\rangle^{\otimes k}$ shared by k eligible parties in their registers $\mathbf{R}_0$ and $\mathbf{R}_1$ are simultaneously

Subroutine A

Input: one-qubit quantum registers $\mathbf{R}_0, \mathbf{S}$, a classical variable **status**, integers n, d
Output: one-qubit quantum registers $\mathbf{R}_0, \mathbf{S}$

1. Prepare two-qubit quantum registers $\mathbf{X}_0^{(1)}, \ldots, \mathbf{X}_d^{(1)}, \ldots, \mathbf{X}_0^{(n-1)}, \ldots, \mathbf{X}_d^{(n-1)}, \mathbf{X}_0^{(n)}$. If **status** = "eligible," copy the content of $\mathbf{R}_0$ to $\mathbf{X}_0^{(1)}$, otherwise set the content of $\mathbf{X}_0^{(1)}$ to "$*$."
2. For $t := 1$ to $n-1$, do the following:
   2.1 Copy the content of $\mathbf{X}_0^{(t)}$ to each of $\mathbf{X}_1^{(t)}, \ldots, \mathbf{X}_d^{(t)}$.
   2.2 Exchange the qubit in $\mathbf{X}_i^{(t)}$ with the party connected via port i for $1 \leq i \leq d$ (i.e., the original qubit in $\mathbf{X}_i^{(t)}$ is sent via port i, and the qubit received via that port is newly set in $\mathbf{X}_i^{(t)}$).
   2.3 Set the content of $\mathbf{X}_0^{(t+1)}$ to $x_0^{(t)} \circ x_1^{(t)} \circ \cdots \circ x_d^{(t)}$, where $x_i^{(t)}$ denotes the content of $\mathbf{X}_i^{(t)}$ for $0 \leq i \leq d$.
3. If the content of $\mathbf{X}_0^{(n)}$ is "$\times$," turn the content of $\mathbf{S}$ over (i.e., if initially the content of $\mathbf{S}$ is "consistent," it is flipped to "inconsistent," and vice versa).
4. Invert every computation and communication in Step 2.
5. Invert every computation in Step 1.
6. Output quantum registers $\mathbf{R}_0$ and $\mathbf{S}$.

Subroutine B

Input: one-qubit quantum registers $\mathbf{R}_0, \mathbf{R}_1$, an integer k
Output: one-qubit quantum registers $\mathbf{R}_0, \mathbf{R}_1$

1. If k is even, apply U_k to the qubit in $\mathbf{R}_0$; otherwise perform CNOT controlled by the qubit in $\mathbf{R}_0$ to that in $\mathbf{R}_1$, and then apply V_k to the qubits in $\mathbf{R}_0$ and $\mathbf{R}_1$.
2. Output quantum registers $\mathbf{R}_0$ and $\mathbf{R}_1$.

zero after each eligible party applies Subroutine B with parameter k, if the particles in $\mathbf{R}_0$s of all eligible parties form the k-cat state.

Subroutine C: Subroutine C is a classical algorithm that computes the maximum value over parties. The procedure is very similar to Subroutine A. In fact, Subroutines A and C can be merged into one subroutine, although they are separately explained for simplicity. Subroutine C is precisely described as follows.

3.2 Complexity Analysis

First we state the complexity of Algorithm I without proof.

Theorem 1. *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph, respectively. Given the number n of parties, Algorithm I exactly elects a unique leader in $\Theta(n^2)$ rounds and $O(Dn^2)$ time. The total communication complexity over all parties is $\Theta(|E|n^2)$.*

If each party initially knows only the upper bound N of the number of parties, each party has only to perform Algorithm I with N instead of n. The complexity in this case is described simply by replacing every n by N in Theorem 1.

Subroutine C

Input: integers z, n, d
Output: an integer $z_{\max}$

1. Let $z_{\max} := z$.
2. For $t := 1$ to $n - 1$, do the following:
   2.1 Let $y_0 := z_{\max}$.
   2.2 Send y_0 via port i for $1 \leq i \leq d$.
   Set y_i to the value received via port i for $1 \leq i \leq d$.
   2.3 Let $z_{\max} := \max_{0 \leq i \leq d} y_i$.
3. Output $z_{\max}$.

Furthermore, Algorithm I is easily modified so that it works well even in the asynchronous settings. Note that all parties receive messages via each port at each round. Now, let each party wait to perform the operations of the $(i + 1)$st round until he finishes receiving all messages that are supposed to be received at the ith round. This modification enables us to simulate synchronous behavior in asynchronous networks. In order to know at which round the received message was originally sent, we tag every message. This modification increases the communication and time complexity by the multiplicative factor $\log n$.

4 Quantum Leader Election Algorithm II

To reduce the amount of quantum communication, our second algorithm make use of a classical technique, called *view*, which was introduced by Yamashita and Kameda [17]. However, a naïve application of view exponentially increases the classical time/communication complexity. To reduce this complexity, this paper introduces the new technique of *folded view*, with which the algorithm still runs in time/communication polynomial with respect to the number of parties.

4.1 View and Folded View

First, we briefly review the classical technique, *view*. Let $G = (V, E)$ be the underlying network topology and let $n = |V|$. Suppose each party corresponding to node $v \in V$ has a value $x_v \in S$ for some set S, and consider a mapping $X : V \to S$ defined by $X(v) = x_v$. For each v and port numbering σ, the *view* $T_{G,\sigma,X}(v)$ is a labeled, rooted tree with infinite depth defined recursively as follows: (1) $T_{G,\sigma,X}(v)$ has the root w with label $X(v)$, corresponding to v, (2) for each vertex v_i adjacent to v in G, $T_{G,\sigma,X}(v)$ has vertex w_i labeled with $X(v_i)$, and an edge from root w to w_i with label $l(v, v_i)$ given by $l(v, v_i) = (\sigma[v](v, v_i), \sigma[v_i](v, v_i))$, and (3) w_i is the root of $T_{G,\sigma,X}(v_i)$. It should be stressed that v, v_i, w, and w_i are not identifiers of parties and are introduced just for definition. For simplicity, we often use $T_X(v)$ instead of $T_{G,\sigma,X}(v)$, because we usually discuss views of some fixed network with some fixed port numbering. The *view of depth* h is the subtree of $T_X(v)$ of depth h with the same root as is that of $T_X(v)$, which is denoted by $T_X^h(v)$. For any value $x \in S$, the number of parties having x can be

computed from $T_X^{2(n-1)}(v)$ [17]. Each party v can construct $T_X^h(v)$ as follows. In the first round, each party v constructs $T_X^0(v)$, i.e., the root of $T_X(v)$. For each party v, if v has $T_X^{i-1}(v)$ in the ith round, v can construct $T_X^i(v)$ in the $(i+1)$st round by exchanging $T_X^{i-1}(v)$ with his neighbors. By induction, in the $(h+1)$st round, each party v can construct $T_X^h(v)$.

Note that the size of $T_X^h(v)$ is exponential in h, which results in exponential time/communication complexity when we construct it. To reduce the time/communication complexity to something bounded by a polynomial, we introduce the new technique called *folded view* by generalizing the OBDD-reduction algorithm [6]. A *folded view (f-view) of depth* h, denoted by $\widetilde{T}_X^h(v)$, is a vertex- and edge-labeled directed acyclic multigraph obtained by merging nodes at the same level in $T_X^h(v)$ into one node if the subtrees rooted at them are isomorphic. The number of nodes in each level of an f-view is obviously bounded by n, and thus the total number of nodes in an f-view of depth h is at most hn. Actually, an f-view of depth $(h+1)$ can be efficiently constructed from a given f-view of depth h without unfolding it. Here we state without proof that every f-view of depth h is constructed in $O(h^2n^3(\log n)^2)$ time for each party and with $O(|E|h^2n^2\log n)$ bits of classical communication. Once $\widetilde{T}_X^{2(n-1)}(v)$ is constructed, each party can count without communication the number of parties having a value x in $O(n^6\log n)$ time.

4.2 The Algorithm

As in the previous section, we assume that the network is synchronous and each party knows the number n of parties prior to the algorithm. Again our algorithm is easily generalized to the asynchronous case. Although it needs a bit more elaboration, which is not mentioned in this version, it is also possible to modify our algorithm to work well even if only the upper bound N of the number of parties is given.

The algorithm consists of two stages, which we call Stages 1 and 2 hereafter. Stage 1 aims to have the n parties share a certain type of entanglement, and thus, this stage requires the parties to exchange quantum messages. In Stage 1, each party performs Subroutine Q $s = \lceil \log n \rceil$ times in parallel to share s pure quantum states $|\phi^{(1)}\rangle, \ldots, |\phi^{(s)}\rangle$ of n qubits. Here, each $|\phi^{(i)}\rangle$ is of the form $(|x^{(i)}\rangle + |\overline{x^{(i)}}\rangle)/\sqrt{2}$ for an n-bit string $x^{(i)}$ and its bitwise negation $\overline{x^{(i)}}$, and the lth qubit of each $|\phi^{(i)}\rangle$ is possessed by the lth party. It is stressed that only one round of quantum communication is necessary in Stage 1.

In Stage 2, the algorithm decides a unique leader among the n parties only by local quantum operations and classical communications with the help of the shared entanglement prepared in Stage 1. This stage consists of at most s phases, each of which reduces the number of eligible parties by at least half. In each phase i, let $E_i \subseteq \{1, \ldots, n\}$ be the set of all ls such that party l is still eligible. First every party runs Subroutine $\tilde{\text{A}}$ to decide if state $|\phi^{(i)}\rangle$ is consistent or inconsistent over E_i. If state $|\phi^{(i)}\rangle$ is consistent, every party performs Subroutine $\tilde{\text{B}}$, which first transforms $|\phi^{(i)}\rangle$ into the $|E_i|$-cat state $(|0^{|E_i|}\rangle + |1^{|E_i|}\rangle)/\sqrt{2}$ shared only by eligible parties and then calls Subroutine B described in the previous section to

Algorithm II

Input: a classical variable **status**, integers n, d
Output: a classical variable **status**

Stage 1:

Let $s := \lceil \log n \rceil$ and prepare one-qubit quantum registers $\mathbf{R}_0^{(1)}, \ldots, \mathbf{R}_0^{(s)}$ and $\mathbf{R}_1^{(1)}, \ldots, \mathbf{R}_1^{(s)}$, each of which is initialized to the $|0\rangle$ state.

Perform s attempts of Subroutine Q in parallel, each with $\mathbf{R}_0^{(i)}$ and d for $1 \leq i \leq s$, to obtain each $y^{(i)}$ and to share each $|\phi^{(i)}\rangle = (|x^{(i)}\rangle + |\overline{x^{(i)}}\rangle)/\sqrt{2}$ of n qubits.

Stage 2:

Let $k := n$.

For $i := 1$ to s, repeat the following:

1. Perform Subroutine $\tilde{\mathrm{A}}$ with **status**, n, d, and $y^{(i)}$ to obtain its output **consistency**.
2. If **consistency** = "consistent," perform Subroutine $\tilde{\mathrm{B}}$ with $\mathbf{R}_0^{(i)}$, $\mathbf{R}_1^{(i)}$, **status**, k, n, and d.
3. If **status** = "eligible," measure the qubits in $\mathbf{R}_0^{(i)}$ and $\mathbf{R}_1^{(i)}$ in the $\{|0\rangle, |1\rangle\}$ basis to obtain a nonnegative integer z; otherwise set $z := -1$.
   Perform Subroutine $\tilde{\mathrm{C}}$ with **status**, z, n, and d to compute nonnegative integers z_{minor} and $c_{z_{\mathrm{minor}}}$.
4. If $z \neq z_{\mathrm{minor}}$, let **status** := "ineligible."
   Let $k := c_{z_{\mathrm{minor}}}$.
5. If $k = 1$, terminate and output **status**.

obtain an inconsistent state. Now each party l measures his qubits to obtain a label x_l and performs Subroutine $\tilde{\mathrm{C}}$ that reduces the number of eligible parties by at least half via minority voting.

More precisely, each party l performs Algorithm II described below with parameters "eligible," n, and d_l. The party who obtains output "eligible" is the unique leader.

Subroutine Q: Subroutine Q is mainly for the purpose of sharing a cat-like quantum state $|\phi\rangle = (|x\rangle + |\overline{x}\rangle)/\sqrt{2}$. It also outputs a classical string, which is used in Stage 2 for each party to obtain the information on $|\phi\rangle$ via just classical communication. This subroutine can be performed in parallel by tagging messages, and thus Stage 1 involves only one round of quantum communication. The precise description of Subroutine Q is found below. Step 6 is necessary to disentangle the qubit in output register $\mathbf{R}_0$ from that in every $\mathbf{R}'_i$.

Subroutine $\tilde{\mathrm{A}}$: Suppose $|\phi\rangle = (|x\rangle + |\overline{x}\rangle)/\sqrt{2}$ is shared by the n parties. Let x_l be the lth bit of x and let X and $\overline{X}$ be mappings defined by $X(v_l) = x_l$ and $\overline{X}(v_l) = \overline{x_l}$ for each l, respectively, where v_l is the node corresponding to the lth party. Similar to Subroutine A in the previous section, Subroutine $\tilde{\mathrm{A}}$ checks the consistency of $|\phi\rangle$. Hereafter v denotes the node corresponding to the party invoking the subroutine. The output y of Subroutine Q is useful to construct f-view $\tilde{T}_X^{n-1}(v)$ or $\tilde{T}_{\overline{X}}^{n-1}(v)$, which can replace quantum communications in the consistency check. The precise description of Subroutine $\tilde{\mathrm{A}}$ is found below.

Subroutine Q

Input: a one-qubit quantum register $\mathbf{R}_0$, an integer d
Output: a one-qubit quantum register $\mathbf{R}_0$, a string y of length d

1. Prepare $2d$ one-qubit quantum registers $\mathbf{R}'_1, \ldots, \mathbf{R}'_d$ and $\mathbf{S}_1, \ldots, \mathbf{S}_d$, each of which is initialized to the $|0\rangle$ state.
2. Create the $(d+1)$-cat state $(|0^{d+1}\rangle + |1^{d+1}\rangle)/\sqrt{2}$ in registers $\mathbf{R}_0$, $\mathbf{R}'_1, \ldots, \mathbf{R}'_d$.
3. Exchange the qubit in $\mathbf{R}'_i$ with the party connected via port i for $1 \leq i \leq d$ (i.e., the original qubit in $\mathbf{R}'_i$ is sent via port i, and the qubit received via that port is newly set in $\mathbf{R}'_i$).
4. Set the content of $\mathbf{S}_i$ to $x_0 \oplus x_i$, for $1 \leq i \leq d$, where x_0 and x_i denote the contents of $\mathbf{R}_0$ and $\mathbf{R}'_i$, respectively.
5. Measure the qubit in $\mathbf{S}_i$ in the $\{|0\rangle, |1\rangle\}$ basis to obtain a bit y_i, for $1 \leq i \leq d$. Set $y := y_1 \cdots y_d$.
6. Clear the content of $\mathbf{R}'_i$ by using y_i, for $1 \leq i \leq d$.
7. Output $\mathbf{R}_0$ and y.

Subroutine $\tilde{\mathbf{A}}$

Input: a classical variable **status**, integers n, d, a string y of length d
Output: a classical variable **consistency**

1. Set $\widetilde{T}_Y^0(v)$ to the node labeled with $(0, \mathbf{status})$, where Y is either X or $\overline{X}$.
2. For $i := 1$ to $(n-1)$, do the following:
   2.1 Send $\widetilde{T}_Y^{i-1}(v)$ and receive $\widetilde{T}_Y^{i-1}(v_j)$ via port j, for $1 \leq j \leq d$, where v_j is the node corresponding to the party connected via port j.
   2.2 If the jth bit y_j of y is 1, negate the first element of every node label in $\widetilde{T}_Y^{i-1}(v_j)$, for $1 \leq j \leq d$.
   2.3 Set the root of $\widetilde{T}_Y^i(v)$ to the node labeled with $(0, \mathbf{status})$.
   Set the jth child of the root of $\widetilde{T}_Y^i(v)$ to $\widetilde{T}_Y^{i-1}(v_j)$, for $1 \leq j \leq d$.
   For every level of $\widetilde{T}_Y^i(v)$, merge nodes at that level into one node if the f-views rooted at them are isomorphic.
3. If both label $(0, \text{“eligible”})$ and label $(1, \text{“eligible”})$ are found among the node labels in $\widetilde{T}_Y^{n-1}(v)$, let **consistency** := “inconsistent”; otherwise let **consistency** := “consistent.”
4. Output **consistency**.

Subroutine $\tilde{\mathbf{B}}$: Suppose $|\phi\rangle = (|x\rangle + |\overline{x}\rangle)/\sqrt{2}$ shared by the n parties is consistent over the set E of eligible parties. After every ineligible party performs Step 2 in Subroutine $\tilde{\mathrm{B}}$, the state shared by the eligible parties is either $\pm(|0^{|E|}\rangle + |1^{|E|}\rangle)/\sqrt{2}$ or $\pm(|0^{|E|}\rangle - |1^{|E|}\rangle)/\sqrt{2}$. The state $\pm(|0^{|E|}\rangle - |1^{|E|}\rangle)/\sqrt{2}$ is shared if and only if the number of ineligible parties that measured $|-\rangle$ in Step 1 is odd, where $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$, respectively. Steps 3 and 4 are for the purpose of having the eligible parties always share $\pm(|0^{|E|}\rangle + |1^{|E|}\rangle)/\sqrt{2}$. Again let v denote the node corresponding to the party that invoked the subroutine, and define the family $\{W_k\}$ of unitary transformations by $W_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{k}} \end{pmatrix}$. The precise description of Subroutine $\tilde{\mathrm{B}}$ is found below.

Subroutine $\tilde{\mathrm{B}}$

Input: one-qubit quantum registers $\mathbf{R}_0, \mathbf{R}_1$, a classical variable **status**, integers k, n, d
Output: one-qubit quantum registers $\mathbf{R}_0, \mathbf{R}_1$

1. Let $w := 0$.
2. If **status** = "ineligible," measure the qubit in $\mathbf{R}_0$ in the $\{|+\rangle, |-\rangle\}$ basis. If this results in $|-\rangle$, let $w := 1$.
3. Construct f-view $\widetilde{T}_W^{(2n-1)}(v)$ to count the number p of parties with $w = 1$, where W is the underlying mapping naturally induced by the w values of all parties.
4. If p is odd and **status** = "eligible," apply W_k to the qubit in $\mathbf{R}_0$.
5. Perform Subroutine B with $\mathbf{R}_0$, $\mathbf{R}_1$ and k.
6. Output quantum registers $\mathbf{R}_0$ and $\mathbf{R}_1$.

Subroutine $\tilde{\mathrm{C}}$

Input: integers z, n, d
Output: integers z_{minor}, $c_{z_{\mathrm{minor}}}$

1. Construct f-view $\widetilde{T}_Z^{(2n-1)}(v)$, where Z is the underlying mapping naturally induced by the z values of all parties.
2. For $i := 0$ to 3, count the number c_i of parties having a value $z = i$ using $\widetilde{T}_Z^{(2n-1)}(v)$. If $c_i = 0$, let $c_i := n$.
3. Let $z_{\mathrm{minor}} \in \{m \mid c_m = \min_{0 \le i \le 3} c_i\}$.
4. Output z_{minor} and $c_{z_{\mathrm{minor}}}$.

Subroutine $\tilde{\mathrm{C}}$: Suppose each party l has value z_l. Subroutine C is a classical algorithm that computes value z_{minor} such that the number of parties with value z_{minor} is nonzero and the smallest among all possible z values. The precise description of Subroutine $\tilde{\mathrm{C}}$ is found below.

4.3 Complexity Analysis

Here we only give the complexity of Algorithm II without proof.

Theorem 2. *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph, respectively. Given the number n of parties [the upper bound N of it], Algorithm II exactly elects a unique leader in $O(n \log n)$ [$O(N \log N)$] rounds and $O(n^6 (\log n)^2)$ [$O(N^7 (\log N)^2)$] time of which only the first round requires quantum communication. The total communication complexity over all parties is $O(|E| n^4 (\log n)^2)$ [$O(|E| N^5 (\log N)^2)$] which includes only $O(|E| \log n)$ [$O(|E| N \log N)$] qubits of quantum communication.*

References

1. A. Ambainis, H. M. Buhrman, Y. Dodis, and H. Röhrig. Multiparty quantum coin flipping. In *Proc. of 19th IEEE Conf. on Computational Complexity*, pages 250–259, 2004.

2. D. Angluin. Local and global properties in networks of processors (extended abstract). In *Proc. of 20th ACM STOC*, pages 82–93, 1980.
3. Z. Bar-Yossef, T. S. Jayram, and I. Kerenidis. Exponential separation of quantum and classical one-way communication complexity. In *Proc. of 36th ACM STOC*, pages 128–137, 2004.
4. C. H. Bennett. Quantum cryptography using any two nonorthogonal states. *Phys. Rev. Lett.*, 68(21):3121–3124, 1992.
5. C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proc. of IEEE Conf. on Computers, Systems and Signal Processing*, pages 175–179, 1984.
6. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.
7. H. M. Buhrman, R. E. Cleve, J. H. Watrous, and R. de Wolf. Quantum fingerprinting. *Phys. Rev. Lett.*, 87(16):167902, 2001.
8. C. Crépeau, D. Gottesman, and A. D. Smith. Secure multi-party quantum computation. In *Proc. of 34th ACM STOC*, pages 643–652.
9. P. Dumais, D. Mayers, and L. Salvail. Perfectly concealing quantum bit commitment from any quantum one-way permutation. In *Proc. of EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 300–315, 2000.
10. A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87, 1990.
11. N. A. Lynch. *Distributed Algorithms.* Morgan Kaufman Publishers, 1996.
12. D. Mayers. Unconditional security in quantum cryptography. *J. ACM*, 48(3):351–406, 2001.
13. R. Raz. Exponential separation of quantum and classical communication complexity. In *Proc. of 31st ACM STOC*, pages 358–367, 1999.
14. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.
15. P. W. Shor and J. Preskill. Simple proof of security of the BB84 quantum key distribution protocol. *Phys. Rev. Lett.*, 85(2):441–444, 2000.
16. K. Tamaki, M. Koashi, and N. Imoto. Unconditionally secure key distribution based on two nonorthogonal states. *Phys. Rev. Lett.*, 90(16):167904, 2003.
17. M. Yamashita and T. Kameda. Computing on anonymous networks: Part I – characterizing the solvable cases. *IEEE Trans. Parallel Distrib. Syst.*, 7(1):69–89, 1996.
18. M. Yamashita and T. Kameda. Computing on anonymous networks: Part II – decision and membership problems. *IEEE Trans. Parallel Distrib. Syst.*, 7(1):90–96, 1996.

Robust Polynomials and Quantum Algorithms

Harry Buhrman[1,2,⋆], Ilan Newman[3,⋆⋆], Hein Röhrig[4], and Ronald de Wolf[1]

[1] CWI, Amsterdam, the Netherlands
[2] ILLC, University of Amsterdam, the Netherlands
[3] Dept. of Computer Science, Haifa University, Israel
[4] Dept. of Computer Science, University of Calgary, Canada

Abstract. We define and study the complexity of *robust* polynomials for Boolean functions and the related fault-tolerant quantum decision trees, where input bits are perturbed by noise. We show that, in contrast to the classical model of Feige et al., every Boolean function can be computed by $O(n)$ quantum queries even in the model with noise. This implies, for instance, the somewhat surprising result that every Boolean function has robust degree bounded by $O(n)$.

1 Introduction

In the last two decades, polynomials of many varieties have been used to good effect in complexity theory. We study a variety here that is tailored to analyzing algorithms with *noisy input*.

Robust Polynomials. A *robust* polynomial for a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a real multivariate polynomial $p(z_1, \ldots, z_n)$ such that for every $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ and every $z = (z_1, \ldots, z_n) \in \mathbb{R}^n$, if $\forall i : |x_i - z_i| \leq 1/3$ then $|f(x) - p(z)| \leq 1/3$ (the $1/3$ in both cases can be changed to any other constant). The *robust degree* of f is the smallest degree of a robust polynomial for f; note that we do not require robust polynomials to be multilinear.

The motivation behind the definition of robust polynomials is twofold: First it can be viewed as a strengthening (restriction) of the notion of approximating polynomials. An approximating polynomial for f is a multivariate real polynomial q that approximates f within an additive term of $1/3$ for each Boolean input. Approximating polynomials for Boolean functions are of interest in themselves and have been the object of study for a while. Their minimal degree is tightly related to the decision tree complexity of f [9, 2]. Indeed, this "polynomial method" [2] is one of the main tools for obtaining lower bounds on the number of queries in quantum algorithms. One difficulty, however, is that approximating polynomials do not directly compose; if $f(x_1, \ldots, x_n)$ is a Boolean function

⋆ H.B., H.R., and R.d.W. supported in part by the EU fifth framework projects QAIP, IST-1999-11234, and RESQ, IST-2001-37559, and an NWO grant.
⋆⋆ partially supported by ISF grant 55/03.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 593–604, 2005.

with an approximating polynomial p_f and $g(y_1, \ldots, y_m)$ is a Boolean function with an approximating polynomial p_g, then the polynomial on $n \cdot m$ variables $p_f(p_g, \ldots, p_g)$, which is obtained by plugging in a copy of p_g for each appearance of y_i, is not necessarily an approximating polynomial for the composed function $f(g, \ldots, g)$ on $n \cdot m$ variables. This difficulty is avoided with robust polynomials; if p_f, p_g are robust for f, g respectively, then their composition is a robust polynomial (and thus also approximating) for the composed function.

Another motivation is the study of quantum decision trees that can tolerate noise in their inputs. We show that a natural quantum analogue of classical fault-tolerant decision trees can be defined. As a result, it will follow that every such algorithm (and in fact every classical noisy decision tree algorithm as well) implies the existence of a robust degree-$2q$ polynomial for the function, where q is the number of queries. This relates the robust degree to fault-tolerant computation in exactly the same way that approximating polynomials are related to bounded-error quantum algorithms. Surprisingly, our results imply robust quantum algorithms with a linear number of queries, as well as robust polynomials of linear degree, for *any* Boolean function. This should be contrasted with the result of Feige et al. [3] who proved that for most Boolean functions an overhead factor of $\Omega(\log n)$ on the number of queries is needed in the noisy case compared to the non-noisy case. In particular, consider the parity function on n variables. This function can be decided trivially by an n-query decision tree, and hence can be represented exactly by an n-degree real multilinear polynomial (which is just the single monomial containing all variables in the $\{-1, 1\}$ representation). Feige et al. [3] prove that in the noisy decision tree any algorithm for PARITY needs $\Omega(n \log n)$ queries. Using standard amplification techniques, this yields a $O(n \log n)$-degree robust polynomial for PARITY. Can one do better? Our results imply that there is a robust polynomial for PARITY of degree $O(n)$. However, we only have an indirect description of this polynomial by means of a quantum algorithm, and do not know how to construct such a polynomial directly.

Noisy Quantum Queries. We now discuss in more detail the model of noisy-decision trees in the quantum world. The notion of a "noisy query" in the quantum case is not obvious and natural as in the classical case since one application of a quantum query can address many different x_i's in superposition. A first proposal would be that for each quantum query, each of the bits is flipped independently with probability ϵ. Each such quantum query introduces a lot of randomness and the algorithm's state after the query is a mixed quantum state rather than a pure state. In fact, this model is a concrete (and very destructive) form of decoherence; the effects of various forms of decoherence on oracle algorithms like Grover's have been studied before, see e.g., [8, 10].

A second model, which we will adopt here, is to assume that we have n quantum procedures, $A_1, \ldots, A_n$, such that A_i outputs x_i with probability at least $1 - \epsilon$. Such a *coherent-noise model* is not unreasonable. For instance, it could be the case that the input bits are actually computed for us by subroutines. Such algorithms can always be made coherent by pushing measurements to the end, which means that we can apply and reverse them at will. To enable us

to apply the A_i's in superposition, we assume we have a black box that maps $\mathcal{A} : |i\rangle|0\rangle \mapsto |i\rangle A_i|0\rangle$. One application of this will count as one query.

A third model, which we will call the *multiple-noisy-copies model*, was studied by Szegedy and Chen [11]. Here, instead of x_i, the algorithm can only query "perturbed" copies $y_{i,1}, \ldots, y_{i,m}$ of x_i. The $y_{i,j}$ are independent Boolean random variables with $\Pr[x_i = y_{i,j}] \geq 1 - \epsilon$ for each $i = 1, \ldots, n$ and $j = 1, \ldots, m$. In contrast to the first proposal, this model leaves the queries perfectly reversible, since the perturbed copies are fixed at the start of the algorithm and the same $y_{i,j}$ can be queried more than once. The assumption of this model is also stronger than the second model, since we can construct a 1-query A_i that just outputs a superposition of all $y_{i,j}$. If m is sufficiently large, this A_i will compute x_i with high success probability, satisfying the assumption of the second model (see Section 4.2 for details).

Robust Quantum Algorithms. Assuming the second model and some fixed ϵ, we call a quantum algorithm *robust* if it computes f with bounded error probability on inputs of n bits given by bounded-error algorithms $A_1, \ldots, A_n$, respectively. A first observation is that every T-query non-robust algorithm can be made robust at a multiplicative cost of $O(\log T)$. With $O(\log T)$ queries, a majority gate, and an uncomputation step, we can construct a unitary $\tilde{U}_x$ that approximates an exact quantum query $U_x : |i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle$ very well: $\|U_x - \tilde{U}_x\| \leq 1/100T$. Since errors add linearly in a quantum algorithm, replacing U_x by $\tilde{U}_x$ in a non-robust algorithm gives a robust algorithm with almost the same final state. In some cases better constructions are possible. For instance, a recent result by Høyer et al. [5] implies a quantum algorithm that robustly computes OR with $O(\sqrt{n})$ queries. This is only a constant factor worse than the noiseless case, which is Grover's algorithm [4]. In fact, we do not know of any function where the robust quantum query complexity is more than a constant factor larger than the non-robust complexity.

Our main result about quantum computing (made precise in Theorem 2) is the following:

> There exists a quantum algorithm that outputs $x_1, \ldots, x_n$, with high probability, using $O(n)$ invocations of the A_i algorithms (i.e., queries).

As already mentioned, this result implies that *every* n-bit function f can be robustly quantum computed with $O(n)$ queries. This contrasts with the classical $\Omega(n \log n)$ lower bound for PARITY. It is quite interesting to note that quantum computers, which usually are more fragile than classical computers, are actually more robust in the case of computing PARITY with noisy inputs. The result for PARITY can be extended to every symmetric function f: for every such function, the optimal quantum algorithm can be made robust with only a constant factor overhead (see Section 4.1).

Our result has a direct bearing on the *direct-sum problem*, which is the question how the complexity of computing n independent instances of a function scales with the complexity of one instance. One would expect that computing n instances with bounded-error takes no more than n times the complexity of one

instance. However, since we want all n instances to be computed correctly *simultaneously* with high probability, the only known general method in the classical world is to compute each instance with error probability reduced to $O(1/n)$. This costs another factor of $O(\log n)$. In fact, it follows from the $\Omega(n \log n)$ bound for PARITY that this factor of $\log n$ is optimal if we can only run algorithms for individual instances in a black-box fashion. In contrast, our result implies that in the quantum world, the bounded-error complexity of n instances is at most $O(n)$ times the bounded-error complexity of one instance. This is a very general result. For example, it also applies to communication complexity [7–Section 4.1.1]. If Alice and Bob have a bounded-error protocol for a distributed function f, using c bits (or qubits) of communication, then there is a bounded-error quantum protocol for n instances of f, using $O(n(c + \log n))$ qubits of communication. The additive $\log n$ is because Alice and Bob need to communicate (possibly in superposition) the index of the instance that they are computing. In contrast, the best known general classical solution uses $\Theta(cn \log n)$ bits of communication.

Note About Related Work. In a recent manuscript, Iwama et al. [6] study a similar but slightly weaker setting. There, the error probability for each input variable is *exactly* ϵ. If ϵ is known, then one can use a version of exact amplitude amplification to "rotate off" the error using $O(1)$ queries and hence make the algorithm robust. If ϵ unknown, it can be estimated very well using quantum amplitude estimation, after which amplitude amplification can be used as if ϵ was known. Iwama et al. derive from this that any quantum algorithm can be made robust (in their model) with only a constant factor overhead. Their model has the disadvantage that it does not cover the subroutine-scenario, where each input bit x_i is computed for us by an algorithm or subroutine A_i whose error we can only upper bound. Our model does not need the assumption that the error is the same for all input bits, and hence does not have this disadvantage.

2 Robust Polynomials — Preliminaries

In this section we study robust polynomials of two different but essentially equivalent types. The first type arises from the multiple-noisy-copies model, the second type is what we discussed in the introduction. For brevity, we omit the proofs of the lemmas presented in this section.

Definition 1. *An* (ϵ, m) perturbation *of* $x \in \{0,1\}^n$ *is a matrix* y *of* $n \times m$ *independent binary random variables* $y_{i,j}$ *so that* $\Pr[y_{i,j} = x_i] \geq 1 - \epsilon$ *for each* $1 \leq j \leq m$.

Definition 2. *A* type-1 (ϵ, m)-robust polynomial *for the Boolean function* $f : \{0,1\}^n \to \{0,1\}$ *is a real polynomial* p *in* nm *variables* $y_{i,j}$ *(with* $1 \leq i \leq n$ *and* $1 \leq j \leq m$*) so that for every* $x \in \{0,1\}^n$ *and* y *an* (ϵ, m) *perturbation of* x, $\Pr[|p(y) - f(x)| \geq 1/3] \leq 1/3$. *Moreover, for every* $v \in \{0,1\}^{nm}$, *we require* $-1/3 \leq p(v) \leq 4/3$.

Note that since $y_{i,j}^2 = y_{i,j}$ for a bit $y_{i,j}$, we can restrict attention to *multilinear* polynomials here.

The approximation "quality" of a type-1 robust polynomial can be boosted at constant multiplicative cost in the degree. Analogously we can improve the parameters to other constants:

Lemma 1. *If there is a* type-1 (ϵ, m)-robust polynomial *of degree d for f, then for some $m' = \mathrm{O}(m)$ there exists a type-1 (ϵ, m')-robust polynomial p of degree $\mathrm{O}(d)$ so that $x \in \{0,1\}^n$ and y an (ϵ, m') perturbation of x, $\Pr[|p(y) - f(x)| \geq 1/9] \leq 1/9$. Moreover, for any $v \in \{0,1\}^{nm'}$, $-1/9 \leq p(v) \leq 10/9$.*

The second kind of robust polynomial is the following:

Definition 3. *For a Boolean function $f : \{0,1\}^n \to \{0,1\}$, we call q a* type-2 ϵ-robust polynomial for f *if q is a real polynomial in n variables so that for every $x \in \{0,1\}^n$ and every $z \in [0,1]^n$ we have $|q(z) - f(x)| \leq 1/3$ if $|z_i - x_i| \leq \epsilon$ for all $i \in [n]$. If $\epsilon = 0$, then q is called an* approximating polynomial *for f.*

A minimal-degree type-2 robust polynomial for f need not be multilinear, in contrast to the type-1 variety. Note that we restrict the z_i's to lie in the set $[0,\epsilon] \cup [1-\epsilon, 1]$ rather than the less restrictive $[-\epsilon, \epsilon] \cup [1-\epsilon, 1+\epsilon]$. This facilitates later proofs, because it enables us to interpret the z_i's as probabilities. However, with some extra work we could also use the less restrictive definition here.

Definition 4. *For $f : \{0,1\}^n \to \{0,1\}$, let $\mathrm{rdeg}_1(f)$ denote the minimum degree of any type-1 $(1/3, 5\log n)$-robust polynomial for f, $\mathrm{rdeg}_2(f)$ be the minimum degree of any type-2 $1/3$-robust polynomial approximating f, and $\widetilde{\deg}(f)$ be the minimum degree among all approximating polynomials for f.*

We characterize the relation of type-1 and type-2 robust polynomials as follows:

Theorem 1. *For every type-2 ϵ-robust polynomial of degree d for f there is a type-1 $(\epsilon/2, \mathrm{O}(\log(n)/(1/2-\epsilon)^2))$-robust polynomial of degree d for f. Conversely, for every type-1 (ϵ, m)-robust polynomial of degree d for f there is a type-2 ϵ-robust polynomial of degree $\mathrm{O}(d)$ for f.*

Proof. Let p be a type-2 ϵ-robust polynomial of degree d for f. We choose $m = \mathrm{O}(\log(n)/(1/2-\epsilon)^2)$. If each $y_{i,j}$ is wrong with probability $\leq \epsilon/2$, then with probability at least $2/3$, the averages $\overline{y}_i$ will satisfy $|\overline{y}_i - x_i| \leq \epsilon$ for all $i \in [n]$. Hence the polynomial $p(\overline{y}_1, \ldots, \overline{y}_n)$ will be a type-1 $(\epsilon/2, \mathrm{O}(\log(n)/(1/2-\epsilon)^2))$-robust polynomial of degree d for f.

For the other direction, consider a type-1 (ϵ, m)-robust polynomial of degree d for f. Using Lemma 1, we boost the approximation parameters to obtain a type-1 (ϵ, m')-robust polynomial p of degree $\mathrm{O}(d)$, with $m' = \mathrm{O}(m)$, so that for any $x \in \{0,1\}^n$ and (ϵ, m') perturbation y of x, $\Pr[|p(y) - f(x)| \geq 1/9] \leq 1/9$. For $z \in \mathbb{R}^n$ with $0 \leq z_i \leq 1$ for all i, let $y_{i,j}$ ($i \in [n]$, $j \in [m']$) be independent random variables, where $y_{i,j} = 1$ with probability z_i. Define $q(z) := \mathrm{E}[p(y)]$. This q is a polynomial in z, because $\mathrm{E}[p(y)] = p(\mathrm{E}[y])$ and $\mathrm{E}[y_{i,j}] = z_i$. Moreover, if

for z there exists $x \in \{0,1\}^n$ with $|z_i - x_i| \leq \epsilon$ for all i, then y is an (ϵ, m') perturbation of x. Therefore $V := \{v : |p(v) - f(x)| \leq 1/9\}$ has probability $\Pr[y \in V] \geq 8/9$ and

$$|f(x) - q(z)| \leq \left| \sum_{v \in V} \Pr[y = v] \, (f(x) - p(v)) \right| + \left| \sum_{v \notin V} \Pr[y = v] \left(1 + \frac{1}{9}\right) \right| < \frac{1}{3} \ .$$

This means that $q(z)$ is a type-2 ϵ-robust polynomial for f of degree $\mathrm{O}(d)$. □

Note that in Definition 2 we require for type-1 polynomials p that for any Boolean assignment $v \in \{0,1\}^{nm}$ to the (possibly real) variables, the polynomial value $p(v)$ lies between $-1/3$ and $4/3$. Because of this totality requirement, the following corollary is given for total Boolean f only.

Corollary 1. $\mathrm{rdeg}_1(f) = \Theta(\mathrm{rdeg}_2(f))$ *for every (total) Boolean function* $f : \{0,1\}^n \to \{0,1\}$.

Robust quantum algorithms provide one way to construct robust polynomials:

Lemma 2. *Let* $f : \{0,1\}^n \to \{0,1\}$ *be a Boolean function. Let* Q *be a quantum algorithm that makes at most* q *queries on inputs from* $\{0,1\}^{n \times m}$. *If for every* $x \in \{0,1\}^n$ *and* y *an* (ϵ, m) *perturbation of* x, *we have that* $\Pr[Q(y) = f(x)] \geq 29/30$, *then there exists a degree-*$2q$ *type-1* (ϵ, m)*-robust polynomial for* f.

3 Quantum Robust Input Recovery

In this section we prove our main result, that we can recover an n-bit string x using $\mathrm{O}(n)$ invocations of algorithms $A_1, \ldots, A_n$ where A_i computes x_i with bounded error.

Theorem 2. *Given* ϵ*-error algorithms* $A_1, \ldots, A_n$ *for the bits* $x_1, \ldots, x_n$, *there is a quantum algorithm that recovers* $x = x_1 \ldots x_n$ *with probability* $2/3$ *using* $\mathrm{O}(n/(1/2 - \epsilon)^2)$ *queries (invocations of the* A_i*).*

We assume that $\epsilon > 0$ and that A_i is a unitary transformation

$$A_i : |0^t\rangle \mapsto \alpha_i |0\rangle |\psi_i^0\rangle + \sqrt{1 - \alpha_i^2} |1\rangle |\psi_i^1\rangle$$

for some $\alpha_i \geq 0$ such that $|\alpha_i|^2 \leq \epsilon$ if $x_i = 1$ and $|\alpha_i|^2 \geq 1 - \epsilon$ if $x_i = 0$; $|\psi_i^0\rangle$ and $|\psi_i^1\rangle$ are arbitrary $(t-1)$-qubit norm-1 quantum states. It is standard that any quantum algorithm can be expressed in this form by postponing measurements (i.e., unitarily write the measurement in an auxiliary register without collapsing the state); any classical randomized algorithm can be converted into this form by making it reversible and replacing random bits by states $(|0\rangle + |1\rangle)/\sqrt{2}$. By applying a NOT to the first qubit after the execution of A_i, we can easily implement

$$\bar{A}_i : |0^t\rangle \mapsto \alpha_i |1\rangle |\psi_i^0\rangle + \sqrt{1 - \alpha_i^2} |0\rangle |\psi_i^1\rangle \ ,$$

Procedure RobustFind(n, $\mathcal{A}$, ϵ, β, γ, δ)

$n \in \mathbb{N}$, $\mathcal{A} : n$ quantum algorithms, $\epsilon, \beta, \gamma, \delta > 0$
Output: $i \in [n] \cup \{\perp\}$ with the following properties:

1. if $\mathcal{A}$ is ϵ-close to $x \in \{0,1\}^n$ and $|x| \geq \beta n$, then $i \neq \perp$ with probability $\geq 1 - \delta$
2. if $\mathcal{A}$ is ϵ-close to $x \in \{0,1\}^n$ and if $i \neq \perp$, then $x_i = 1$ with probability $\geq 1 - \gamma$

Complexity: $\mathrm{O}\left(\frac{1}{\left(\frac{1}{2}-\epsilon\right)^2} \cdot \sqrt{\frac{1}{\beta}} \cdot \log \frac{1}{\gamma\delta}\right)$ invocations of the A_i

Procedure AllInputs(n, $\mathcal{A}$, ϵ)

$n \in \mathbb{N}$, $\mathcal{A} : n$ algorithms, $\epsilon > 0$

```
 1: for i ← to n do
 2:    run A_i
 3:    x̃_i ← result of A_i
 4: for k ← 1 to log(ε(log n)^2) do
 5:    ε' ← ε/2^{k−1}
 6:    for ℓ ← 1 to 1.7ε'n do
 7:       i ← RobustFind(n, A(x̃), ε, 0.3ε', 1/8, 1/8)
 8:       if i ≠⊥ then
 9:          x̃_i ← 1 − x̃_i
10: for m ← n/(log n)^2 down to 1 do
11:    i ← RobustFind(n, A(x̃), ε, m/n, 1/(20n), 1/(20n))
12:    if i ≠⊥ then
13:       x̃_i ← 1 − x̃_i
14: return x̃
```

which operates like A_i but outputs 1 when A_i would have output 0 and vice versa. Define $A_i(b)$ by $A_i(0) := A_i$ and $A_i(1) = \bar{A}_i$. If we plug the right bit x_i into A_i, then for all A_i we expect output 0: for the unique good $x \in \{0,1\}^n$, $\mathcal{A}(x) := (A_1(x_1), \ldots, A_n(x_n))$ is ϵ-close to 0^n by the following notion of closeness:

Definition 5. *For $\epsilon < 1/2$ and decision algorithms $\mathcal{A} = (A_1, \ldots, A_n)$, we say $\mathcal{A}$ is* ϵ-close *to $x \in \{0,1\}^n$ if $\Pr[A_i$ outputs $x_i] \geq 1 - \epsilon$ for all $i \in [n]$.*

Our algorithm builds on a robust quantum search algorithm by Høyer, Mosca, and de Wolf [5], which we call RobustFind. This subroutine takes a vector $\mathcal{A}$ of n quantum algorithms and in the good case returns an index i so that the "high probability" output of A_i is 1. This allows us to verify a purported solution $\tilde{x} \in \{0,1\}^n$ by running RobustFind on $\mathcal{A}(\tilde{x})$ to find differences with the real input x. In fact, adjusting the parameters to RobustFind as we move closer and closer to a good solution, AllInputs (as defined by the pseudo code on page 599) manages to construct the unique x with high probability. Note that RobustFind is the only quantum component of our otherwise classical algorithm.

The first step of our algorithm (Lines 1–3) is to classically sample each i once and to store this initial approximation into a variable $\tilde{x}_i$. We call $i \in [n]$ a *bad* index if $x_i \neq \tilde{x}_i$. The following rounds of the algorithm (Lines 4–9) use

RobustFind with error probabilities $\gamma = \delta = 1/8$ and a decreasing estimate of the number of bad indices, β. This way we refine $\tilde{x}$ until we have fewer than $n/(\log n)^2$ bad indices. At that point, we can afford to set γ and δ to very small values to eliminate all remaining bad indices with high probability.

Success Probability. Let B_0 denote the random variable counting the number of bad indices after Line 3 and let B_k denote the random variable of the number of bad indices after the iteration k of the `for` loop in Lines 4–9. By $\mathcal{G}_k$ we denote the event $B_k \leq n\epsilon/2^{k-1}$. We have

$$\Pr[\mathcal{G}_{k_{\max}}] \geq \Pr[\mathcal{G}_0] \prod_{k=1}^{k_{\max}} \Pr[\mathcal{G}_k | \mathcal{G}_{k-1}] \quad . \tag{1}$$

We now show that $\Pr[\mathcal{G}_k|\mathcal{G}_{k-1}]$ is large by means of Chernoff bounds on the number of bad indices that we find in round k and the number of errors we make. For $k = 0$, we know that $\mathrm{E}[B_0] \leq \epsilon n$ and thus $\Pr[B_0 \leq 2\epsilon n] \geq 0.9$. In round k, we want to reduce the upper bound on the number of bad indices from $2n\epsilon/2^{k-1}$ to $n\epsilon/2^{k-1}$. Let E_k denote the random variable of the number of errors that we make in round k, i.e., the number of wrongly identified bad indices. Similarly, let C_k denote the random variable of the number of correctly identified bad indices. Then $B_k = B_{k-1} - C_k + E_k$, so

$$\Pr[\mathcal{G}_k|\mathcal{G}_{k-1}] \geq \Pr[E_k \leq 1.1\gamma r | \mathcal{G}_{k-1}] \cdot \Pr\left[C_k \geq B_{k-1} - \frac{n\epsilon}{2^{k-1}} + 1.1\gamma r \,\middle|\, \mathcal{G}_{k-1}\right] \tag{2}$$

We choose the number of repetitions of RobustFind to be $r := 1.7n\epsilon/2^{k-1}$ and our lower bound on the fraction of bad indices to be $\beta := 0.3\epsilon/2^{k-1}$. Then $\mathrm{E}[E_k|\mathcal{G}_{k-1}] \leq \gamma r$ and $\Pr[E_k \leq 1.1\gamma r | \mathcal{G}_{k-1}] \geq 1 - e^{-\Omega(r)}$. To bound the second factor in (2), we need to take into account that we have no guarantee on the success probability of a single RobustFind invocation if the number of bad indices falls below βn. However, if this happens, then $C_k \geq B_{k-1} - \beta n \geq B_{k-1} - n\epsilon/2^{k-1} + 1.1\gamma r$, i.e., the second factor in (2) is trivially 1. Therefore it is safe to assume that each invocation of RobustFind has success probability at least $(1-\gamma)(1-\delta)$. Hence,

$$\mathrm{E}[B_{k-1} - C_k + 1.1\gamma r | \mathcal{G}_{k-1}] \leq 2\frac{n\epsilon}{2^{k-1}} - (1-\gamma)(1-\delta)r + 1.1\gamma r \leq 0.9\frac{n\epsilon}{2^{k-1}}$$
$$\text{and } \Pr\left[B_{k-1} - C_k + 1.1\gamma r \leq \frac{n\epsilon}{2^{k-1}} \,\middle|\, \mathcal{G}_{k-1}\right] \geq 1 - e^{-\Omega(r)} \quad .$$

Altogether, this establishes $\Pr[\mathcal{G}_k|\mathcal{G}_{k-1}] \geq 1 - e^{-\Omega(r)}$.

Substituting this bound into (1) we obtain, with $k_{\max} = \log(\epsilon(\log n)^2)$ and $r = 1.7n\epsilon/2^{k-1} = \Omega(n/(\log n)^2)$,

$$\Pr[\mathcal{G}_{k_{\max}}] \geq \Pr[\mathcal{G}_0] \left(1 - e^{-\Omega(r)}\right)^{k_{\max}} \geq 0.9 \left(1 - \frac{\log(\epsilon(\log n)^2)}{e^{\Omega(n/(\log n)^2)}}\right) = 0.9 - \mathrm{o}(1).$$

Hence, for large n with probability 0.8 we have at most $n/(\log n)^2$ bad indices at the end of the `for` loop in Lines 4–9. In this case, we will find with

constant probability all bad indices by making the individual error probability in RobustFind so small that we can use a union bound: we determine each of the remaining bad indices with error probability $1/(10n)$. This implies an overall success probability $\geq 0.8 \cdot 0.9 > 2/3$.

Complexity. We bound the number of queries to f in Lines 4–9 as follows:

$$\sum_{k=1}^{k_{\max}} \sum_{\ell=1}^{n\epsilon/2^{k-1}} C \frac{1}{\left(\frac{1}{2}-\epsilon\right)^2} \sqrt{\frac{1}{\epsilon/2^k}} \leq C' \frac{\sqrt{\epsilon}}{\left(\frac{1}{2}-\epsilon\right)^2} n \sum_{k=1}^{\infty} \frac{1}{2^{k/2}} = \mathrm{O}\left(\frac{n}{\left(\frac{1}{2}-\epsilon\right)^2}\right)$$

for some constants C, C'. Lines 10–13 result in

$$\mathrm{O}\left(\sum_{m=1}^{n/(\log n)^2} \frac{1}{\left(\frac{1}{2}-\epsilon\right)^2} \sqrt{\frac{n}{m}} \log n\right) = \mathrm{O}\left(\frac{n}{\left(\frac{1}{2}-\epsilon\right)^2}\right)$$

many queries. Therefore, the total query complexity of AllInputs is $\mathrm{O}(n/(1/2-\epsilon)^2)$.

4 Making Quantum Algorithms Robust

4.1 Inputs Computed by Quantum Algorithms

Here we state a few corollaries of Theorem 2. First, once we have recovered the input x we can compute any function of x without further queries, hence

Corollary 2. *For every $f : \{0,1\}^n \to \{0,1\}$, there is a robust quantum algorithm that computes f using $\mathrm{O}(n)$ queries.*

In particular, PARITY can be robustly quantum computed with $\mathrm{O}(n)$ queries while it takes $\Omega(n \log n)$ queries classically [3].

Second, in the context of the direct-sum problem, the complexity of quantum computing a vector of instances of a function scales linearly with the complexity of one instance.

Corollary 3 (Direct Sum). *If there exists a T-query bounded-error quantum algorithm for f, then there is an $\mathrm{O}(Tn)$-query bounded-error quantum algorithm for n independent instances of f.*

As mentioned, the best classical upper bound has an additional factor of $\log n$, and this is optimal in a classical black-box setting.

Thirdly, all *symmetric* functions can be computed robustly on a quantum computer with the same asymptotic complexity as non-robustly. A function is symmetric if its value only depends on the Hamming weight of the input. Let $\Gamma(f) := \min\{|2k-n+1| : f \text{ changes value if the Hamming weight of the input changes from } k \text{ to } k+1\}$. The non-robust algorithm for computing f with $\mathrm{O}(\sqrt{n(n-\Gamma(f))})$ queries [1–Theorem 4.10] can be made robust by a similar algorithm as the one used in the proof of our Theorem 2, giving:

Theorem 3. *For every symmetric function f, there is a robust quantum algorithm that computes f using $\mathrm{O}(\sqrt{n(n-\Gamma(f))})$ quantum queries.*

4.2 Multiple Noisy Copies

As mentioned in the introduction, the assumption that we have a bounded-error algorithm A_i for each of the input bits x_i also covers the model of [11] where we have a sequence $y_{i,1}, \ldots, y_{i,m}$ of noisy copies of x_i. These we can query by means of a mapping $|i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle|y_{i,j}\rangle$. Here we spell out this connection in some more detail. First, by a Chernoff bound, choosing $m := \mathrm{O}(\log(n)/\epsilon^2)$ implies that the average $\overline{y}_i := \sum_{j=1}^{m} y_{i,j}/m$ is close to x_i with very high probability: $\Pr[|\overline{y}_i - x_i| \geq 2\epsilon] \leq 1/(100n)$. By the union bound, with probability 99/100 this closeness will hold for all $i \in [n]$ simultaneously. Assuming this is the case, we implement the following unitary mapping using one query: $A_i : |0^{\log(m)+1}\rangle \mapsto \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |j\rangle|y_{i,j}\rangle$. Measuring the last qubit of the resulting state gives x_i with probability at least $1 - 2\epsilon$. Hence, we can run our algorithm from Section 3 and recover x using $\mathrm{O}(n)$ queries to the $y_{i,j}$. Similarly, all consequences mentioned in Section 4.1 hold for this multiple-noisy-copies model as well.

5 Making Approximating Polynomials Robust

Theorem 4. $\mathrm{rdeg}_{1,2}(f) = \mathrm{O}(n)$ *for every* $f : \{0,1\}^n \to \{0,1\}$.

Proof. By Corollary 2 and the discussion in Section 4.2, f has an $\mathrm{O}(n)$-query robust quantum algorithm in the multiple-noisy-copies model that operates on $\mathrm{O}(\log n)$ copies. By Lemma 2 this induces a type-1 robust polynomial for f of degree $\mathrm{O}(n)$. And finally, by Corollary 1 there also exists a degree-$\mathrm{O}(n)$ type-2 robust polynomial for f. □

In particular, this shows that for functions with approximate degree $\Theta(n)$ we can make the approximating polynomial robust at only constant factor overhead in the degree. This case includes explicit functions like PARITY and MAJORITY, but also random (hence almost all) functions. It is open whether approximating polynomials can *always* be made robust at only a constant overhead in the degree. The best we can do is show that a non-robust degree-d approximating polynomial can be made robust at a cost of a factor $\mathrm{O}(\log d)$. Our proof makes use of the well known notion of *certificate complexity*.

Definition 6. *An assignment* $C : S \to \{0,1\}$ *of values to some subset* $S \subseteq [n]$ *of the* n *variables is* consistent *with* $x \in \{0,1\}^n$ *if* $x_i = C(i)$ *for all* $i \in S$. *For* $b \in \{0,1\}$, *a* b-certificate *for* f *is an assignment* C *such that* $f(x) = b$ *whenever* x *is consistent with* C. *The* size *of* C *is* $|S|$, *the cardinality of* S. *The* certificate complexity $C_x(f)$ of f on x *is the size of a smallest* $f(x)$*-certificate that is consistent with* x. *The* certificate complexity *of* f *is* $C(f) = \max_x C_x(f)$.

Lemma 3. *Let* p *be an* ϵ*-approximating polynomial for* $f : \{0,1\}^n \to \{0,1\}$, *and* $c = C(f)$ *be the certificate complexity of* f. *If* $x \in \{0,1\}^n$ *and* $z \in [0,1]^n$ *satisfy* $|x_i - z_i| \leq 1/10c$ *for all* $i \in [n]$, *then* $|p(z) - f(x)| \leq \epsilon + 2/15$.

Proof. Consider a certificate C for x of size c. We will use x^C and $x^{\overline{C}}$ to denote the parts of x corresponding to C and to its complement, respectively, and write

$x = x^C x^{\overline{C}}$. If $y \in \{0,1\}^n$ is chosen according to the z-distribution ($y_i = 1$ with probability z_i), then

$$p(z) = \mathrm{E}_y[p(y)] = \sum_{y^C y^{\overline{C}}} \Pr[y^C] \Pr[y^{\overline{C}}] p(y^C y^{\overline{C}}) = \sum_{y^{\overline{C}}} \Pr[y^{\overline{C}}] \cdot \mathrm{E}_{y^C}[p(y^C y^{\overline{C}})] \ .$$

Now consider the expectation $\mathrm{E}_{y^C}[p(y^C y^{\overline{C}})]$, where $y^{\overline{C}} \in \{0,1\}^{n-c}$ is fixed, while the y^C-bits are still chosen according to the z-distribution. Consider the c-variate polynomial obtained from p by fixing the bits in $y^{\overline{C}}$. Since the "error" in the z^C-variables is at most $1/10c$, we have $\Pr[y^C = x^C] \geq (1-1/10c)^c \geq 9/10$, so $|\mathrm{E}_{y^C}[p(y^C y^{\overline{C}})] - p(x^C y^{\overline{C}})| \leq (1/10)(4/3) = 2/15$. But $f(x^C y^{\overline{C}}) = f(x)$, because the input $x^C y^{\overline{C}}$ satisfies the same certificate as x. Hence

$$|\mathrm{E}_{y^C}[p(y^C y^{\overline{C}})] - f(x)| \leq |\mathrm{E}_{y^C}[p(y^C y^{\overline{C}})] - p(x^C y^{\overline{C}})| + |p(x^C y^{\overline{C}}) - f(x)| \leq 2/15 + \epsilon,$$

and also $|p(z) - f(x)| \leq \epsilon + 2/15$. □

This lemma implies that we can make a non-robust approximating polynomial robust at the cost of a factor of $\mathrm{O}(\log C(f))$ in the degree (replace each variable by a $\mathrm{O}(\log C(f))$-degree error-reducing polynomial). Since $C(f)$ and $\widetilde{\deg}(f)$ are polynomially related ($C(f) = \mathrm{O}(\widetilde{\deg}(f)^4)$, see [2]), we obtain:

Theorem 5. $\mathrm{rdeg}_{1,2}(f) = \mathrm{O}(\widetilde{\deg}(f) \cdot \log \widetilde{\deg}(f))$.

6 Summary and Open Problems

The main results of this paper are as follows:

- For every n-bit Boolean function f there is an n-variate polynomial p of degree $\mathrm{O}(n)$ that *robustly* approximates it, i.e., $p(x)$ remains close to $f(x)$ if we slightly vary the n inputs.
- There is an $\mathrm{O}(n)$-query quantum algorithm that *robustly* recovers n noisy input bits. Hence every n-bit function can be quantum computed with $\mathrm{O}(n)$ queries in the presence of noise. This contrasts with the classical case, where most functions need $\Theta(n \log n)$ queries.

Note that the use of the robust Or algorithm by [5] is not necessary for recovering the whole input. It would suffice to use Grover's algorithm, that runs in time $\sqrt{n/t}$ when there are t marked items. When we know an estimate of t, like in our $\log n$ rounds algorithm, we can make the error of a single query as small as $1/\sqrt{n/t}$ at the cost of a $\log(n/t)$ multiplicative factor. Standard analysis shows that in this case Grover's algorithm behaves as the robust Or algorithm. However this way we would not obtain the results about every symmetric function (Theorem 3).

We mention some open problems. First, in contrast to the classical case (Parity) we do not know of any function where making a quantum algorithm

robust costs more than a constant factor. Such a constant overhead suffices in the case of symmetric functions and functions whose approximate degree is $\Omega(n)$. It is conceivable that quantum algorithms (and polynomials) can *always* be made robust at a constant factor overhead. Proving or disproving this would be very interesting. We are not aware of a direct "closed form" or other natural way to describe a robust degree-n polynomial for the parity of n bits, but can only infer its existence from the existence of a robust quantum algorithm. Given the simplicity of the non-robust representing polynomial for PARITY, one would hope for a simple closed form for robust polynomials for PARITY as well.

Finally, we have chosen our model of a noisy query such that we can coherently make a query and reverse it. It is not clear to what extent non-robust quantum algorithms can be made resilient against decohering queries, since the usual transformations to achieve fault-tolerant quantum computation do not immediately apply to the query gate, which acts on a non-constant number of quantum bits simultaneously.

Acknowledgments. We thank Peter Høyer for inspiring initial discussions that led to our main result, and Michele Mosca for sending us a version of [6].

References

1. R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. Earlier version in FOCS 98.
2. H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288(1):21–43, 2002.
3. U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.
4. L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM STOC*, pages 212–219, 1996.
5. P. Høyer, M. Mosca, and R. de Wolf. Quantum search on bounded-error inputs. In *Proceedings of 30th ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 291–299. Springer, 2003.
6. K. Iwama, R. Putra, and S. Yamashita. Quantum query complexity of biased oracles. Unpublished manuscript and talk at EQIS conference, September 2003.
7. E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.
8. G. L. Long, Y. S. Li, W. L. Zhang, and C. C. Tu. Dominant gate imperfection in Grover's quantum search algorithm. *Physical Review A*, 61:042305, 2000.
9. N. Nisan and M. Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, 1994. Earlier version in STOC'92.
10. N. Shenvi, K. R. Brown, and K. B. Whaley. Effects of a random noisy oracle on search algorithm complexity. *Physical Review A*, 68:052313, 2003.
11. M. Szegedy and X. Chen. Computing Boolean functions from multiple faulty copies of input bits. In *Proceedings of 5th LATIN*, volume 2286 of *Lecture Notes in Computer Science*, pages 539–553, 2002.

Quantum Interactive Proofs with Competing Provers

Gus Gutoski and John Watrous

Department of Computer Science, University of Calgary, Alberta, Canada

Abstract. This paper studies quantum refereed games, which are quantum interactive proof systems with two competing provers: one that tries to convince the verifier to accept and the other that tries to convince the verifier to reject. We prove that every language having an ordinary quantum interactive proof system also has a quantum refereed game in which the verifier exchanges just one round of messages with each prover. A key part of our proof is the fact that there exists a single quantum measurement that reliably distinguishes between mixed states chosen arbitrarily from disjoint convex sets having large minimal trace distance from one another. We also show how to reduce the probability of error for some classes of quantum refereed games.

1 Introduction

A *refereed game* consists of a conversation between a computationally bounded verifier and two computationally unbounded provers regarding some input string x. The two provers use their unbounded computational power to compete with each other: one prover, called the *yes-prover*, attempts to convince the verifier to accept x, while the other prover, called the *no-prover*, attempts to convince the verifier to reject x. At the end of the interaction, the verifier decides whether to accept or reject the input x, effectively deciding which of the provers wins the game. Such games represent games of incomplete information; the messages exchanged between one prover and the verifier are considered to be hidden from the other player.

A language L is said to have a refereed game with error ε if there is a polynomial-time verifier satisfying the the following conditions. For each string $x \in L$, there exists a yes-prover that can always convince the verifier to accept x with probability at least $1 - \varepsilon$, regardless of the no-prover's strategy, and for each $x \notin L$, there exists a no-prover that can always convince the verifier to reject x with probability at least $1 - \varepsilon$, regardless of the yes-prover's strategy. A *turn* for one of the provers consists of a message from the verifier to that prover, followed by a response from that prover back to the verifier. One may consider the case where the provers turns are played sequentially or in parallel.

The refereed games model is based on the interactive proof system model [1, 2, 3, 4], which has a rich history that we will not survey here. The refereed games model, and variations on this model, were considered in the classical case in Refs. [5, 6, 7, 8, 9, 10], among others. Much of what is known about

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 605–616, 2005.

the complexity-theoretic aspects of the classical refereed games model is due to Feige and Kilian [10]. The class of languages having classical refereed games in which the provers may play any polynomial number of turns coincides with EXP (deterministic time $2^{p(n)}$ for some polynomial p). The simulation of EXP by a polynomial-turn refereed game is due to Feige and Kilian [10], and is based on arithmetization technique developed by Lund, Fortnow, Karloff and Nisan [11] and used in proofs of IP = PSPACE [12, 13]. The containment of this class in EXP is due to Koller and Megiddo [8]. On the other hand, the class of languages having games in which the provers play precisely one turn each, with the turns played in parallel, coincides with PSPACE [10]. Apparently little is known about the expressive power of classical refereed games intermediate between these two extremes. For instance, games with a constant number of prover turns may correspond to PSPACE, EXP, or some complexity class between the two.

Similar to the classical case, quantum refereed games are based on the quantum interactive proof system model [14, 15]. Quantum refereed games differ from classical ones in that the provers and the verifier may perform quantum computations and exchange quantum messages. Our two main motives for considering the quantum refereed games model are to better understand the power of quantum interactive proof systems and to examine the effect of quantum information on the complexity of finding strategies for two-player games.

The main result of this paper establishes that any language having a quantum interactive proof system also has a quantum refereed game with exponentially small probability of error wherein each prover plays just one turn (with the yes-prover playing first). An interesting fact about the resulting game from the point of view of understanding quantum interactive proofs is that entanglement between the provers and the verifier does not play any role in this game, and may without loss of generality be assumed not to exist. More specifically, the game we define has the following general form: the yes-prover sends the verifier a mixed quantum state, the verifier processes this state and sends some state to the no-prover, and the no-prover measures the state and sends a classical result to the verifier. The verifier checks the result of the measurement and accepts or rejects.

A key ingredient for our result is an information-theoretic assertion stating that there exists a quantum measurement that can reliably distinguish between states chosen from two disjoint convex sets of quantum states. This assertion generalizes a well-known fact about the relation between the trace distance between two states and their distinguishability, and may be viewed as a quantitative version, from the point of view of quantum information theory, of the fact from convex analysis that disjoint convex sets are separated by some hyperplane.

The remainder of this paper is organized as follows. We begin by defining quantum refereed games in Sect. 2. In Sect. 3 we prove the fact concerning measurements distinguishing convex sets mentioned previously. Using this fact, we then prove in Sect. 4 that a two-turn quantum refereed game exists for any language L having a quantum interactive proof system. In Sect. 5 we describe a method for error reduction in two-turn quantum refereed games. The paper

concludes with Sect. 6, which mentions some open problems about quantum refereed games.

2 Definitions

In this section we define the quantum refereed games model and some complexity classes based on this model. Throughout the paper we assume all strings are over the alphabet $\Sigma = \{0, 1\}$. For $x \in \Sigma^*$, $|x|$ denotes the length of x. We let *poly* denote the set of polynomial-time computable functions $f : \mathbb{N} \to \mathbb{N} \setminus \{0\}$ for which there exists a polynomial p such that $f(n) \leq p(n)$ for all n. We also let 2^{-poly} denote the set of polynomial-time computable functions ε such that $\varepsilon(n) = 2^{-f(n)}$ for all n for some $f \in poly$.

The model for quantum computation that provides a basis for quantum refereed games is the quantum circuit model, with which we assume the reader is familiar. As mentioned in Sect. 1, a quantum refereed game has a verifier V and two competing provers Y and N. Each of V, Y, and N is defined by a mapping on input strings $x \in \Sigma^*$ where $V(x)$, $Y(x)$, and $N(x)$ are each sequences of quantum circuits. The circuits in these sequences are assumed to be composed only of gates taken from some universal set of quantum gates. Thus, each of the circuits implements a unitary operation on its input qubits. However, we lose no generality by allowing only unitary operations because arbitrary admissible quantum operations, including measurements, can be simulated by unitary circuits as described in Ref. [16].

For each prover, the qubits upon which that prover's circuits act are partitioned into two sets: one set of qubits is private to that prover and the other is shared with the verifier. These shared qubits act as a quantum channel between the verifier and that prover. No restrictions are placed on the complexity of the provers' circuits, which captures the notion that the provers' computational power is unbounded—each of the provers' circuits can be viewed as an arbitrary unitary operation.

The qubits on which the verifier's circuits act are partitioned into three sets: one set is private to the verifier and two sets are shared with each of the provers. One of the verifier's private qubits is designated as the *output qubit*. At the end of the game, acceptance is dictated by a measurement of the output qubit in the computational basis. We also require that the verifier's sequence of circuits $V(x)$ be generated by a polynomial-time Turing machine on input x. This uniformity constraint captures the notion that the verifier's computational power is limited.

In addition to the verifier and provers, a quantum refereed game consists of a *protocol* that dictates the number and order of turns taken by the provers. The circuits in the verifier's and provers' sequences are applied to the initial state in which each qubit is in state $|0\rangle$ in such a way as to implement the protocol of the game.

The games we study in this paper have the following protocol: a message from the yes-prover to the verifier, a message from the verifier to the no-prover, and a message from the no-prover the the verifier. Quantum refereed games that follow

this protocol will be called *short quantum games*. We note that entanglement between the provers and the verifier is immaterial in games of this form—each prover takes only one turn, and thus has no need to remember anything after his turn ends. Thus, when convenient, we may assume that the provers do not have private qubits but instead may perform arbitrary admissible quantum operations (i.e., completely positive trace-preserving maps) on their message qubits.

We now define the complexity class SQG based on short quantum games of the type just described. For $c, s : \mathbb{N} \to [0,1]$, the set $\mathsf{SQG}(c,s)$ consists of all languages $L \subseteq \Sigma^*$ for which there exists a verifier V for a short quantum game such that the following conditions hold: (i) there exists a yes-prover Y such that, for all no-provers N and all $x \in L$, $Y(x)$ convinces $V(x)$ to accept x with probability at least $1 - c(|x|)$, and (ii) there exists a no-prover N such that, for all yes-provers Y and all $x \notin L$, $N(x)$ convinces $V(x)$ to reject x with probability at least $1 - s(|x|)$. The functions c and s are called the *completeness error* and *soundness error*, respectively. We define $\mathsf{SQG}\left(2^{-poly}, 2^{-poly}\right)$ to be the set of all languages $L \subseteq \Sigma^*$ such that $L \in \mathsf{SQG}(\varepsilon, \varepsilon)$ for every $\varepsilon \in 2^{-poly}$. Let us also write SQG as shorthand for $\mathsf{SQG}\left(2^{-poly}, 2^{-poly}\right)$.

The class QIP contains all problems having single-prover quantum interactive proof systems as in Ref. [15]. The main complexity-theoretic result of the present paper states that $\mathsf{QIP} \subseteq \mathsf{SQG}$. We prove this result by exhibiting a short quantum game that solves a promise problem called the CLOSE-IMAGES problem, which is known to be complete for QIP [15]. It is convenient for us to use the formulation of this problem based on the one found in Ref. [17].

The promise problem CLOSE-IMAGES is defined for any desired $\varepsilon \in 2^{-poly}$ as follows. Given are descriptions of two mixed state quantum circuits Q_0 and Q_1, which both implement some admissible (i.e., completely positive and trace-preserving) transformation from n qubits to m qubits. The promise is that either (i) there exist n-qubit mixed states ρ_0 and ρ_1 such that $Q_0(\rho_0) = Q_1(\rho_1)$, or (ii) for all n-qubit mixed states ρ_0 and ρ_1, the states $Q_0(\rho_0)$ and $Q_1(\rho_1)$ have fidelity squared at most $\varepsilon(n)$. In other words, the images of Q_0 and Q_1 are either overlapping or are far apart. The goal is to accept when case (i) holds and reject when case (ii) holds.

3 Distinguishing Convex Sets of Quantum States

We motivate discussion in this section by pointing out that, for any mixed-state quantum circuit Q, the image $\mathcal{A} = \{Q(\rho) : \rho \text{ a mixed state}\}$ of the admissible transformation associated with Q is a compact, convex set of mixed states. If the images of two circuits Q_0 and Q_1 are far apart, then one could reasonably hope that there is a quantum measurement that reliably distinguishes between outputs $Q_0(\rho_0)$ and $Q_1(\rho_1)$ of these transformations, with the measurement depending only on Q_0 and Q_1, and not on the choice of input states ρ_0 and ρ_1. In this section we prove that indeed there always exists such a measurement. More generally, we prove that given any two disjoint convex sets of mixed quantum states, there exists a single measurement that distinguishes states drawn

arbitrarily from one set from the other with success probability determined by the minimal trace distance between the sets. The short quantum game for the CLOSE-IMAGES problem we define in Sect. 4 relies heavily upon the existence of such a measurement.

Let us first begin with some notation. Given a finite dimensional Hilbert space $\mathcal{H}$, let $\mathbf{L}(\mathcal{H})$ denote the set of all linear operators on $\mathcal{H}$, let $\mathbf{H}(\mathcal{H})$ denote the set of all Hermitian operators on $\mathcal{H}$, let $\mathbf{Pos}(\mathcal{H})$ denote the set of all positive semidefinite operators on $\mathcal{H}$, and let $\mathbf{D}(\mathcal{H})$ denote the set of all density operators (i.e., unit trace positive semidefinite operators) on $\mathcal{H}$. For $A, B \in \mathbf{L}(\mathcal{H})$, define $\langle A, B \rangle = \operatorname{tr} A^\dagger B$. This is an inner product on $\mathbf{L}(\mathcal{H})$ that is sometimes called the Hilbert-Schmidt inner product.

For a vector $|\psi\rangle \in \mathcal{H}$, $\||\psi\rangle\|$ denotes the Euclidean norm of $|\psi\rangle$, and for an operator $A \in \mathbf{L}(\mathcal{H})$, $\|A\|$ denotes the operator norm of A. The trace norm of A, denoted $\|A\|_{\mathrm{tr}}$ is defined by $\|A\|_{\mathrm{tr}} = \operatorname{tr} \sqrt{A^\dagger A}$. The trace norm and the operator norm are dual to one another with respect to the Hilbert-Schmidt inner product, meaning that the following fact holds.

Fact 1. *For every $A \in \mathbf{L}(\mathcal{H})$,*

$$\|A\| = \max\{|\langle B, A\rangle| : B \in \mathbf{L}(\mathcal{H}),\ \|B\|_{\mathrm{tr}} \leq 1\} \quad ,$$
$$\|A\|_{\mathrm{tr}} = \max\{|\langle B, A\rangle| : B \in \mathbf{L}(\mathcal{H}),\ \|B\| \leq 1\} \quad .$$

See, for instance, Bhatia [18] for a proof of this fact.

The trace norm characterizes the distinguishability of a given pair of density matrices $\rho_0, \rho_1 \in \mathbf{D}(\mathcal{H})$ in the following sense. There exists a binary-valued quantum measurement such that if $\rho \in \{\rho_0, \rho_1\}$ is chosen uniformly at random, then the measurement correctly determines which of ρ_0 or ρ_1 was given with probability $\frac{1}{2} + \frac{1}{4}\|\rho_0 - \rho_1\|_{\mathrm{tr}}$. Furthermore, such a measurement is optimal in the sense that no other quantum measurement can possibly distinguish between ρ_0 and ρ_1 with a higher success rate. An immediate corollary of this fact is that for a given pair ρ_0 and ρ_1, there exists a measurement that correctly identifies a chosen state $\rho \in \{\rho_0, \rho_1\}$ with probability of correctness at least $\frac{1}{2}\|\rho_0 - \rho_1\|_{\mathrm{tr}}$, even if ρ is chosen by an adversary that knows the measurement.

Consider the following variant of the distinguishability problem: We are given $\rho \in \mathbf{D}(\mathcal{H})$ chosen from one of two disjoint convex sets of density operators $\mathcal{A}_0, \mathcal{A}_1 \subseteq \mathbf{D}(\mathcal{H})$, and we are asked to determine the set from which ρ was chosen. For simplicity we will assume $\mathcal{A}_0$ and $\mathcal{A}_1$ are closed sets. Under this assumption, it is meaningful to define the trace distance $\operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1)$ between $\mathcal{A}_0$ and $\mathcal{A}_1$ as the minimum of the quantity $\|\rho_0 - \rho_1\|_{\mathrm{tr}}$ over all choices of $\rho_0 \in \mathcal{A}_0$ and $\rho_1 \in \mathcal{A}_1$. We prove that there exists a single measurement with the property that if an arbitrary ρ is chosen from $\mathcal{A}_0$ with probability $1/2$, and otherwise ρ is chosen from $\mathcal{A}_1$, then the measurement correctly determines which set ρ was chosen from with probability at least $\frac{1}{2} + \frac{1}{4}\operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1)$. This fact therefore generalizes the fact concerning a single pair of quantum states mentioned above, as singleton sets are of course closed and convex. As above, this fact implies that if ρ is chosen from $\mathcal{A}_0 \cup \mathcal{A}_1$ in an arbitrary manner, even depending on the

measurement itself, then the measurement will correctly determine from which of $\mathcal{A}_0$ or $\mathcal{A}_1$ the state ρ was chosen with probability at least $\frac{1}{2}\operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1)$.

The proof of this fact begins with a well-known result from convex analysis, which informally states that there exists a separating hyperplane between any two disjoint convex sets. Typically, the separation result is stated in terms of the vector space $\mathbb{R}^n$, but it translates to $\mathbf{H}(\mathcal{H})$ for a given space $\mathcal{H}$ without complications, as $\mathbf{H}(\mathcal{H})$ may be identified with the vector space $\mathbb{R}^{m^2}$, for $m = \dim(\mathcal{H})$. Here we state a restricted variant of this fact that is most convenient for our purposes—see Rockafellar [19], for instance, for a more general statement.

Fact 2. *Let $\mathcal{A}, \mathcal{B} \subseteq \mathbf{H}(\mathcal{H})$ be disjoint convex sets with $\mathcal{A}$ compact and $\mathcal{B}$ open. Then there exists a Hermitian operator $H \in \mathbf{H}(\mathcal{H})$ and a real number $a \in \mathbb{R}$ such that $\langle H, X\rangle \geq a > \langle H, Y\rangle$ for all $X \in \mathcal{A}$ and $Y \in \mathcal{B}$.*

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

Theorem 1. *Let $\mathcal{A}_0, \mathcal{A}_1 \subseteq \mathbf{D}(\mathcal{H})$ be closed convex sets of density operators. Then there exist measurement operators $E_0, E_1 \in \mathbf{Pos}(\mathcal{H})$ with $E_0 + E_1 = I$ such that the following holds. For every pair $\rho_0 \in \mathcal{A}_0$ and $\rho_1 \in \mathcal{A}_1$, if ρ is chosen uniformly from $\{\rho_0, \rho_1\}$ and measured via the measurement $\{E_0, E_1\}$, the measurement will correctly determine whether $\rho \in \mathcal{A}_0$ or $\rho \in \mathcal{A}_1$ with probability at least $\frac{1}{2} + \frac{1}{4}\operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1)$.*

Proof. Let $d = \operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1)$. If $d = 0$, the theorem is trivially satisfied by the measurement defined by $E_0 = E_1 = \frac{1}{2}I$ (which is equivalent to a random coin-flip), so assume that $d > 0$. Let $\mathcal{A} = \mathcal{A}_0 - \mathcal{A}_1 = \{\rho_0 - \rho_1 : \rho_0 \in \mathcal{A}_0, \rho_1 \in \mathcal{A}_1\}$. Then $\mathcal{A}$ is a compact convex set of Hermitian operators and $\|X\|_{\mathrm{tr}} \geq d$ for every $X \in \mathcal{A}$. Let $\mathcal{B} = \{Y \in \mathbf{H}(\mathcal{H}) : \|Y\|_{\mathrm{tr}} < d\}$ denote the open ball of radius d in $\mathbf{H}(\mathcal{H})$ with respect to the trace norm. The sets $\mathcal{A}$ and $\mathcal{B}$ satisfy the conditions of Fact 2, and therefore there exists a Hermitian operator $H \in \mathbf{H}(\mathcal{H})$ and a real number $a \in \mathbb{R}$ such that $\langle H, X\rangle \geq a > \langle H, Y\rangle$ for all $X \in \mathcal{A}$ and $Y \in \mathcal{B}$. Because $Y \in \mathcal{B}$ if and only if $-Y \in \mathcal{B}$ for every Y, it follows that $-a < a$, and therefore $a > 0$.

Let $K = \frac{d}{a}H$. We therefore have that $\langle K, X\rangle \geq d$ for every $X \in \mathcal{A}$ and $\langle K, \frac{1}{d}Y\rangle < 1$ for every $Y \in \mathcal{B}$. As $\frac{1}{d}Y$ ranges over all Hermitian operators with trace norm smaller than 1, this implies $\|K\| \leq 1$ by Fact 1. Now, let $K^+, K^- \in \mathbf{Pos}(\mathcal{H})$ denote the positive and negative parts of K, meaning that they satisfy $K = K^+ - K^-$ and $\langle K^+, K^-\rangle = 0$. As $\|K\| \leq 1$ it follows that $K^+ + K^- \leq I$.

At this point we define $E_0, E_1 \in \mathbf{Pos}(\mathcal{H})$ as follows:

$$E_0 = K^+ + \frac{1}{2}(I - K^+ - K^-) \quad \text{and} \quad E_1 = K^- + \frac{1}{2}(I - K^+ - K^-) \ .$$

The operators E_0 and E_1 are both positive semidefinite and satisfy $E_0 + E_1 = I$, and therefore represent a binary-valued POVM.

Now suppose $\rho_0 \in \mathcal{A}_0$ and $\rho_1 \in \mathcal{A}_1$ are chosen arbitrarily, and ρ is chosen uniformly from the set $\{\rho_0, \rho_1\}$. Let C denote the event that the measurement

$\{E_0, E_1\}$ correctly determines which of ρ_0 and ρ_1 was selected. We have $\Pr[C] = \frac{1}{2}\langle E_0, \rho_0\rangle + \frac{1}{2}\langle E_1, \rho_1\rangle$, and therefore

$$\Pr[C] - \Pr[\neg C] = \frac{1}{2}\langle E_0 - E_1, \rho_0 - \rho_1\rangle = \frac{1}{2}\langle K, \rho_0 - \rho_1\rangle \geq \frac{d}{2} \ ,$$

with the inequality following from the fact that $\rho_0 - \rho_1 \in \mathcal{A}$. Consequently the measurement is correct with probability at least $\frac{1}{2} + \frac{d}{4}$ as required. □

As before, it follows from this theorem that the measurement $\{E_0, E_1\}$ will correctly identify an arbitrary choice of $\rho \in \mathcal{A}_0 \cup \mathcal{A}_1$ with probability at least $\frac{1}{2}\operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1)$.

4 A Short Quantum Game for QIP

In this section, we prove that any language with a quantum interactive proof system also has a short quantum game by solving the QIP-complete problem CLOSE-IMAGES from Sect. 2.

First, let us recall that the fidelity $F(\rho, \xi)$ between two quantum states $\rho, \xi \in \mathbf{D}(\mathcal{H})$ is defined as $F(\rho, \xi) = \left\|\sqrt{\rho}\sqrt{\xi}\right\|_{\mathrm{tr}}$. The following fact gives one relationship between the fidelity and the trace norm.

Fact 3 ([20]). *Let $\rho, \xi \in \mathbf{D}(\mathcal{H})$. Then*

$$1 - \frac{1}{2}\|\rho - \xi\|_{\mathrm{tr}} \leq F(\rho, \xi) \leq \sqrt{1 - \frac{1}{4}\|\rho - \xi\|_{\mathrm{tr}}} \ .$$

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

Theorem 2. $\mathsf{QIP} \subseteq \mathsf{SQG}\left(1/2, 2^{-poly}\right)$.

Proof. It suffices to show that CLOSE-IMAGES is in $\mathsf{SQG}(1/2, 2^{-poly})$. Suppose the input encodes mixed state quantum circuits Q_0 and Q_1, each mapping n qubits to m qubits. Let $\mathcal{H}$ and $\mathcal{K}$ be Hilbert spaces with dimensions 2^n and 2^m corresponding to the n input qubits and m output qubits respectively. We may view Q_0 and Q_1 as corresponding to admissible transformations $Q_0, Q_1 : \mathbf{D}(\mathcal{H}) \to \mathbf{D}(\mathcal{K})$. Let $\mathcal{A}_i = \{Q_i(\rho) : \rho \in \mathbf{D}(\mathcal{H})\} \subseteq \mathbf{D}(\mathcal{K})$ denote the image of Q_i for $i = 0, 1$. The sets $\mathcal{A}_0$ and $\mathcal{A}_1$ are closed, convex sets of density operators.

Consider the following verifier for a short quantum game:

1. Receive n-qubit registers X_0 and X_1 from the yes-prover.
2. Choose $i \in \{0, 1\}$ uniformly at random and apply Q_i to register X_i. Let the output be contained in an m-qubit register Y, which is then sent to the no-prover.
3. Receive a classical bit b from the no-prover. Accept if $b \neq i$ and reject if $b = i$.

If (Q_0, Q_1) is a "yes" instance of CLOSE-IMAGES then there exist $\rho_0, \rho_1 \in \mathbf{D}(\mathcal{H})$ such that $Q_0(\rho_0) = Q_1(\rho_1)$. The strategy for the yes-prover is to prepare the registers X_0 and X_1 in states ρ_0 and ρ_1, respectively, and to send them to the verifier in step 1 of the verifier's protocol. Because $Q_0(\rho_0) = Q_1(\rho_1)$, the state contained in the register Y is independent of i, so the no-prover can do no better than randomly guessing in step 3. The verifier will therefore accept with probability $1/2$ in this case.

If (Q_0, Q_1) is a "no" instance of CLOSE-IMAGES then for any desired $\varepsilon \in 2^{-poly}$ we are promised that

$$\sqrt{\varepsilon(n)} \geq \max_{\xi_0, \xi_1 \in \mathbf{D}(\mathcal{H})} \{F(Q_0(\xi_0), Q_1(\xi_1))\} \geq 1 - \frac{1}{2} \operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1) \ .$$

It follows that $\operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1) \geq 2 - 2\sqrt{\varepsilon(n)}$.

Regardless of the state of the registers X_0 and X_1 sent to the verifier by the yes-prover, we must have that the reduced state of the register Y sent to the no-prover is given by some state $\xi \in \mathcal{A}_0 \cup \mathcal{A}_1$, and moreover that $\Pr[\xi \in \mathcal{A}_0] = \Pr[\xi \in \mathcal{A}_1] = 1/2$. By Theorem 1 there exists a quantum measurement $\{E_0, E_1\}$ that correctly determines whether $\rho \in \mathcal{A}_0$ or $\rho \in \mathcal{A}_1$ with probability at least

$$\frac{1}{2} + \frac{1}{4} \operatorname{dist}(\mathcal{A}_0, \mathcal{A}_1) \geq 1 - \frac{\sqrt{\varepsilon(n)}}{2} \ .$$

The strategy for the no-prover is to perform the quantum measurement $\{E_0, E_1\}$ and send the result to the verifier in step 3. This causes the verifier to reject with probability at least $1 - \sqrt{\varepsilon(n)}/2$. As this argument holds for every $\varepsilon \in 2^{-poly}$, we have that the soundness error is 2^{-poly} as required. □

5 Error Reduction

Suppose that both the completeness and soundness error c and s of a refereed game are bounded below $1/2$ by an inverse polynomial. Then it follows from Chernoff bounds that these error probabilities can be made exponentially close to zero by repeating the game a polynomial number of times in succession and taking a majority vote. Of course, sequential repetition necessarily increases the number of turns in the game and so it is natural to ask if error reduction can be achieved without affecting the turn complexity of the game.

A natural approach to this task is to run many copies of the refereed game in parallel and to accept or reject based on the outcomes of the repetitions. This technique is purely classical and has been successfully applied to classical single- and multi-prover interactive proof systems (see for example Ref. [21] and the references therein). A potential problem with this technique is that the provers need not treat each repetition independently—they might try to correlate the parallel repetitions (or entangle them in the quantum case) in some devious way such that the completeness and/or soundness error does not decrease as desired.

In the quantum setting, the general case of this problem has not been completely solved. But for three-message single-prover quantum interactive proof systems with zero completeness error, Ref. [15] proves that parallel repetition followed by a unanimous vote does indeed achieve the exponential reduction in soundness error that one might expect, regardless of any possible entanglement by the prover among the parallel copies.

In this section, we prove that parallel repetition followed by a unanimous vote can be used to improve the error bounds for short quantum games by reducing the problem to error reduction for single-prover quantum interactive proof systems with three or fewer messages. The reduction is achieved by fixing a yes- or no-prover P that is guaranteed to win with a certain probability. By viewing the verifier-prover pair (V, P) as a new composite verifier, we are left with what is now effectively a one- or two-message quantum interactive proof system in which the opposing prover is the lone prover. We define a verifier-prover pair (V', P') that runs many copies of (V, P) in parallel and accepts based on a unanimous vote. We can then employ the error reduction result of Ref. [15] to prove that the error of the new game decreases exponentially in the number of repetitions.

We formalize this argument shortly, but first we require additional notation. Given finite-dimensional Hilbert spaces $\mathcal{H}$ and $\mathcal{K}$, let $\mathbf{L}(\mathcal{H}, \mathcal{K})$ denote the set of all linear operators mapping $\mathcal{H}$ to $\mathcal{K}$ and let $\mathbf{T}(\mathcal{H}, \mathcal{K})$ denote the set of all linear operators mapping the vector space $\mathbf{L}(\mathcal{H})$ to $\mathbf{L}(\mathcal{K})$. The trace norm can be extended to $\mathbf{T}(\mathcal{H}, \mathcal{K})$ as follows. For $T \in \mathbf{T}(\mathcal{H}, \mathcal{K})$,

$$\|T\|_{\mathrm{tr}} = \sup_{X \in \mathbf{L}(\mathcal{H}) \setminus \{0\}} \frac{\|T(X)\|_{\mathrm{tr}}}{\|X\|_{\mathrm{tr}}} \ .$$

Let $\mathcal{L}$ be a Hilbert space with $\dim(\mathcal{L}) = \dim(\mathcal{H})$ and let $I_{\mathbf{L}(\mathcal{L})}$ denote the identity transformation on $\mathbf{L}(\mathcal{L})$. Then for $T \in \mathbf{T}(\mathcal{H}, \mathcal{K})$, the *diamond norm* $\|T\|_\diamond$ of T is given by $\|T\|_\diamond = \left\|T \otimes I_{\mathbf{L}(\mathcal{L})}\right\|_{\mathrm{tr}}$. Further information on the diamond norm may be found in Kitaev, Shen, and Vyalyi [22]. The diamond norm satisfies several nice properties that the trace norm (extended to $\mathbf{T}(\mathcal{H}, \mathcal{K})$) does not. For example, the diamond norm is multiplicative with respect to tensor products: $\|T_1 \otimes T_2\|_\diamond = \|T_1\|_\diamond \|T_2\|_\diamond$ for any choice of transformations T_1 and T_2.

We are now prepared to give the main result of this section, whose proof is based on the proof of Theorem 6 of Ref. [15].

Theorem 3. $\mathsf{SQG}(c, s) \subseteq \mathsf{SQG}(kc, s^k) \cap \mathsf{SQG}(c^k, ks)$ *for any choice of* $c, s : \mathbb{N} \to [0, 1]$ *and* $k \in \mathit{poly}$.

Proof. We first prove that $\mathsf{SQG}(c, s) \subseteq \mathsf{SQG}(kc, s^k)$. Let $L \in \mathsf{SQG}(c, s)$ and let $V(x) = (V(x)_1, V(x)_2)$ be a verifier witnessing this fact. For the remainder of this proof, we assume that the input $x \in \Sigma^*$ is fixed. For brevity we drop the argument and write $V = (V_1, V_2)$ and use similar notation for the provers.

Let $V' = (V_1^{\otimes k}, V_2^{\otimes k})$ be a verifier that runs k copies of the protocol of V in parallel and accepts if and only if every one of the k copies accepts. We must show that V' has completeness error at most kc and soundness error at most s^k.

First consider the case $x \in L$. Let $Y = (Y_1)$ be a yes-prover that always convinces V to accept with probability at least $1 - c$. Let $Y' = (Y_1^{\otimes k})$ be a yes-prover that runs k independent copies of the protocol of Y in parallel. Then no no-prover can win any one of the k copies with probability greater than c and so by the union bound we know that the completeness error of the repeated game is at most kc.

Next consider the case $x \notin L$. Let $N = (N_1)$ be a no-prover that always convinces V to reject with probability at least $1 - s$. Let $N' = (N_1^{\otimes k})$ be a no-prover that runs k independent copies of the protocol of N in parallel. We now show that no yes-prover can win against N' using verifier V' with probability greater than s^k.

Let Π_{init} denote the projection of the entire system onto the all-$|0\rangle$ initial state. Then the projection $\Pi'_{\text{init}} = \Pi_{\text{init}}^{\otimes k}$ corresponds to the initial state of the repeated game. Let Π_{acc} denote the projection onto the states for which the output qubit belonging to V is 1. Then the projection $\Pi'_{\text{acc}} = \Pi_{\text{acc}}^{\otimes k}$ corresponds to the accepting state of V'. Let $\mathcal{V}_N$ denote the Hilbert space corresponding to the private qubits of V and the private and message qubits of N and let $\mathcal{M}_Y$ denote the Hilbert space corresponding to the yes-prover's message qubits. Define $T_N \in \mathbf{T}(\mathcal{V}_N \otimes \mathcal{M}_Y, \mathcal{M}_Y)$ as $T_N(X) = \text{tr}_{\mathcal{V}_N}(\Pi_{\text{init}})X(\Pi_{\text{acc}}V_2N_1V_1)$.

As mentioned earlier, we may view (V, N) as a new composite verifier and the yes-prover as the lone prover for some one-message quantum interactive proof system (i.e., a message from the prover to (V, N)). In this context, Lemma 7 of Ref. [15] asserts that the maximum probability with which any prover could convince the verifier (V, N) to accept x is precisely $\|T_N\|_\diamond^2$. Because (V, N) has soundness error at most s, we have $\|T_N\|_\diamond^2 \leq s$.

Define a similar transformation $T'_N \in \mathbf{T}((\mathcal{V}_N \otimes \mathcal{M}_Y)^{\otimes k}, \mathcal{M}_Y^{\otimes k})$ using V', N', Π'_{init}, and Π'_{acc}. It follows that $T'_N = T_N^{\otimes k}$. From the multiplicativity of the diamond norm, it follows that the maximum probability with which any prover could convince (V', N') to accept x is $\|T'_N\|_\diamond^2 = \left\|T_N^{\otimes k}\right\|_\diamond^2 = \|T_N\|_\diamond^{2k} \leq s^k$, which establishes the desired result.

Due to the symmetric nature of quantum refereed games, we can modify the above proof to show that $\mathsf{SQG}(c, s) \subseteq \mathsf{SQG}(c^k, ks)$. In particular, define the verifier V'' so that he rejects if and only if all k copies reject. For the case $x \notin L$, the proof that V'' has soundness error ks is completely symmetric to the proof that V' has completeness error kc.

For the case $x \in L$, we let Y and Y' be yes-players as above. Define the Hilbert spaces $\mathcal{V}_Y$ and $\mathcal{M}_N$ and the projections Π_{rej} and Π'_{rej} in the appropriate symmetric manner as per the above proof. The transformation $T_Y \in \mathbf{T}(\mathcal{V}_Y \otimes \mathcal{M}_N, \mathcal{M}_N)$ is defined as $T_Y(X) = \text{tr}_{\mathcal{V}_Y}(V_1Y_1\Pi_{\text{init}})X(\Pi_{\text{rej}}V_2)$.

As before, we may view (V, Y) as a new composite verifier and the no-prover as the lone prover for some quantum interactive proof system. The differences here are that the quantum interactive proof is now a two-message proof instead of a one-message proof (i.e., a message from (V, Y) to the prover followed by the prover's reply to (V, Y)) and that the prover's goal is now to convince the verifier (V, Y) to reject x instead of to accept x.

Fortunately, it is still straightforward to apply Lemma 7 of Ref. [15] to this quantum interactive proof system and so we may claim that the maximum probability with which any prover could convince the verifier (V, Y) to reject x is precisely $\|T_Y\|_\diamond^2$. That V'' has completeness error c^k follows as before. □

The proof of Theorem 3 can be extended to allow for the slightly more general protocol wherein the verifier sends a message to the yes-prover (via some circuit V_{init}) before the short quantum game commences. This extension follows from the fact that we can apply Lemma 7 of Ref. [15] to the augmented transformations

$$T_N(X) = \text{tr}_{\mathcal{V}_N}(V_{\text{init}}\Pi_{\text{init}})X(\Pi_{\text{acc}}V_2N_1V_1) \ ,$$
$$T_Y(X) = \text{tr}_{\mathcal{V}_Y}(V_1Y_1V_{\text{init}}\Pi_{\text{init}})X(\Pi_{\text{rej}}V_2) \ .$$

Combining Theorems 2 and 3 we obtain the following corollary, which is the main result of this paper.

Corollary 1. $\mathsf{QIP} \subseteq \mathsf{SQG}$.

Proof. Given a desired error bound 2^{-p} where $p \in \mathit{poly}$, choose $\varepsilon \in 2^{-\mathit{poly}}$ so that $p\varepsilon \leq 2^{-p}$. We have $\mathsf{QIP} \subseteq \mathsf{SQG}\,(1/2, \varepsilon) \subseteq \mathsf{SQG}\,(2^{-p}, 2^{-p})$. □

6 Conclusion

We introduced in this paper the quantum refereed game model of computation and gave a short quantum game with exponentially small error for languages with single-prover quantum interactive proof systems. However, we have only scratched the surface of the quantum games model, and many questions about it remain unanswered. Some examples follow.

- The two-turn game presented in this paper has an asymmetric protocol. Is there also a two-turn quantum refereed game for QIP in which the no-prover sends the first message, or in which the provers play one turn in parallel?
- It is known that $\mathsf{QIP} \subseteq \mathsf{EXP}$. How does SQG relate to EXP?
- We mentioned in Sect. 1 that classical refereed games characterize EXP [10], which implies that many-turn quantum refereed games are at least as powerful as EXP. What upper bounds can be proved on the power of refereed quantum games?
- We demonstrated that parallel repetition followed by a unanimous vote can reduce error for short quantum games. Is there a way to reduce the error in *any* quantum refereed game without affecting the number of turns in the game?

Acknowledgments

This research was supported by Canada's NSERC, the Canada Research Chairs program, the Canadian Institute for Advanced Research (CIAR), and a graduate student scholarship from the Province of Alberta.

References

1. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18** (1989) 186–208
2. Babai, L.: Trading group theory for randomness. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing. (1985) 421–429
3. Babai, L., Moran, S.: Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. Journal of Computer and System Sciences **36** (1988) 254–276
4. Ben-Or, M., Goldwasser, S., Kilian, J., Wigderson, A.: Multi-prover interactive proofs: how to remove intractability assumptions. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. (1988) 113–131
5. Reif, J.: The complexity of two-player games of incomplete information. Journal of Computer and System Sciences **29** (1984) 274–301
6. Feige, U., Shamir, A., Tennenholtz, M.: The noisy oracle problem. In: Advances in Cryptology – Proceedings of Crypto'88. Volume 403 of Lecture Notes in Computer Science., Springer–Verlag (1990) 284 – 296
7. Feige, U., Shamir, A.: Multi-oracle interactive protocols with constant space verifiers. Journal of Computer and System Sciences **44** (1992) 259–271
8. Koller, D., Megiddo, N.: The complexity of two-person zero-sum games in extensive form. Games and Economic Behavior **4** (1992) 528–552
9. Feigenbaum, J., Koller, D., Shor, P.: A game-theoretic classification of interactive complexity classes. In: Proceedings of the 10th Conference on Structure in Complexity Theory. (1995) 227–237
10. Feige, U., Kilian, J.: Making games short. In: Proceedings of the Twenty-Ninth annual ACM Symposium on Theory of Computing. (1997) 506 – 516
11. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM **39** (1992) 859–868
12. Shamir, A.: IP = PSPACE. Journal of the ACM **39** (1992) 869–877
13. Shen, A.: IP = PSPACE: simplified proof. Journal of the ACM **39** (1992) 878–880
14. Watrous, J.: PSPACE has constant-round quantum interactive proof systems. Theoretical Computer Science **292** (2003) 575–588
15. Kitaev, A., Watrous, J.: Parallelization, amplification, and exponential time simulation of quantum interactive proof system. In: Proceedings of the 32nd ACM Symposium on Theory of Computing. (2000) 608–617
16. Aharonov, D., Kitaev, A., Nisan, N.: Quantum circuits with mixed states. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing. (1998) 20–30
17. Rosgen, B., Watrous, J.: On the hardness of distinguishing mixed-state quantum computations. arXiv.org e-Print cs.CC/0407056 (2004)
18. Bhatia, R.: Matrix Analysis. Springer (1997)
19. Rockafellar, R.T.: Convex Analysis. Princeton University Press (1970)
20. Fuchs, C., van de Graaf, J.: Cryptographic distinguishability measures for quantum-mechanical states. IEEE Transactions on Information Theory **45** (1999) 1216–1227
21. Raz, R.: A parallel repetition theorem. SIAM Journal of Computing **27** (1998) 763–803
22. Kitaev, A., Shen, A., Vyalyi, M.: Classical and Quantum Computation. Volume 47 of Graduate Studies in Mathematics. American Mathematical Society (2002)

Roundings Respecting Hard Constraints

Benjamin Doerr

Mathematisches Seminar II, Christian–Albrechts–Universität zu Kiel,
D–24098 Kiel, Germany
bed@numerik.uni-kiel.de

Abstract. A problem arising in integer linear programming is to transform a solution of a linear system to an integer one which is "close". The customary model to investigate such problems is, given a matrix A and a $[0,1]$ valued vector x, to find a binary vector y such that $\|A(x-y)\|_\infty$ (the violation of the constraints) is small. Randomized rounding and the algorithm of Beck and Fiala are ways to compute such solutions y, whereas linear discrepancy is a lower bound measure.

In many applications one is looking for roundings that, in addition to being close to the original solution, satisfy some constraints without violation. The objective of this paper is to investigate such problems in a unified way. To this aim, we extend the notion of linear discrepancy, the theorem of Beck and Fiala and the method of randomized rounding to this setting.

Whereas some of our examples show that additional hard constraints may seriously increase the linear discrepancy, the latter two sets of results demonstrate that a reasonably broad notion of hard constraints may be added to the rounding problem without worsening the obtained solution significantly.

Of particular interest might be our results on randomized rounding. We provide a simpler way to randomly round fixed weight vectors (cf. Srinivasan, FOCS 2001). It has the additional advantage that it can be derandomized with standard methods.

1 Introduction and Results

1.1 Rounding Problems, Randomized Rounding and Linear Discrepancy

Solving integer linear programs (ILPs) is NP–hard, solving linear programs without integrality constraints is easy (in several respects). Therefore a natural and widely used technique is to solve the linear relaxation of the ILP and then transform (typically by rounding) its solution into an integer one.

In doing so, one usually has to accept that the constraints are violated to some extent. There are several ways to deal with such violations, including simply accepting them, repairing them and preventing them by solving a linear program with stricter constraints in the first step. We do not want to go into detail here, but note that in any case the central theme is rounding the solution of the

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 617–628, 2005.

relaxation in such a way that the constraints are violated not too much. The underlying theoretical concept is the one of linear discrepancy.

Definition 1 (Linear Discrepancy Problem). *Given a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $x \in [0,1]^n$, find a $y \in \{0,1\}^n$ such that $\|A(x-y)\|_\infty$ is small. We write*

$$\operatorname{lindisc}(A,x) := \min_{y\in\{0,1\}^n} \|A(x-y)\|_\infty,$$
$$\operatorname{lindisc}(A) := \max_{x\in[0,1]^n} \operatorname{lindisc}(A,x).$$

Thus $\operatorname{lindisc}(A,x)$ is the rounding error inflicted by an optimal rounding of x. It is known that this can be quite high. Spencer [Spe87] gives an example of a binary $n \times n$ matrix A such that $\operatorname{lindisc}(A) = \Omega(\sqrt{n})$.

Whereas linear discrepancies provide bounds on how good roundings can possibly be, there are a number of positive results. A very general approach is the one of *randomized rounding* introduced in Raghavan and Thompson [RT87, Rag88]. Here the integer vector y is obtained from the solution x of the relaxation by rounding each component j independently with probabilities derived from y_j. In particular, if $x \in [0,1]^n$, we have $\Pr(y_j = 1) = x_j$ and $\Pr(y_j = 0) = 1 - x_j$ for all j.

Since the components are rounded independently, the deviation $(A(x-y))_i$ in constraint i is a sum of independent random variables. Thus it is highly concentrated around its mean, which by choice of the probabilities is zero. Large deviation bounds like the Chernoff inequality allow to quantify such violations. Derandomizations transform this randomized approach into a deterministic algorithm (see [Rag88, SS96]).

Another well-known rounding result is due to Beck and Fiala [BF81]. They give a polynomial time algorithm computing a rounding y such that $\|A(x-y)\|_\infty < \|A\|_1$, where $\|A\|_1 = \max_{j\in[n]} \sum_{i=1}^m |a_{ij}|$. This result is particularly useful for sparse matrices. A one-sided version was proven by Karp et al. [KLR$^+$87] and applied to a global routing problem.

1.2 Hard Constraints

The notion of linear discrepancy prices all violations of constraints the same. This is feasible if all constraints are of the same kind. There are, however, a number of problems where this is definitely not the case. We sketch a simple one that carries most of the typical structure we are interested in.

Raghavan and Thompson [RT87] investigate the following routing problem. Given an undirected graph and several source–sink pairs (s_i, t_i), we are looking for paths f_i from s_i to t_i such that the maximum edge congestion is minimized. Solving the non-integral relaxation and applying path stripping (cf. [GKR$^+$99]), we end up with this rounding problem: Round the solution $(x_P)_P$ of the linear system

$$\text{Minimize } W \text{ s. t.} \quad \sum_{P \ni e} x_P \le W, \ \forall e$$
$$\sum_{P \in \mathcal{P}_i} x_P = 1, \ \forall i$$
$$x_P \ge 0, \ \forall P$$

to an integer one such that the first set of constraints is violated not too much and the second one is satisfied without any violation.

The first group of constraints ensures that W is the maximum congestion of an edge. Here a rounding error just enlarges the congestion (our objective value). The second kind of constraints is different. It ensures that each request is satisfied exactly once. Here no violation can be tolerated — it would result in demands satisfied more than once or not at all.

Further examples of rounding problems with hard constraints include other routing applications ([RT91, Sri01]), many flow problems ([RT87, RT91, GKR$^+$99]), partial and capacitated covering problems ([GKPS02]), the assignment problem with extra constraints ([AFK02]) and the linear discrepancy problem for hypergraphs in more than two colors ([DS03]).

1.3 Prior Work

For linear programs with right hand side of the hard constraints equal to one and hard constraints depending on disjoint sets of variables, Raghavan and Thompson [RT87] presented an easy solution. In the example above, for each i they pick one $P \in \mathcal{P}_i$ with probability x_P and set $y_P = 1$ and $y_{P'} = 0$ for all $P' \in \mathcal{P}_i \setminus \{P\}$.

The general case of the *integer splittable flow problem*, however, seems to require a more complicated random experiment. In the integer splittable flow problem, each source–sink pair has associated an integral demand d_i and the task is to find an integer flow f_i from s_i to t_i having value d_i. Using the approach sketched in the previous subsection, we would end up with the same rounding problem with the 1 replaced by d_i in the second set of constraints. Note that for this rounding problem, the ideas of Raghavan and Thompson (and all promising looking simple extensions) fail. Guruswami et al. [GKR$^+$99] state on the integral splittable flow problem (ISF) in comparison to the unsplittable flow problem that "standard roundings techniques are not as easily applied to ISF".

On FOCS 2001, Srinivasan [Sri01] presented a way to compute randomized roundings that respect the constraint that the sum of all variables remains unchanged (cardinality constraint) and fulfill some negative correlation properties (that imply Chernoff bounds). Among other results, this yields a randomized algorithm for the integer splittable flow problem.

The deterministic "pipage rounding" algorithm of Ageev and Sviridenko [AS] allows to round edge weights of in a bipartite graph in such a way that the sum of weights incident with a vertex changes by less than one ("degree preservation"). This yields improved approximation algorithms for maximum coverage problems and max-cut problems with given sizes of parts. Ageev and Sviridenko note that

their ideas could be used in a randomized way, but "the resulting algorithm will be too sophisticated to admit derandomization".

The ideas of [AS] and [Sri01] were combined in Gandhi, Khuller, Parthasarathy and Srinivasan [GKPS02] to obtain randomized roundings of edge weights in bipartite graphs that are degree preserving and fulfill negative correlation properties on sets of edges incident with a common vertex. This again yields improved randomized approximation algorithms for several problems as well as some nice per-user fairness properties.

1.4 Our Contribution

As can be seen from the previous subsection, there is now a decent amount of knowledge on rounding problems with hard constraints. However, most of these results focus rather on a particular application than on the common theme of respecting hard constraints. While still having an eye on the application, the main aim of this paper is to investigate rounding problems with hard constraints in a unified way.

To this end, we introduce the corresponding linear discrepancy notion and extend previous rounding results to deal with hard constraints. Though we find examples showing that the linear discrepancy can increase unexpectedly by adding hard constraints (Theorem 8), our algorithmic results show that reasonable hard constraints can be added without seriously worsening the optima.

We show that for constraints on disjoint sets of variables, a rounding error of $2\|A\|_1$ can be achieved, which is twice the bound of Beck and Fiala. For constraints of type $By = Bx$, where B is an arbitrary totally unimodular $m \times n$ matrix, we have a bound of $(1 + m)\|A\|_1$.

We provide a way to generate randomized roundings that satisfy hard constraints as in [Sri01]. They satisfy the key properties of the ones given there (hence our roundings yield all his results as well), but seem to be conceptually much simpler. This allows to derandomize them with respect to large deviation results. Our approach can be extended to the setting of [GKPS02], but we will not discuss this here.

We have to defer detailed descriptions to the remainder of the paper. In simple words though, our results show that many known rounding results (in particular, randomized rounding and its derandomizations) still work when suitable hard constraints are added. For reasons of space, many proofs are omitted in the paper.

2 Definitions and Notation

For a number r write $[r] := \{n \in \mathbb{N} \mid n \leq r\}$. For a matrix $A \in \mathbb{R}^{m \times n}$ let $\|A\|_1 := \max_{j \in [n]} \sum_{i \in [m]} |a_{ij}|$ denote the operator norm induced by the L_1 norm. For matrices A and vectors x we write $A_{|I \times J}$ and $x_{|J}$ to denote the restrictions (submatrices or subvectors) on the index sets $I \times J$ and J respectively.

Throughout the paper let $A \in \mathbb{R}^{m_A \times n}$, $B \in \mathbb{R}^{m_B \times n}$ and $x \in [0, 1]^n$ such that $Bx \in \mathbb{Z}^{m_B}$. We call the problem to find a $y \in \{0, 1\}^n$ such that $Bx = By$ and $\|A(x - y)\|_\infty$ is small a *rounding problem with hard constraints.*

Definition 2 (Linear Discrepancy with Hard Constraints). *Let* $A \in \mathbb{R}^{m_A \times n}$, $B \in \mathbb{R}^{m_B \times n}$ *and* $x \in [0,1]^n$ *such that* $Bx \in \mathbb{Z}^{m_B}$. *Put* $E(B,x) = \{y \in \{0,1\}^n \mid Bx = By\}$. *Then*

$$\operatorname{lindisc}(A,B,x) := \min_{y \in E(B,x)} \|A(x-y)\|_\infty,$$
$$\operatorname{lindisc}(A,B) := \max_{\substack{x \in [0,1]^n \\ Bx \in \mathbb{Z}^{m_B}}} \operatorname{lindisc}(A,B,x).$$

If $E(B,x) = \emptyset$, we have $\operatorname{lindisc}(A,B,x) = \infty$. Of course, the interesting case for our problem is that $E(B,x)$ is not empty. Therefore, we will assume that B is totally unimodular. This is justified by the following corollary of the theorems of Hoffman and Kruskal [HK56] and Ghouila-Houri [GH62].

Theorem 1. *The following properties are equivalent:*

(i) B *is totally unimodular.*
(ii) For all $x \in \mathbb{R}^n$ *there is a* $y \in \mathbb{Z}^n$ *such that* $\|x-y\|_\infty < 1$ *and* $\|B(x-y)\|_\infty < 1$.

3 Sparse Matrices

In this section, we extend the theorem of Beck and Fiala (cf. Section 1.1) to include hard constraints.

Theorem 2. *Let* B *be totally unimodular. Then*

a) $\operatorname{lindisc}(A,B) < (1+m_B)\|A\|_1$.
b) If $\|B\|_1 = 1$, *then* $\operatorname{lindisc}(A,B) < 2\|A\|_1$ *independent of* m_B.

Proof (Theorem 2). Set $\Delta := \|A\|_1$. Set $y = x$. Successively we will round y to a 0, 1 vector. Let $\delta > 0$ to be determined later. We repeat the following rounding process:

Put $J := \{j \in [n] \mid y_j \notin \{0,1\}\}$, and call these columns floating (the others fixed). Set $I_A := \{i \in [m_A] \mid \sum_{j \in J} |a_{ij}| > \delta\}$ and $I_B := \{i \in [m_B] \mid \sum_{j \in J} |b_{ij}| > 0\}$, and call these rows active (the others ignored). We will ensure that during the rounding process the following conditions are fulfilled (this is clear for the start, because $y = x$):

(i) $(A(x-y))_{|I_A} = 0$,
(ii) $(B(x-y))_{|I_B} = 0$,
(iii) $y \in [0,1]^n$.

If there is no floating column, that is, $J = \emptyset$, then our rounding process terminates with $y \in \{0,1\}^n$. Hence assume that there are still floating columns. We consider the system of equations

$$A_{|I_A \times J}\ z_{|J} = 0,\quad B_{|I_B \times J}\ z_{|J} = 0,\quad z_{|[n]\setminus J} = 0. \tag{1}$$

We have $|J|\,\Delta \geq \sum_{j\in J}\sum_{i\in I_A}|a_{ij}| = \sum_{i\in I_A}\sum_{j\in J}|a_{ij}| > |I_A|\,\delta$, hence $|J| > |I_A|\delta/\Delta$.

Case 1: $I_A \neq \emptyset$. The system (1) consists of at most $|I_A| + |I_B| + (n - |J|)$ equations. We will determine δ later in such a way the system (1) is under-determined. Then it has a non-trivial solution z.

By definition of J and (iii), there is a $\lambda > 0$ such that at least one component of $y + \lambda z$ becomes fixed and still $y \in [0,1]^n$. Note that $y + \lambda z$ instead of y also fulfills (i) and (ii). Set $y := y + \lambda z$. Since (i) to (iii) are fulfilled for this new y and also no previously fixed y_j becomes floating again (due to (iii)), we can continue this rounding process until all $y_j \in \{0,1\}$.

Case 2: $I_A = \emptyset$. Since $B_{|I_B\times J}\ x_{|J}$ is integral and B (and thus $B_{|I_B\times J}$) is totally unimodular, there is a $z \in \{0,1\}^J$ such that $B_{|I_B\times J}\ z = B_{|I_B\times J}\ x_{|J}$ (cf. e.g. Theorem 1). Define $\tilde{y} \in \{0,1\}^n$ by $\tilde{y}_j = z_j$ for $j \in J$ and $\tilde{y}_j = y_j$ else. Note that this implies $B(x - \tilde{y}) = 0$. Since $\tilde{y} \in \{0,1\}^n$ we end the rounding process with result $\tilde{y}$.

We show $\|A(x - y)\|_\infty < \delta$ for the resulting y. Let $i \in [m_A]$. Denote by $y^{(0)}$ and $J^{(0)}$ the values of y and J when the row i first became ignored. We have $y_j^{(0)} = y_j$ for all $j \notin J^{(0)}$ and $|y_j^{(0)} - y_j| < 1$ for all $j \in J^{(0)}$. Note that $\sum_{j\in J^{(0)}}|a_{ij}| \leq \delta$, since i is ignored. Thus

$$|(A(x-y))_i| = |(A(x-y^{(0)}))_i + (A(y^{(0)}-y))_i| = |0 + \sum_{j\in J^{(0)}} a_{ij}(y_j^{(0)} - y_j)| < \delta.$$

It remains to determine δ in such a way that the linear systems regarded are under-determined.

Part a) For the general case, put $\delta = (1 + m_B)\Delta$. Since $I_A \neq \emptyset$ in Case 1, $|I_B| \leq m_B$ and $|J| > |I_A|\delta/\Delta$, we have

$$|I_A| + |I_B| + (n - |J|) < |I_A| + |I_B| + n - |I_A|(1 + m_B) \leq n.$$

Part b) Assume now that $\|B\|_1 = 1$, that is, the constraints encoded in B belong to disjoint sets of variables. Then $|J| \geq 2|I_B|$ holds throughout the rounding process: If a constraint from B is active, it depends on at least two variables not yet fixed — simply because $B_{|I_B\times J}\ y_{|J} = B_{|I_B\times J}\ x_{|J}$ is integral and $B \in \{-1,0,1\}^{m_B\times n}$. Therefore, $\delta = 2\Delta$ suffices. We then have $|I_A| + |I_B| + (n - |J|) \leq |I_A| + n - \frac{1}{2}|J| < n$. □

The dependence on m_B in Part a) is of the right order, as the first example in Section 5 shows. In particular, a bound like $\mathrm{lindisc}(A, B, x) \leq (1 + \|B\|_1)\|A\|_1$ as could be conjectured from a) and b), does not hold. Let us also remark that the rounding algorithms of Karp et al. [KLR+87] admits similar extensions. We omit the details.

4 Randomized Rounding

In this section, we modify the approach of randomized rounding to respect hard constraints. The particular problem is to design a random experiment that at

the same time respects the hard constraints and generates "independent looking"randomized roundings (satisfying Chernoff bounds for example). Our random experiment is different from the one in [Sri01], which enables us to derandomize it. However, it also satisfies the main properties (A1) to (A3) of his approach. To ease reading, we describe our result in its simplest version in the following section and sketch possible extensions in the second one.

4.1 Randomized Construction and Derandomization

In this section, we only treat the case that $B \in \{0,1\}^{m_B \times n}$ and $\|B\| = 1$. Hence, we only regard so-called cardinality constraints that contain disjoint sets of variables.

Randomized construction: Assume first that all x_j are in $\{0, \frac{1}{2}, 1\}$. Since $\sum_{j\in[n]} b_{ij}x_j \in \mathbb{Z}$ for all $i \in [m_B]$ by assumption, we conclude that all $E_i := \{j \in [n] \,|\, x_j = \frac{1}{2}, b_{ij} = 1\}$ have even cardinality. Now partitioning each E_i into pairs[1] (j_1, j_2) and independently flipping a coin to decide whether $(y_{j_1}, y_{j_2}) = (1, 0)$ or $(y_{j_1}, y_{j_2}) = (0, 1)$ solves the problem in a randomized way (variables x_j with j contained in no E_i can be rounded independently at random).

For x_j having finite binary expansion, we iterate this procedure digit by digit: If x has binary length ℓ, write $x = x' + 2^{-l+1}x''$ with $x'' \in \{0, \frac{1}{2}\}^n$ and $x' \in [0,1]^n$ having binary length $\ell - 1$. Compute y'' as rounding of x'' as above. Put $x := x' + 2^{-l+1}y''$. Note that x now has binary length $\ell - 1$. Repeat this procedure until a binary vector is obtained. For each x having finite binary expansion, this defines a probability distribution $D(B, x)$ on $\{0,1\}^n$.

Theorem 3. *Let $y = (y_1, \ldots, y_n)$ be a sample from $D(B, x)$. Then it holds:*

(A1) *y is a randomized rounding of x: For all $j \in [n]$, $\Pr(y_j = 1) = x_j$.*
(A2) *$D(B, x)$ is distributed on $E(B, x)$: $\Pr(By = Bx) = 1$.*
(A3) *For all $S \subseteq [n]$ and $b \in \{0,1\}$, $\Pr(\forall j \in S : y_j = b) \leq \prod_{j\in S} \Pr(y_j = b)$.*

Proof. (A1): Let $j \in [n]$. If $x_j \in \{0,1\}$, the claim is trivial. Let x_j therefore have binary length $\ell \geq 1$. Let $\tilde{x}_j$ be the outcome of the first random experiment (i.e., $\tilde{x}_j$ is a random variable having binary length at most $\ell - 1$). By induction,

$$\begin{aligned}\Pr(y_j = 1) &= \sum_{\varepsilon\in\{-1,1\}} \Pr(\tilde{x}_j = x_j + \varepsilon 2^{-\ell}) \Pr(y_j = 1 \,|\, \tilde{x}_j = x_j + \varepsilon 2^{-\ell}) \\ &= \sum_{\varepsilon\in\{-1,1\}} \tfrac{1}{2}(x_j + \varepsilon 2^{-\ell}) = x_j.\end{aligned}$$

(A2): By definition of $D(B, x)$, in each rounding step the sum of the values with index in E_i is unchanged for all $i \in [m_B]$. Hence $(By)_i = \sum_{j\in E_i} y_j = \sum_{j\in E_i} x_j = (Bx)_i$.

[1] As we will see, the particular choice of this partition is completely irrelevant. Assume therefore that we have fixed some deterministic way to choose it (e.g., greedily in the natural order of $[n]$).

(A3): Let $S \subseteq [n]$. We show the claim for $b = 1$. Again, if $x \in \{0,1\}^n$, there is nothing to show. Let x therefore have binary length $\ell \geq 1$. Let $\tilde{x}$ be the outcome of the first rounding step. This is a random variable, that is uniformly distributed on the set $R(x)$ of possible outcomes (which is determined by x and the way we choose the partition into pairs). Note that for each $z \in R(x)$, also $\bar{z} := 2x - z \in R(x)$. Note also that $\prod_{j\in S} z_j + \prod_{j\in S} \bar{z}_j \leq 2\prod_{j\in S} x_j$. Hence by induction

$$\Pr(\forall j \in S : y_j = 1) = \sum_{z\in R(x)} \Pr(\tilde{x} = z) \Pr((\forall j \in S : y_j = 1) \mid \tilde{x} = z)$$

$$= \frac{1}{|R(x)|} \sum_{z\in R(x)} \prod_{j\in S} z_j \leq \frac{1}{|R(x)|}(\tfrac{1}{2}|R(x)| \cdot 2 \prod_{j\in S} x_j) = \prod_{j\in S} x_j = \prod_{j\in S} \Pr(y_i = 1).$$

□

As shown in [PS97], (A3) implies the usual Chernoff-Hoeffding bounds on large deviations.

We build on the following theorem of Raghavan [Rag88], which is a derandomization of the (independent) randomized rounding technique.

Theorem 4 (Raghavan (1988)). *For any $A \in \{0,1\}^{m\times n}$ and $x \in [0,1]^n$ a $y \in \{0,1\}^n$ can be computed in $O(mn)$ time such that $\|A(x-y)\|_\infty \leq (e-1)\sqrt{s\ln(2m)}$, where $s = \max\{\|Ax\|_\infty, \ln(2m)\}$.*

Noting that the pairing trick in a single iteration allows us to write Ay in the form "matrix times vector of independent random variables", we prove the following result.

Theorem 5. *Let $A \in \{0,1\}^{m_A\times n}$ and $B \in \{0,1\}^{m_B\times n}$ such that $\|B\|_1 = 1$.*

a) Let $x \in [0,1]^n$ such that $Bx \in \mathbb{Z}^{m_B}$. Then for all $\ell \in \mathbb{N}$, a binary vector y such that $Bx = By$ and

$$\|A(x-y)\|_\infty \leq 52\sqrt{\max\{\|Ax\|_\infty, \ln(4m_A)\}\ln(4m_A)} + n2^{-\ell}$$

can be computed in time $O(mn\ell)$.

b) $\operatorname{lindisc}(A,B) \leq 5\sqrt{n\ln(4m_A)}$.

4.2 Extensions

(1) We always assumed that Bx is integral. A trivial reduction (by adding dummy variables) extends our results to arbitrary Bx. We then have:

(A2+) For all $i \in [m_B]$, $(By)_i$ is a randomized rounding of $(Bx)_i$.

In particular, $(By)_i \in \{\lfloor (Bx)_i \rfloor, \lceil (Bx)_i \rceil\}$ with probability one.

(2) Raghavan [Rag88] also obtains the bound $\|A(x - y)\|_\infty \leq e\ln(2m)/\ln(e\ln(2m)/\|Ax\|_\infty)$ for the case that $\|Ax\|_\infty \leq \ln(2m)$. This is strongest for constant $\|Ax\|_\infty$, where it yields a bound of $O(\frac{\log m}{\log\log m})$ instead of our bound of $O(\log m)$. Since the typical application of randomized rounding seems to be that $\|Ax\|_\infty$ is large, we do not try to improve our result in this direction.

(3) One subtle aspect in derandomizing Chernoff bounds lies in the computation of the involved pessimistic estimators. There is no problem if one works in a model that allows exact computations of real numbers. In the more realistic RAM model, things are more complicated. Raghavan's derandomization then only works for 0, 1 matrices A. Srivastav and Stangier [SS96] gave a solution that works for matrices having arbitrary entries in $[0,1]\cap\mathbb{Q}$, though has a higher time complexity of $O(mn^2\log(mn))$.

Here again the simplicity of our approach pays off. Since we only need to derandomize Chernoff type large deviation bounds, we can plug in any algorithmic version of the underlying large deviation inequality.

(4) If $B \in \{-1, 0, 1\}$, one can modify the definition of $\tilde{y}$ in the proof above in such a way that $B\tilde{y} = 0$. An extension to further values, however, is not possible as we might run into the problem that no integral solution exists at all. For example, the single constraint $\sum_{i\in[3]} \frac{4}{5}x_i = 2$ is satisfied by $x_i = \frac{5}{6}$, but clearly no 0, 1 solution exists.

(5) The constant of 52 is not the full truth. Things become much better, if $\|Ax\|_\infty \gg \ln(4m_A)$. In this case, the constant reduces to less than 6.

4.3 Applications

In this subsection, we sketch two applications. Note that — and this is one advantage of the results presented above — our results in simple words just state that randomized rounding and the corresponding derandomizations work as before even if a few hard constraints are added to the problem. This seems to be particularly useful for real-world application, which usually lack the plainness of problems regarded in theoretical sciences.

We start with derandomizing Srinivasan's [Sri01] solution for the integral splittable flow problem (cf. Subsection 1.2 and 1.3). Note that for most of the other randomized results in [Sri01], deterministic algorithms of same quality have already been given earlier by Ageev and Sviridenko [AS].

The integral splittable flow problem extends the unit flow version of Raghavan and Thompson [RT87]. From the problem formulation, it is clear that Theorem 3 and 5 can be applied: The hard constraints depend on disjoint sets of variables, namely the paths obtained from applying the path stripping procedure to the flow satisfying a particular demand. Analogous to the result of Raghavan and Thompson for unit flows and derandomizing Srinivasan [Sri01] (with larger constants), we obtain the following.

Theorem 6. *A solution of the relaxation with objective value $W \geq \ln(4|E|)$ can efficiently be transformed into an integer solution with objective value $W + 52\sqrt{W \ln(4m_A)}$.*

As a second example, let us consider the packing problem

$$\max c^t x \quad \text{such that} \quad Ax \leq k, x \in \{0,1\}^n.$$

We may view this as a scheduling problem. We want to select a set of jobs maximizing our profit in such a way that all m machines are busy for at most k time units. Using an additional scaling trick, Raghavan [Rag88] showed that for $k = \Omega(\log m_A)$, approximations with additive error exist.

In a real world scenario, additional constraints often are present (or show up while a first solution is analyzed). Here, one may assume that different parties have a particular interest in some jobs to be scheduled. In this case, we have disjoint sets $F_1, \ldots, F_\ell$ of jobs favored by party $i \in [\ell]$, and a fairness condition might impose that from each set F_i, at least a given number of r jobs has to be scheduled.

Note that r can (and usually will) be small compared to k. Hence large deviation bounds will not be applicable. However, the following easily solves the problem: (i) Solve the relaxation with additional constraints $\sum_{j \in F_i} x_j \geq r$, $i \in [\ell]$. Denote the solution by $\hat{x}$. (ii) Apply randomized rounding or its derandomization on $\hat{x}$ with the additional hard constraints that $\sum_{j \in F_i} y_j$ is a randomized rounding of $\sum_{j \in F_i} \hat{x}_j$ for all $i \in [\ell]$ (cf. the extensions subsection for a remark on these dependencies). We thus obtain an integer solution of similar quality as Raghavan's that also satisfies our fairness requirements.

4.4 Comparison to the Approach of Srinivasan

In Srinivasan [Sri01], randomized roundings satisfying hard constraints as in Theorem 3 were generated. His approach is to repeat regarding two variables only, and fixing one to an integer value and propagating the other with an updated probability distribution.

This sequential rounding approach seems to be much harder to work with. We currently do not see how this algorithm can be derandomized. Also, we feel that proving the properties (A1) to (A3) must be quite complicated (proofs are omitted in [Sri01]).

Note that the complexity of both approaches is very similar. Working with real numbers in [Sri01] hides part of complexity that is present in the bit-wise model used in this paper.

5 Examples and Lower Bounds

The following simple example shows that hard constraints may increase the rounding error significantly. It also shows that the dependence on m_B in part a) of Theorem 2 is of the right order.

Example 1: Let n be a multiple of 4. Let $A = (1\,0\,1\,0\ldots) \in \mathbb{R}^{1\times n}$, $m_B = n-1$ and $B \in \{0,1\}^{m_B \times n}$ such that $b_{ij} = 1$ if and only if $j \in \{i, i+1\}$. Let $x = \frac{1}{2}\mathbf{1}_n$. Then

$$\begin{aligned} \operatorname{lindisc}(A, x) &= 0, \\ \operatorname{lindisc}(A, x') &\le \tfrac{1}{2} \quad \text{for all } x' \in [0,1]^n, \\ \operatorname{lindisc}(A, B, x) &= \tfrac{1}{4}n \quad (= \tfrac{1}{4}(1+m_B)\|A\|_1). \end{aligned}$$

Example 2: The linear discrepancy problem for hypergraphs is to compute for a given mixed coloring (each vertex receives a weighted mixture of colors) a pure coloring in such a way that each hyperedges in total contains (roughly) the same amount of each color with respect to both colorings.

Definition 3 (Linear Discrepancy Problem for Hypergraphs). *Let $c \in \mathbb{N}_{\ge 2}$. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph. A mapping $p : V \to [0,1]^c$ such that $\sum_{d\in[c]} p(v)_d = 1$ for all $v \in V$ is called* mixed coloring of $\mathcal{H}$. *It is called* pure coloring, *if for all $v \in V$ there is a (unique) $d \in [c]$ such that $p(v)_d = 1$. In this case, we say that v has color d and write $\hat{p}(v) = d$.*

The discrepancy of two mixed colorings p, q is $\operatorname{disc}(\mathcal{H}, p, q) = \max_{d\in[c]} \max_{E\in\mathcal{E}} \left|\sum_{v\in E} p(v)_d - \sum_{v\in E} q(v)_d\right|$. *The objective in the linear discrepancy problem for hypergraphs is to find for given hypergraph $\mathcal{H}$ and mixed coloring p a pure one q such that* $\operatorname{disc}(\mathcal{H}, p, q)$ *is small. Put* $\operatorname{lindisc}(\mathcal{H}, c) := \max_p \min_q \operatorname{disc}(\mathcal{H}, p, q)$.

A hypergraph is called *totally unimodular*, if its incidence matrix is totally unimodular. It is well known that totally unimodular hypergraph behave nicely in linear discrepancy problems.

Theorem 7. *Let $\mathcal{H} = (V, \mathcal{E})$ be a totally unimodular hypergraph.*

a) **De Werra [dW71]:** *For all numbers c of colors, the combinatorial discrepancy* $\operatorname{lindisc}(\mathcal{H}, \frac{1}{c}\mathbf{1}_V)$ *is less than* 1.
b) **Hoffman, Kruskal [HK56]:** *The linear discrepancy* $\operatorname{lindisc}(\mathcal{H}, 2)$ *of $\mathcal{H}$ in* 2 *colors is less than* 1.

The constant in b) was recently [Doe01] improved to the sharp bound of $|V|/(|V|+1)$. Contrary to what one might expect, a combination of a) and b) is not true:

Theorem 8. *For all $c \ge 3$ there is a totally unimodular hypergraph $\mathcal{H}$ such that* $\operatorname{lindisc}(\mathcal{H}, c) \ge \ln(c+1) - 1$. *In consequence, the bound* $\operatorname{lindisc}(\mathcal{H}, c) < 1$ *for totally unimodular hypergraphs holds only in the case $c = 2$.*

References

[AFK02] S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Math. Program.*, 92:1–36, 2002.

[AS] A. Ageev and M. Sviridenko. Pipage rounding: a new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*. To appear. Also available from the authors' homepages.

[BF81] J. Beck and T. Fiala. "Integer making" theorems. *Discrete Applied Mathematics*, 3:1–8, 1981.

[Doe01] B. Doerr. Lattice approximation and linear discrepancy of totally unimodular matrices. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 119–125, 2001.

[DS03] B. Doerr and A. Srivastav. Multicolour discrepancies. *Combinatorics, Probability and Computing*, 12:365–399, 2003.

[dW71] D. de Werra. Equitable colorations of graphs. *Rev. Française Informat. Recherche Opérationnelle*, 5(Ser. R-3):3–8, 1971.

[GH62] A. Ghouila-Houri. Caractérisation des matrices totalement unimodulaires. *C. R. Acad. Sci. Paris*, 254:1192–1194, 1962.

[GKPS02] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding in bipartite graphs. In *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 323–332, 2002.

[GKR+99] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 19–28, New York, 1999. ACM.

[HK56] A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 223–246. 1956.

[KLR+87] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica*, 2:113–129, 1987.

[PS97] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM J. Comput.*, 26:350–368, 1997.

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *J. Comput. Syst. Sci.*, 37:130–143, 1988.

[RT87] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[RT91] P. Raghavan and C. D. Thompson. Multiterminal global routing: a deterministic approximation scheme. *Algorithmica*, 6:73–82, 1991.

[Spe87] J. Spencer. *Ten Lectures on the Probabilistic Method*. SIAM, 1987.

[Sri01] A. Srinivasan. Distributions on level-sets with applications to approximations algorithms. In *Proc. 41th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 588–597, 2001.

[SS96] A. Srivastav and P. Stangier. Algorithmic Chernoff-Hoeffding inequalities in integer programming. *Random Structures & Algorithms*, 8:27–58, 1996.

Sorting Stably, In-Place, with $O(n \log n)$ Comparisons and $O(n)$ Moves

Gianni Franceschini*

Dipartimento di Informatica, Università di Pisa,
Largo Bruno Pontecorvo 3, 56127 Pisa, Italy
francesc@di.unipi.it

Abstract. We settle a long-standing open question, namely whether it is possible to sort a sequence of n elements stably (i.e. preserving the original relative order of the equal elements), using $O(1)$ auxiliary space and performing $O(n \log n)$ comparisons and $O(n)$ data moves. Munro and Raman stated this problem in [J. Algorithms, 13, 1992] and gave an in-place but unstable sorting algorithm that performs $O(n)$ data moves and $O(n^{1+\epsilon})$ comparisons. Subsequently [Algorithmica, 16, 1996] they presented a stable algorithm with these same bounds. Recently, Franceschini and Geffert [FOCS 2003] presented an *unstable* sorting algorithm that matches the asymptotic lower bounds on all computational resources.

1 Introduction

In the comparison model the only operations allowed on the totally ordered domain of the input elements are the comparison of two elements and the transfer of an element from a cell of memory to another. Therefore, in this model it is natural to measure the efficiency of an algorithm with three metrics: the number of comparisons it requires, the number of element moves it performs and the number of auxiliary memory cells it uses, besides the ones strictly necessary for the input elements. It is well known that in order to sort a sequence of n elements, at least $n \log n - n \log e$ comparisons have to be performed in the worst case. Munro and Raman [13] set to $\lfloor 3/2n \rfloor$ the lower bound for the number of moves. In the general case of input sequences with repeated elements, an important requirement for a sorting algorithm is to be *stable*: the relative order of equal elements in the final sorted sequence is the same found in the original one.

The sorting problem is fundamental in computer science and has been widely studied from the very beginning. The Heapsort [16] is the first space-optimal sorting algorithm performing $O(n \log n)$ comparisons in the worst case. However, this algorithm is unstable and the number of element moves performed in the worst case is $O(n \log n)$. The existence of a sorting algorithm that is stable, comparison and space optimal was proven by Pardo [14]. Concerning the partition-

* Work supported in part by the Italian Ministry of Research and Education. Part of the work done while visiting EECS Department, Berkeley.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 629–640, 2005.

based approach, the ordinary Quicksort [3] is only space optimal. An unstable, comparison and space optimal sorting can be derived using an in-place selection algorithm like [9]. Katajainen and Pasanen [6] presented an unstable sorting requiring $o(n \log n)$ moves in the worst case while guaranteeing in-placeness and $O(n \log n)$ comparisons. That algorithm performs $O(n \log n / \log \log n)$ data moves in the worst case.

Concerning the sorting algorithms with optimal number of data moves, the classical selection sort operates in-place, and performs $O(n)$ moves in the worst case but it is not stable and performs $O(n^2)$ comparisons in the worst case. An improvement in the number of comparisons came from Munro and Raman [11] with a generalization of the heapsort performing $O(n^{1+\epsilon})$ comparisons in the worst case while maintaining the unstability. Finally, a stable algorithm with these same bounds was presented in [12].

If the space optimality is given up, the address-table sort [7] performs an optimal number of comparisons and moves but requiring $O(n)$ auxiliary cells of memory. It can be easily modified to achieve the stability. The space requirement has been reduced to $O(n^{\varepsilon})$ by a variant of samplesort [11].

Recently, Franceschini and Geffert [1] presented an *unstable* sorting algorithm that matches the asymptotic lower bounds on all the computational resources, space, comparisons and data moves.

Our Result. In this paper we settle a long-standing open question explicitly stated by Munro and Raman in [11], namely whether it is possible to sort a sequence of n elements stably, using $O(1)$ auxiliary space, performing $O(n \log n)$ comparisons and $O(n)$ data moves. So far, the best known algorithm for stable in-place sorting with $O(n)$ moves was the one presented by Munro and Raman in [12], performing $O(n^{1+\epsilon})$ comparisons in the worst case.

Two basic techniques are very common when space-efficiency of algorithms and data structures in the comparison model is the objective. The first one is the *bit stealing* [10]: a bit of information is encoded in the relative order of a pair of distinct input elements. The second common technique is the *internal buffering* [8], in which some of the input elements are used as placeholders in order to simulate a working area and permute the other elements at less cost. It is easy to understand how disruptive the internal buffering is when the stability of the algorithm is an objective: if the placeholders are not distinct, the original order of identical placeholders can be lost using the simulated working area. As witness of the clash between stability and internal buffering technique, we can cite the difference in complexity between the first in-place linear time merging algorithm, due to Kronrod [8], and the first stable one by Pardo [14].

Our strategy for the development of a stable sorting algorithm matching the asymptotic lower bounds on all the computational resources can be synthesized in four major points. Let n and d be, respectively, the number of input elements and the number of distinct elements in the sequence to be sorted.

First (Section 2), we show how to extract from the input sequence $O(n/\log n)$ pairs of distinct elements to be used for encoding purposes with the basic technique of bit-stealing.

Second (Section 3), we show how to extract $b = O(min(d, n/\log^2 n))$ distinct elements from the sequence, stably and within our computational bounds.

Third (Section 4), we show how to deal with a sequence with $\Omega(n/\log^2 n)$ distinct elements. We divide our sorting problem in $O(\log^2 n)$ sub-problems of size $O(n/\log^2 n)$ and show how to solve those sub-problems assuming the availability of a sufficient number of distinct elements to be used as placeholders, that is in case $b = \Theta(n/\log^2 n)$. After that, we show how to fuse the $O(\log^2 n)$ sorted sub-sequences with a multi-way stable merging technique requiring a very limited amount of placeholders to deliver a sorted sequence of $O(n)$ elements.

Fourth (Section 5), we show how to deal with the hardest case, namely a sequence with $b = o(n/\log^2 n)$ distinct elements. First, we partition the sequence in three zones $C'YC''$ around a pivot element (the occurrences of the pivot will be in zone Y). Then we show how to group the identical elements laying in sub-zones of size $O(b \log^2 n)$ and how to acquire and encode by bit-stealing a linked structure traversing the groups in sorted order. Finally, we show how to use the groups and the encoded linked structure to permute first C' using YC'' as working zone and then C'' using $C'Y$ *without disrupting the (sorted) order of* C'. This is a major and crucial difference with other unstable sorting algorithms like [6] or [1] where the internal buffering process is iterated $O(\log n)$ times until the sub-problem becomes directly tractable without internal buffering.

2 Stealing Bits

With the *bit-stealing* technique [10] the value of a bit is encoded in the relative order of two distinct elements (e.g. the increasing order for 0). In this section we show how to collect $O(n/\log n)$ pairs of distinct elements, stably and within our computational bounds.

The *rank* of an element x_i belonging to a *sequence* $S = x_1 \dots x_t$ is the cardinality of the multiset $\{x_j \in S \mid x_j < x_i \text{ or } (x_j = x_i \text{ and } j \leq i)\}$. The *rank* of an element x in a *set* $\mathcal{S}$ is the rank of x in any sequence S of the elements in $\mathcal{S}$. Let $r = \lceil n/\log n \rceil$ and let π' and π'' be, respectively, the element with rank r and the element with rank $n - r + 1$ in the input sequence. We want to partition *stably* and in-place the input sequence in five zones $A'P'AP''A''$ such that, for each $a' \in A'$, $p' \in P'$, $a \in A$, $p'' \in P''$ and $a'' \in A''$, we have that $a' < p' = \pi' < a < p'' = \pi'' < a''$. That can be done in $O(n)$ comparisons and $O(n)$ moves using the stable in-place selection and the stable in-place partitioning of Katajainen and Pasanen [4, 5].

Zones A' and A'' can be sorted stably and in-place in $O(n)$ time simply using the stable in-place mergesort (e.g. [15]). If there are no elements in A, we are done since the input sequence is already sorted. Otherwise we are left with the unsorted subsequence A and with a set M of $r = O(n/\log n)$ pairs of distinct elements, that is $M = \{(Q'[1], Q''[1]), (Q'[2], Q''[2]), \dots, (Q'[r], Q''[r])\}$, where $Q' = A'P'$ and $Q'' = P''A''$.

The starting addresses of Q' and Q'' can be maintained in two locations of auxiliary memory (we can use $O(1)$ auxiliary locations) and so, for any i, we

can retrieve the addresses of the elements of the i-th pair in $O(1)$ operations. Therefore, we can view M as a *sequence of encoding words* of t bits each, for any t. We have to pay attention to the costs of using encoding bits or encoding words, though: *reading* an encoding word of t bits takes t comparisons, *changing* it costs t comparisons and $O(t)$ data moves in the worst case. Moreover, we could have chosen the ranks of π' and π'' as cr and $n - cr + 1$ for any constant c, so that the number of encoded bits would be cr without changing the asymptotical bounds of the algorithm. Let m be the size of A, we can make the following assumption:

Assumption 1. *We can use an* auxiliary encoding memory $\mathcal{M}$ *consisting of* $O(\frac{r}{\log m})$ *words of* $\lceil \log m \rceil$ encoding bits *each and with the following cost model. For any word* w*, for any* $q \leq \lceil \log m \rceil$ *and for any group* g *of* q *bits of* w*:*

- *retrieving the values of all the bits in* g *requires* q *comparisons;*
- *changing the values of all the bits in* g *requires* $\Theta(q)$ *moves.*

If we are able to solve the following problem over the sequence A, we are able to solve the original problem.

Problem 1. Under Assumption 1, sort the sequence A of m ($< n$) elements stably, using $O(1)$ locations of auxiliary memory, performing $O(m \log m)$ comparisons and $O(m)$ data moves.

In the following sections, we will use the auxiliary encoding memory $\mathcal{M}$ as normal auxiliary memory (of course each use of $\mathcal{M}$ must be correctly accounted in the complexity analysis). We will declare explicitly any new auxiliary data (indices, pointers...) stored in $\mathcal{M}$.

3 Extracting a Set of Distinct Elements

Let d be the number of distinct elements in A. In this section we show how to go from A to CB such that:

Property 1.

(a) B contains $b = min(d, \lceil r/\log m \rceil)$ elements,
(b) each $x \in B$ is the rightmost occurrence of its kind in A (and therefore the elements in B are distinct),
(c) C is the sequence that can be obtained from A simply removing the elements in B.

The elements in B will be used in Sections 4 and 5 as in the technique of *internal buffering* ([8]). Basically, some of the elements are used as placeholders to simulate a working area in which the other elements can be permuted efficiently. If the placeholders are not distinct, the stability of the algorithm become a difficult task since the original order of identical placeholders can be lost using the simulated working area. The elements in B are distinct so we do not have to worry, as long as we can sort $O(m)$ elements with $o(m)$ placeholders (Section 4). However, as we will see in Section 5, if $|B|$ is too small, we have to deal with bigger buffers whose internal order has to be preserved *entirely*, not only the relative order of equal elements.

Main Cycle of the Buffer Extraction. We first present the main cycle of the algorithm for the creation of B. Before we start, let us recall a basic technique for space-efficient block exchange. From a block $X = x_1 \dots x_t$ of t consecutive elements we can obtain the reverse $X^R = x_t \dots x_1$ in linear time and in-place simply exchanging x_1 with x_t, x_2 with x_{t-1} and so forth. Two consecutive blocks X and Y, possibly of different sizes, can be exchanged in-place and linear time with three block reversals, since $YX = (X^R Y^R)^R$.

We will denote the evolving sequence using the identifier of the original sequence A until the end of the algorithm. There are three phases.

First Phase. We maintain two indices i and j initially set, respectively, to 1 and m. Moreover, we allocate an array I of $\lceil r/\log m \rceil$ words in $\mathcal{M}$. The following steps are repeated until $i > \lceil r/\log m \rceil$ or $j < i$:

1. SEARCH$(A[j], A[1 \dots i-1])$.
2. If $A[j]$ is not in $A[1 \dots i-1]$, exchange $A[j]$ and $A[i]$, PROCESS$(A[1 \dots i])$, set $I[i] = j$ and increase by one i.
3. Decrease by one j.

At the end of this first phase, we have collected the $b = min(d, \lceil r/\log m \rceil)$ buffer elements in $A[1 \dots b]$. This set of elements respects point (b) and, obviously, point (a) in Property 1. At the end of this section we will explain how the procedures SEARCH and PROCESS can be defined in order to stay within the target bounds.

Second Phase. We have to collect all the b elements in $A[b+1 \dots m]$ that have been exchanged in the execution of step 2 in the first phase. We know where they are because, for each exchange, we stored the value of j in the array I. Let $s_1, \dots, s_b$ be the elements we have to collect, indexed from the leftmost to the rightmost in $A[b+1 \dots m]$. The position of s_i is stored in $I[|I| - i + 1]$. We have to extract them maintaining their relative order and the relative order of the other $m - 2b$ elements in $A[b+1 \dots m]$. Let $A_1, \dots, A_b$ be the sub-sequences of $A[b+1 \dots m]$ that separate $s_1, \dots, s_b$, that is $A[b+1 \dots m] = A_1 s_1 A_2 s_2, \dots A_b s_b$ (some of them can be void). We collect $s_1, \dots, s_b$ in a sub-sequence S starting from the position of s_1. During the process, S slides toward the right end of $A[b+1 \dots m]$. Let Z be an array of $b-1$ words stored in $\mathcal{M}$. Let $S = s_1$ and $i = 2$. The following steps are repeated until $i > b$:

1. If $|S| \leq I[|I| - i + 1] - I[|I| - i] - 1$, do a block exchange between the two adjacent blocks S and A_i ($I[|I| - i + 1] - I[|I| - i] - 1$ is the size of A_i) and set $Z[i-1] = 0$.
2. Otherwise, let $S = S'S''$ with $|S'| = I[|I| - i + 1] - I[|I| - i] - 1$, exchange S' with A_i (obvious exchange of two non adjacent but equal sized blocks) and set $Z[i-1] = |S'|$. After that $S = S''S'$.
3. In both cases, now S is adjacent to s_i; let $S = Ss_i$ and increase i by one.

Third Phase. At the end of the first two phases we have that $B = A[1 \dots b]$ precedes $C = A_1 A_2, \dots A_{b-1} A_b S$. Moreover, the order of the elements $A_1, \dots, A_b$ has been preserved. This may not be true for the elements in S because of step 2 of the second phase. Therefore, C does not respect point (c) in Property 1.

First, we have to recover the order of the elements in S holding within the first and the second phase. Basically, this can be done in-place, using the information stored in Z to "un-roll" all the rotations (from $S = S'S''$ to $S = S''S'$) executed with step 2 of the second phase. Let $i = b$. The following steps are repeated until $i = 1$:

1. If $Z[i-1] > 0$, exchange the blocks $S[1 \dots i - Z[i-1] - 1]$ and $S[i - Z[i - 1] \dots i - 1]$.
2. Decrease i by one.

Finally, we reverse S, recovering the original order holding before the first phase, and we exchange S with B. After that the sequence CB respects all the points in Property 1.

Lemma 1. *Under Assumption 1, the buffer extraction algorithm operates in-place, Property 1 holds for the sequence CB, and the comparisons and moves performed are, respectively, $O(mX_s + X_p + m)$ and $O(Y_p + m)$, where*

- *X_s upper bounds the number of comparisons of each invocation of* Search *in step 1,*
- *X_p and Y_p are, respectively, the total number of comparisons and moves performed by the b invocations of* Process *in step 2.*

Managing a Growing Set of Distinct Elements Compactly. We are left with the task of performing efficiently the operations Search and Process in steps 1 and 2 of the first phase of the buffer extraction algorithm. We have to solve the following problem.

Problem 2. Under Assumption 1, we have to handle the growth of a set $\mathcal{B}$ of at most $\lceil r/\log m \rceil = O(m/log^2 m)$ distinct elements so that the following properties hold:

- At any time, $\mathcal{B}$ is stored in $|\mathcal{B}|$ contiguous memory locations.
- At any time, $\mathcal{B}$ can be searched with $O(\log m)$ comparisons.
- When $\mathcal{B}$ is complete, the total number of comparisons and moves performed is $O(|\mathcal{B}| \log^2 m)$.

This subproblem can be easily solved with any *implicit dictionary* supporting search and insertion operations respectively in $O(\log |\mathcal{B}|)$ and $O(\log^2 |\mathcal{B}|)$ time, where the bound for insertion can also be in amortized sense. For this purpose we can use the implicit dictionary in [2] (as a matter of fact, such structure is excessively powerful for Problem 2; in the full paper we will give a self-contained, ad-hoc solution). Therefore, by Lemma 1, we have that:

Theorem 1. *Under Assumption 1, a sequence CB having Property 1 can be obtained from A, stably, using $O(1)$ cells of auxiliary memory, performing $O(m)$ moves and $O(m \log m)$ comparisons in the worst case.*

4 Sorting with Many Distinct Elements

In this section we show how to sort a sequence CB satisfying Property 1 and with $b = |B| = \lceil r/\log m \rceil$. The two-level technique used in [1] can be easily adapted to sort stably b consecutive elements using the b buffer elements in B, the stolen bits in $\mathcal{M}$ and $O(1)$ auxiliary space. We have that:

Lemma 2. *Under Assumption 1, b elements can be sorted stably, using $O(1)$ auxiliary space and another set of b distinct elements as placeholders, with $O(b)$ moves and $O(b \log m)$ comparisons.*

Assuming that, we are able to show how to sort CB using a multi-way stable merging technique requiring a very limited amount of placeholders.

We want to solve the following problem.

Problem 3. We have $s \leq \frac{\log m}{\log \log m}$ sorted sequences $E_1, \ldots, E_s$ of $k \leq \frac{m}{s}$ elements each and a set $\mathcal{U}$ of $s(\lceil \log m \rceil)^2$ distinct elements. Under the Assumption 1, we want to sort the sk elements, stably, using $O(1)$ auxiliary locations, with $O(sk \log m)$ comparisons and $O(sk)$ moves.

Each sorted sequence is divided into $\gamma = k/(\lceil \log m \rceil)^2$ *fragments* of $(\lceil \log m \rceil)^2$ contiguous elements each (for simplicity, let us suppose $(\lceil \log m \rceil)^2$ divides k). Starting from the fragment with the largest element, we will denote the j-th fragment of the sequence E_i with F_i^j. The fragments of E_i are linked in a bidirectional list following the reverse sorted order of E_i. The fragment with the largest element of a sequence is the *head* of the list. For each list we need $2k/(\lceil \log m \rceil)^2$ words of $\lceil \log m \rceil$ bits to store the pointers; for that, we use $\mathcal{M}$ in the usual way. One of the basic events in the process we are about to describe is the exchange of fragments (possibly belonging to two different sorted sequences). From now on we will assume that, when a fragment is moved, the pointers of its successor and its predecessor (if any) in its linked list are updated.

Let us denote the whole sequence of elements with P and with P_i the i-th fragment of P from the left end, for $i = 1, \ldots, s$. The initial configuration is $P = E_1 E_2 \ldots E_{s-1} E_s U$, where $E_i = F_i^\gamma F_i^{\gamma-1} \ldots F_i^2 F_i^1$ and U contains the set $\mathcal{U}$ in some order. First, we exchange F_1^1 with F_1^γ, F_2^1 with $F_1^{\gamma-1}$ and so forth until the s heads are the first s fragments of P ($P_1 = F_1^1, P_2 = F_2^1 \ldots$).

For $i = 1, \ldots, s$ the fragment P_i is associated with a small integer p_i of $O(\log \log m)$ bits with the index (in P_i) of the first (from the right end) element of P_i not in $\mathcal{U}$. Two more indices *num* and *last* are maintained: *num* stores the current number of merged elements and *last* stores the address of the rightmost (in the whole P) fragment. All the p_i are stored in $\mathcal{M}$ while *num* and *last* are in stored in *two normal cells of memory*. Initially all the small indices are set to $(\lceil \log m \rceil)^2$, *num* is set to 0 and *last* to $|P| - |U| - (\lceil \log m \rceil)^2 + 1$.

Then the merging phase begins.

1. The largest element among $P_1[p_1], P_2[p_2], \ldots, P_s[p_s]$ is found (for the stability, in case of equal elements the one in the fragment of the sorted sequence with the largest index is chosen). Let it be $P_i[p_i]$.

2. $P_i[p_i]$ is exchanged with $P[|P| - num]$, p_i is decreased by one and num is increased by one.
3. If P_i contains only elements of $\mathcal{U}$, that is $p_i = 0$, then let ν be the address of the next fragment of E_i. P_i is exchanged with the fragment starting at ν and then the fragment starting at ν is exchanged with the one starting at $last$. Finally, we set $last = last - (\lceil \log m \rceil)^2$ and repeat until $num = sk$.

After the execution $P = US$, where in U we have the elements of $\mathcal{U}$ in some order and in S we have the stably sorted sequence of the sk elements. The wanted bounds can be proved easily. We highlight a central part in the account of the number of moves: with each iteration, decreasing by one the selected p_i costs $O(\log \log m)$ moves *in the worst case* but only $O(1)$ moves *in amortized sense* leading to the wanted linear bound for the total number of moves.

With the fragmented multi-way merging technique we have just introduced, the sequence CB can be sorted in the following way (when $b = |B| = \lceil r/\log m \rceil$).

1. C is logically divided into $t = \lceil |C|/b \rceil$ sub-sequences $C_1 C_2 \ldots C_{t-1} C_t$ of b elements each. Every C_i is sorted using a variation of the two-level technique in [1] with B as internal buffer.
2. The resulting sequence $C'_1 C'_2 \ldots C'_{t-1} C'_t$ is sorted running a constant number of iterations of the multi-way mergesort using the fragmented multi-way merging with $s = \frac{\log m}{\log \log m}$.
3. B is sorted with the stable in-place mergesort ([15]) and C and B are merged in-place stably ([15]).

Therefore, by Lemma 2 and by the solution to Problem 3, we can conclude that:

Theorem 2. *Under Assumption 1, a sequence CB with Property 1 and $|B| = \lceil r/\log m \rceil$ can be sorted stably, using $O(1)$ auxiliary cells, performing $O(m \log m)$ comparisons and $O(m)$ moves in the worst case.*

5 Sorting with Few Distinct Elements

In this section we show how to sort a sequence CB satisfying Property 1 and with $b = |B| < \lceil r/\log m \rceil$, that is, when the number d of distinct elements in CB is less than $\lceil r/\log m \rceil$.

The first step consists in solving the following abstract problem.

Problem 4. We have two sequences V and G of $t \leq m$ elements each and a set $\mathcal{D}$ of $d' < \lceil r/\log m \rceil$ elements. V has $d'' \leq d'$ distinct elements. We are given an $O(1)$ time boolean function *belongs_to_V*(x) that, at any time, returns true if and only if x belongs to the set of elements originally contained in V.

We want to go from sequence VGD to sequence $V'GD'$ where V' contains the elements in V sorted stably and D, D' contain the elements in $\mathcal{D}$ in any order. Under Assumption 1, we have to use $O(1)$ auxiliary locations and perform $O(t \log m)$ comparisons and $O(t)$ moves.

Our solution to Problem 4 has three phases.

First Phase. V is logically divided into $\frac{|V|}{d'(\lceil \log m \rceil)^2}$ contiguous blocks $V_1 V_2 \ldots$ of $d'(\lceil \log m \rceil)^2$ elements each. We want to sort any block V_i, stably, using $O(1)$ auxiliary locations, $O(|V_i| \log m)$ comparisons and $O(|V_i|)$ moves. This can be accomplished the same way we sorted the sequence C in Section 4. First, each sub-block of d' contiguous elements of V_i is sorted using the d' elements of $\mathcal{D}$ as placeholders (as in Section 4).

Then, the $(\lceil \log m \rceil)^2$ sorted sub-blocks of V_i are merged with a constant number of iterations of the multi-way mergesort using the multi-way merging technique (Section 4). We have to distinguish two cases, though. If $|\mathcal{D}| = d' \geq \frac{(\lceil \log m \rceil)^3}{\log \log m}$, we can apply the solution to Problem 3 we presented in Section 4.

If, on the other hand, $|\mathcal{D}| = d' < \frac{(\lceil \log m \rceil)^3}{\log \log m}$ we may not have a sufficient number of distinct elements for the set $\mathcal{U}$ in Problem 3. However, if $d' < \frac{(\lceil \log m \rceil)^3}{\log \log m}$ then $|V_i| = d'(\lceil \log m \rceil)^2 = \mathit{polylog}(m)$. Therefore, we can use the same multi-way merging technique in Section 4 but with fragments of size $O(\log \log m)$ instead of $O(\log^2 m)$. That reduces the size of the set $\mathcal{U}$ from $O(s \log^2 m)$ to $O(s \log \log m)$. If we choose $s = \frac{\log m}{(\log \log m)^2}$, the number of iterations of the s-way mergesort needed to sort the whole block V_i is still a constant but the size of $\mathcal{U}$ is $O(\frac{\log m}{\log \log m})$. Therefore, the elements in $\mathcal{U}$ do not have to be distinct anymore because we can maintain in a single word of (real) auxiliary memory the whole permutation to bring them back to their original order when the fragmented s-way merging process is done.

Second Phase. After the first phase, each block V_i of V is sorted and divided into at most $d'' \leq d'$ runs of equal elements. Since $|V_i| = d'(\lceil \log m \rceil)^2$, the total number t_r of runs in V is less than or equal to $t/(\lceil \log m \rceil)^2$. For any run, let the first element be the *head* and the rest of the run be the *tail*.

First, each block V_i is divided into two sub-blocks H_i and V_i'. H_i contains the heads of all the runs of V_i and V_i' contains all the tails. Both H_i and V_i' are in sorted order. This subdivision can be easily accomplished with the same technique used in the second phase of the buffer extraction algorithm in Section 3. Since the elements we are extracting (the heads of the runs of a single block V_i) are distinct, we do not have to store their original order: we simply sort them when they are finally collected at the left end of V_i.

Second, some information about runs and blocks is collected and stored in $\mathcal{M}$. An array I_H with $\frac{|V|}{d'(\lceil \log m \rceil)^2}$ entries of two words each is stored in $\mathcal{M}$. For any i, the first word of $I_H[i]$ contains $|H_i|$ and the second word contains the index of the first run of V_i (the index is between 1 and t_r, from the leftmost run in V to the rightmost). An array I_R with t_r entries of four words each is stored in $\mathcal{M}$. For any i, the first word of $I_R[i]$ is initially set to i, the second one contains the address of the head of the i-th (in V) run, the third one contains the starting address of the tail of the i-th run and the fourth one contains the size of the i-th run. Finally, an array $I_{R^{-1}}$ with t_r entries of two words each is stored in $\mathcal{M}$. For any i, the first word of $I_{R^{-1}}[i]$ is initially set to i and the second word of $I_{R^{-1}}[i]$ is initially set to 1. All this information can be obtained simply by scanning V. In general, for any array I of multi-word entries, we will denote the p-th word of the i-th entry with $I[i][p]$.

Third, $I_{R^{-1}}$ is sorted stably by head, that is, at any time of the sorting process, the sorting key for the two-word value in the i-th entry of $I_{R^{-1}}$ is $V[I_R[I_{R^{-1}}[i][1]][2]]$. The sorting algorithm used is mergesort with a linear time in-place stable merging (e.g. [15]). During the execution of the algorithm, *every time the two-word value in the i-th entry of $I_{R^{-1}}$ is moved to the j-th entry, the corresponding entry in I_R is updated, that is $I_R[I_{R^{-1}}[j][1]][1]]$ is set to j.* We remark that *only the entries of the encoded array $I_{R^{-1}}$ are moved* (where any abstract move of an encoded value causes $O(\log m)$ actual moves of some elements contained in zones Q' and Q'' of Section 2). In this process, none of the elements in V are moved.

Fourth, for $i = 2$ to t_r, let $I_{R^{-1}}[i][2]$ be $I_{R^{-1}}[i-1][2] + I_R[I_{R^{-1}}[i][1]][4]$ (that is, if we had the elements in V sorted stably into another sequence V', $I_{R^{-1}}[i][2]$ would be the starting address in V' of the i-th run in the stable sorted order).

Third Phase. After the second phase we are able to evaluate the function $\alpha_V : \{1, \ldots, t\} \to \{1, \ldots, t\}$ such that $\alpha_V(j)$ is the rank of the element $V[j]$ in the sequence V, performing $O(\log m)$ comparisons.

1. Let V_i be the block of $V[j]$. We know where H_i starts and ends, in fact $H_i = V[s_i \ldots s_i + I_H[i][1] - 1]$ where $s_i = (i-1)d'(\lceil \log m \rceil)^2 + 1$. Therefore, we can perform a binary search for $V[j]$ in H_i and find the index p_j *in* V_i of the run to which $V[j]$ belongs.
2. The index p'_j *in* V of the run of $V[j]$ is $I_H[i][2] + p_j - 1$.
3. Using the array I_R, we can find the position k_j of $V[j]$ in its run. If $j = I_R[p'_j][2]$ then $k_j = 1$ ($V[j]$ is the head of its run). Otherwise, $k_j = j - I_R[p'_j][3] + 2$ ($V[j]$ belongs to the tail of its run).
4. Finally, we have that $\alpha_V(j) = I_{R^{-1}}[I_R[p'_j][1]][2] + k_j - 1$.

Using this algorithm and the given function *belongs_to_V*(x) to discern between the elements originally contained in V and the ones originally in G, it is possible to sort the elements in V efficiently, using G as internal buffer while preserving the original order of its elements.

Before the formal description of this last phase is given, a short outline is needed. The algorithm has two nested iterations. The outer iteration scans the elements of V following the sorted order (we know the order of the runs from the previous phase, therefore the elements can be scanned in sorted order easily). During the scan, three kinds of elements can be found: heads of runs, elements belonging to the tails of their runs, and buffer elements from G (as we will see, the inner iteration is responsible for the presence in these elements). If a buffer element is found (recognized using the given function *belongs_to_V*(x)), there is nothing to do: the element of V previously stored in this position has already reached its final destination. If a head is found, nothing can be done since the heads are the cornerstones of the algorithm used to find the rank of an element in V. As we will see, their treatment is delayed until the very end of the algorithm. Finally, if an element x of a tail is found, the inner iteration starts. The purpose of the inner iteration is to scan the cycle (of the permutation that disposes the elements of V in sorted order) to which the element belongs. During the scan of

the cycle of x two kinds of elements can be found: heads of runs and tail elements (the first found is obviously x). Again, the heads are left in their position. On the other hand, any tail element y is ranked (with α_V), let its rank in V be r_y, and is exchanged with the element in G corresponding to its rank, that is $b_y = G[r_y]$. Then, there can be two cases: if $V[r_y]$ is a head, it cannot be moved and then b_y is left in the position in V previously occupied by y and treated in a special way; if $V[r_y]$ is a tail element, we immediately exchange $V[r_y]$ with b_y, recovering the correct position for b_y. Therefore, the next element of the cycle is in the position previously occupied by y. After the two nested iterations, a final simple iteration performs t_r exchanges that bring the heads in their final positions.

```
 1: FOR i = 1 to t_r and j = 2 to I_R[I_{R^-1}[i][1]][4] DO
 2:    start ← I_R[I_{R^-1}[i][1]][3] + j − 1
 3:    IF belongs_to_V(V[start]) THEN
 4:       next ← α_V(start)
 5:       WHILE next ≠ start DO
 6:          Exchange V[start] and G[next]
 7:          IF is_head(next) THEN
 8:             next ← α_V(next)
 9:          ELSE
10:             next_tmp ← next
11:             next ← α_V(next)
12:             Exchange V[start] and V[next_tmp]
13: FOR i = 1 to t_r DO
14:    head ← I_R[I_{R^-1}[i][1]][2]
15:    Exchange V[head] and G[I_{R^-1}[i][2]]
16: Exchange G and V
```

The function $is\_head(x)$ returns true if $V[x]$ is the head of its run. It can be calculated in the very same way the rank of an element in V is (with the exclusion of the fourth step).

With the solution to the abstract Problem 4, we can finally sort the sequence CB when $b = |B| < \lceil r / \log m \rceil$.

1. C is partitioned into three sub-sequences $C'UC''$, where U contains all the elements equal to the element c_m of rank $\lceil |C| / 2 \rceil$ in C, and C', C'' contain all the elements of C, respectively, less than and greater than c_m. This partition can be easily obtained using the stable in-place selection and the stable in-place partitioning in [4, 5].
2. To sort C' and C'' we can apply the solution to Problem 4. First, we set $V = C'$, $G = (UC'')[1 \ldots |C'|]$, $D = B$, $belongs\_to\_V(x) = (x < c_m)$ and sort C'. Then we set $V = C''$, $G = (C'U)[1 \ldots |C''|]$, $D = B$, $belongs\_to\_V(x) = (c_m < x)$ and sort C''. (obviously there can be extreme situations in which C' or C'' are void)
3. B is sorted with the normal mergesort using a linear time in-place stable merging (e.g. [15]) and the two sequences are merged (again with [15]).

We have that:

Theorem 3. *Under Assumption 1, a sequence CB that satisfies Property 1 with $|B| < \lceil r/\log m \rceil$ can be sorted, stably, using $O(1)$ cells of auxiliary memory, performing $O(m \log m)$ comparisons and $O(m)$ moves in the worst case.*

By Theorems 1, 2 and 3 we can conclude that Problem 1 is solved and state the main result of the paper.

Theorem 4. *Any sequence of n elements can be sorted stably, using $O(1)$ auxiliary locations of memory, performing $O(n \log n)$ comparisons and $O(n)$ moves in the worst case.*

References

[1] Gianni Franceschini and Viliam Geffert. An In-Place Sorting with $O(n \log n)$ Comparisons and $O(n)$ Moves. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.

[2] Gianni Franceschini and Roberto Grossi. Optimal cache-oblivious implicit dictionaries. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, 2003.

[3] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, April 1962.

[4] J. Katajainen and T. Pasanen. Sorting multisets stably in minimum space. In O. Nurmi and E. Ukkonen, editors, *SWAT '92*, volume 621 of *LNCS*, pages 410–421, Helsinki, Finland, 8–10 July 1992. Springer-Verlag.

[5] J. Katajainen and T. Pasanen. Stable minimum space partitioning in linear time. *BIT*, 32(4):580–585, 1992.

[6] J. Katajainen and T. Pasanen. In-place sorting with fewer moves. *Inform. Process. Lett.*, 70:31–37, 1999.

[7] D. E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, 1973.

[8] M. A. Kronrod. Optimal ordering algorithm without operational field. *Soviet Math. Dokl.*, 10:744–746, 1969.

[9] T. W. Lai and D. Wood. Implicit selection. In *SWAT '88*, volume 318 of *LNCS*, pages 14–23, 1988.

[10] J. I. Munro. An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time. *J. Comput. System Sci.*, 33:66–74, 1986.

[11] J. I. Munro and V. Raman. Sorting with minimum data movement. *J. Algorithms*, 13:374–93, 1992.

[12] J. I. Munro and V. Raman. Fast stable in-place sorting with $O(n)$ data moves. *Algorithmica*, 16:151–60, 1996.

[13] J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theoret. Comput. Sci.*, 165:311–23, 1996.

[14] Luis Trabb Pardo. Stable sorting and merging with optimal space and time bounds. *SIAM Journal on Computing*, 6(2):351–372, June 1977.

[15] Jeffrey Salowe and William Steiger. Simplified stable merging tasks. *Journal of Algorithms*, 8(4):557–571, December 1987.

[16] J. W. J. Williams. Heapsort (Algorithm 232). *Comm. Assoc. Comput. Mach.*, 7:347–48, 1964.

Cycle Cover with Short Cycles

Nicole Immorlica, Mohammad Mahdian, and Vahab S. Mirrokni*

Computer Science and Artificial Intelligence Lab,
MIT, Cambridge, MA 02139
{nickle, mahdian, mirrokni}@theory.lcs.mit.edu

Abstract. Cycle covering is a well-studied problem in computer science. In this paper, we develop approximation algorithms for variants of cycle covering problems which bound the size and/or length of the covering cycles. In particular, we give a $(1 + \ln 2)$-approximation for the lane covering problem [3, 4] in weighted graphs with metric lengths on the edges and an $O(\ln k)$ approximation for the bounded cycle cover problem [9] with cycle-size bound k in uniform graphs. Our techniques are based on interpreting a greedy algorithm (proposed and empirically evaluated by Ergun et al. [3, 4]) as a dual-fitting algorithm. We then find the approximation factor by bounding the solution of a factor-revealing non-linear program. These are the first non-trivial approximation algorithms for these problems. We show that our analysis is tight for the greedy algorithm, and change the process of the dual-fitting algorithm to improve the factor for small cycle bounds. Finally, we prove that variants of the cycle cover problem which bound cycle size or length are APX-hard.

1 Introduction

Given a graph and a subset of marked elements (nodes, edges, or some combination thereof), a cycle cover problem seeks to find a minimum length set of cycles whose union contains all marked elements. Many practically important problems in routing and navigation can be formulated as cycle cover problems with additional constraints on the set of cycles in the solution.

One commonly studied cycle cover problem is the *Chinese postman problem*, first introduced in 1962 by Guan [8], in which the objective is to cover every edge at least once by one (not necessarily simple) cycle of minimum length. Besides its obvious application to mail delivery in China, this problem finds application in a variety of routing problems such as robot navigation and city snow plowing planning.

In many applications of a similar nature, the objective is to find *several* covering cycles with an additional constraint on the size or length of the cycles. For example, a group of companies might want to design a set of trucking routes (cycles) of minimum cost that satisfy all their shipping requirements (i.e., traverses a set of given edges) and obey union regulations which limit the driving time

* Supported in part by NSF contracts ITR-0121495 and CCR-0098018.

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 641–653, 2005.

and number of stops each trucker can make [3, 4]. In graph theoretic terms, this translates to covering all or some of the edges of a given graph with cycles, with an upper bound on the size (i.e., number of edges) or length (i.e., total distance) of each covering cycle. Another application arises in the design of fault-tolerant optical networks. In this application, studied by Hochbaum and Olinick [9], the objective is to find a backup path for every edge of the network, so that when a link of the optical network fails, the network can route traffic around the fault without increasing the size (and hence the errors) of the transmission by more than a bounded amount. This reduces to covering the graph with short cycles with an additional constraint that the cycles should be simple.

Although the Chinese postman problem is polynomially solvable in directed and undirected graphs, any variant which allows multiple covering cycles but places a constant upper bound on the size or length of the covering cycles is NP-hard [4]. In fact, we will show that these variants are APX-hard.

In this paper, we study approximation algorithms for the problem of finding cycles of bounded size that cover a subset of the edges of a graph. We usually assume the edge lengths of the graph form a metric. This problem is also known as the *lane covering problem* [3, 4]. To the best of our knowledge, the only approximation algorithm known for this problem is a trivial 2-approximation algorithm that covers each edge with a cycle of size 2. We show that a greedy heuristic proposed and empirically evaluated by Ergun et al. [3, 4] can be interpreted as a dual-fitting algorithm in which edges grow their dual variables at rate proportional to their length (see [12] for a discussion of the technique of dual-fitting). We use this fact, and a factor-revealing non-linear program (see [12]) to show that this algorithm achieves an approximation factor of $1+(k-1)(1+2^{-1/(k-1)})$ for constant k, and $1+\ln(2) \approx 1.69$ if k is given as part of the input. This is the first approximation algorithm that provably beats the trivial 2-approximation algorithm. Using the factor-revealing program, we show that our analysis is tight for this greedy algorithm. For small values of k, we show how the approximation factor of the algorithm can be improved by increasing dual variables at a rate that *non-linearly* depends on the length of the edges. In particular, for $k = 3$ we show that the approximation factor can be improved to 1.54 from $3-\sqrt{2} \approx 1.59$.

We also explore several variants of the problem and show how our algorithm extends to these variants. One problem that we will consider is the lane covering problem with a constraint on the length, as well as the size of the cycles. We show that for this problem our algorithm gives the approximation factor $1 + \ln 2 + \epsilon$ for any $\epsilon > 0$. Another problem, called the *bounded cycle cover problem*, has the additional restriction that cycles should be simple as well as of bounded size [9]. For this problem, our approach gives the first $O(\ln k)$-approximation algorithm.

We also prove that cycle cover problems which place a bound on the size or length of the cycles is APX-hard. Our proof uses a construction of Holyer [10].

Related Works. Cycle cover problems in graphs have been studied extensively from a combinatorial standpoint. The book of Zhang [20] reviews much of this literature. The Chinese postman problem was first introduced by Guan [8]. Edmonds and Johnson [2] gave the first polynomial time algorithms for the problem

in undirected graphs. Papadimitriou [15] proved that the problem is NP-hard in mixed graphs. Raghavachari and Veerasamy [16] gave a 3/2-approximation for this instance of the problem. A variant of the Chinese postman problem, the *minimum weight cycle cover problem*, adds the restriction that covering cycles must be simple. This problem was shown to be NP-hard by Thomassen [18]. Itai et al. [11] proved an upper bound on the length of such a cycle cover in 2-connected graphs and gave an algorithm to find it. The bounded cycle cover problem, which constrains cycles to be of bounded size as well as simple, was introduced by Hochbaum and Olinick [9] to solve an optical network design problem. They presented a heuristic for the problem along with an empirical analysis. *Ring covering*, a related optical network design problem with a slightly different objective was proposed by Slevinsky et al. [17]. Kennington et al. [13] present a heuristic to solve the problem. The lane covering problem was introduced by Ergun et al. [3, 4], who gave a heuristic for the problem along with an empirical analysis. A variant on the cycle covering problem which imposes a lower bound on the size of each cycle has been studied as well [1]. Other covering problems include covering a graph by cliques [7].

Structure of the Paper. In Section 2, we give a formal statement of the lane covering problem. In Section 3 we present a natural greedy algorithm and analyze it in Section 4. In Section 5, we present a method that improves the approximation factor of our algorithm for $3 \leq k \leq 5$. In Section 6, we discuss two related cycle covering problems to which we can apply our techniques. Finally, in Section 7, we present our APX-hardness result.

2 Problem Statement

Let $G = (V, E)$ be a complete bidirected graph. A nonnegative length ℓ_e is assigned to each edge $e \in E$. These lengths are symmetric (i.e., $\ell_{uv} = \ell_{vu}$ for every $u, v \in V$) and satisfy the triangle inequality (i.e., $\ell_{uv} \leq \ell_{uw} + \ell_{wv}$ for every $u, v, w \in V$). In the *lane covering problem* [3, 4], we are given a subset L of directed edges of G called *lanes* and an integer $k \geq 3$. The objective is to find a collection of (not necessarily disjoint nor simple) cycles that cover all edges of L, each containing at most k edges, with minimum total length. In another variant of the lane covering problem, the *length-constrained lane covering problem* [3, 4], we are also given a bound B on the length of each covering cycle. The goal is to find a minimum length cycle cover of L of cycles of length at most B and size at most k. Except where noted, in this paper, we will focus on the lane covering problem. However, as shown in Section 6, our algorithmic techniques and lower bounds also apply to the more general length-constrained lane covering problem.

3 The Greedy Algorithm

In this section we present a natural greedy algorithm for the lane covering problem that was first proposed and analyzed empirically by Ergun et al. [4]. This

$$\min \sum_{C \in \mathcal{C}} \ell_C x_C$$
$$\text{s.t. } \forall e \in L : \sum_{C:\ e \in C} x_C \geq 1$$
$$\forall C \in \mathcal{C} :\ x_C \geq 0$$

(a) Primal program

$$\max \sum_{e \in L} \ell_e \alpha_e$$
$$\text{s.t. } \forall C \in \mathcal{C} : \sum_{e \in L \cap C} \ell_e \alpha_e \leq \ell_C$$
$$\forall e \in L :\ \alpha_e \geq 0$$

(b) Dual program

Fig. 1. Linear programming relaxation

algorithm relies on a notion of the *cost effectiveness* of a cycle C, similar to the one used in the greedy set cover algorithm. We define the *cost effectiveness* of a cycle C as the ratio of the total length of edges in $C \cap L$ to the total length of the edges in C. Using this notation, the algorithm can be stated as follows.

Algorithm 1

- While there is an edge in L, do the following
  - Find the most cost-effective cycle C in the graph consisting of at most k edges. If there is more than one such cycle, pick one arbitrarily.
  - Pick C and remove its edges from L.

When k is a constant, the number of cycles of size k is at most a polynomial in the size of the graph, and therefore Algorithm 1 can clearly be implemented in polynomial time. However, when k is part of the input, it is not clear how to implement this algorithm efficiently. More precisely, in order to establish a polynomial running time for Algorithm 1, we need to show that it is possible to find the most cost-effective cycle in polynomial time. This is done in the following lemma, the proof of which is deferred to the full version of the paper.

Lemma 1. *There is a polynomial time algorithm that given a graph G, a non-negative length ℓ_e for every $e \in E(G)$, a set $L \subseteq E$ of lanes, and a parameter k computes the most cost-effective cycle in G of size at most k.*

Here we present a different formulation of Algorithm 1, that allows us to analyze it using the method of dual fitting. Before stating the algorithm, we present an LP relaxation of the problem (see Figure 1(a)). In the LP relaxation of the problem, $\mathcal{C}$ denotes the collection of all cycles with at most k edges in G, and for a cycle C, ℓ_C denotes $\sum_{e \in C} \ell_e$. The dual of this program can be written as shown in Figure 1(b). Notice that the dual variable corresponding to the first inequality in the primal program is $\ell_e \alpha_e$. We are now ready to describe the restatement of Algorithm 1 in terms of the dual variables α_e:

Algorithm 2

- Initialize α_e's to zero for all $e \in L$.
- Increase all α_e's at the same rate until one of the following events occur. If two events happen at the same time, break the tie arbitrarily.

- For a cycle $C \in \mathcal{C}$, sum of $\ell_e \alpha_e$ for all $e \in L \cap C$ becomes equal to ℓ_C (In other words, the edges in $L \cap C$ can pay for the cycle C with their dual variables). In this case, pick C, freeze the value of α_e for $e \in L \cap C$, and remove these edges from L (i.e., these edges will not contribute to other cycles any more).

As shown in the next section, the above formulation of the greedy algorithm enables us to use the technique of dual-fitting in combination with a factor-revealing program to analyze the algorithm.

4 Analysis

The idea behind primal-dual algorithms is that the algorithm computes a solution for the problem (the primal solution), together with a *feasible* solution for the dual linear program, so that the ratio of the cost of the two solutions can be bounded by a factor λ. Since by LP duality every feasible solution of the dual LP is a lower bound on the cost of the optimal primal solution, this would imply that the algorithm has an approximation factor of λ.

Algorithm 2 computes a solution for the problem, and a solution α_e for the dual LP such that the cost of the primal solution is equal to the cost of the dual solution ($\sum_{e \in L} \ell_e \alpha_e$). However, α_e's do not necessarily constitute a feasible solution for the dual. The idea of dual-fitting [19] is to find a value λ such that when we divide all α_e's by λ, we obtain a feasible dual solution. Since this feasible dual solution has cost equal to the cost of the primal solution divided by λ and is also a lower bound on the optimal value, this proves that the algorithm is a λ-approximation.

In order to find the best λ for which the above analysis works, we use the technique of factor-revealing programs [12]. This technique consists of proving several inequalities between various parameters in the instance of the problem and writing them as a maximization program whose solution bounds the worst value for λ. We call this maximization program a *factor-revealing program*. Unlike [12], the factor-revealing program that we get is non-linear. The final step of the analysis is to bound the solution of this program.

For a cycle $C \in \mathcal{C}$, denote the edges of $L \cap C$ by $e_1, e_2, \ldots, e_p$ and their corresponding α_e's and ℓ_e's by $\alpha_1, \ldots, \alpha_p$ and $\ell_1, \ldots, \ell_p$, and let ℓ_C denote the total length of edges in C (i.e., sum of ℓ_i's plus the length of the edges in $C \setminus L$). We would like to find a constant λ so that for all cycles $C \in \mathcal{C}$, we have $\frac{1}{\lambda} \sum_{i=1}^{p} \ell_i \alpha_i \leq \ell_C$. The smallest such constant is equal to the maximum of the ratio $\left(\sum_{i=1}^{p} \ell_i \alpha_i\right) / \ell_C$, where the maximum is taken over all cycles $C \in \mathcal{C}$ in all instances of the lane covering problem.

The idea is to prove several inequalities between α_i's, ℓ_i's, and ℓ_C, and write them as the constraints of a factor-revealing program (treating α_i's, ℓ_i's and ℓ_C as variables) with $\left(\sum_{i=1}^{p} \ell_i \alpha_i\right) / \ell_C$ as the objective function. The solution of this maximization program gives us an upper bound on the smallest feasible value of λ. We start by assuming, without loss of generality, that

$$\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_p \tag{1}$$

This means that Algorithm 2 first covers e_1 at time α_1, then covers e_2 at time α_2, and so on. Consider the time $t = \alpha_i$, just before the algorithm covers e_i. At this moment, all of the edges $e_i, e_{i+1}, \ldots, e_p$ are not covered yet, and therefore they are contributing toward the cycle C. The total value of this contribution is $t \sum_{j \geq i} \ell_j$. This value cannot be greater than the length of C, since otherwise we would have picked C earlier in the algorithm. Thus, for every i we have the following inequality:

$$\alpha_i \sum_{j \geq i} \ell_j \leq \ell_C. \tag{2}$$

Furthermore, each edge e is contained in a cycle of size two of length $2\ell_e$, and it can pay for this cycle at time 2. Thus, Algorithm 2 never increases an α_e beyond 2. So, for every i,

$$\alpha_i \leq 2. \tag{3}$$

The last inequality is the metric inequality: for every edge e_i in the cycle, the length of this edge is at most the cost of the path between the endpoints of e_i that uses the other edges of the cycle. Therefore, for every i,

$$\ell_i \leq \ell_C - \ell_i \tag{4}$$

Summarizing all the above inequalities, we get the following lemma.

Lemma 2. *Fix a cycle C with p edges in L, and scale all ℓ_i's so that $\ell_C = 1$. Let $\lambda_k := \max_{1 \leq p \leq k}\{z_p\}$, where z_p denotes the solution of the following maximization program.*

$$\text{maximize} \quad \sum_{j=1}^{p} \ell_j \alpha_j \tag{5}$$

$$\text{subject to} \quad \alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_p \tag{6}$$

$$\forall i: \quad \alpha_i \leq \frac{1}{\sum_{j \geq i} \ell_j} \tag{7}$$

$$\forall i: \quad \alpha_i \leq 2 \tag{8}$$

$$\forall i: \quad \ell_i \leq \frac{1}{2} \tag{9}$$

$$\sum_{j=1}^{p} \ell_j \leq 1 \tag{10}$$

$$\forall i: \quad \ell_i \geq 0 \tag{11}$$

Then Algorithm 2 is a λ_k-approximation algorithm.

Proof. By the above argument, for every cycle C with p edges in L, the values of ℓ_i's (scaled by l_C) and α_i's constitute a feasible solution of the maximization program (5). Thus, $(\sum_{j=1}^{p} \ell_j \alpha_j) \leq z_p \leq \lambda_k$. Therefore, if we scale down all α_e's

by a factor of λ_k, they will satisfy the constraints of the dual program (1(b)). This means that $\frac{1}{\lambda_k}\sum_{e\in L}\ell_e\alpha_e$ is a lower bound on the value of the optimal solution. On the other hand, it is clear from the description of Algorithm 2 that the cost of the solution is precisely $\sum_{e\in L}\ell_e\alpha_e$. Thus, Algorithm 2 always outputs a solution whose cost is at most λ_k times the cost of the optimal solution.

All that remains is to prove an upper bound on the value of λ_k.

Lemma 3. *For every k, the value λ_k defined by the factor-revealing program (5) in Lemma 2 is at most $1+(k-1)\left(1-2^{-1/(k-1)}\right)$.*

Proof. Let $\beta_i := \frac{1}{\sum_{j\geq i}\ell_j}$. Since ℓ_i's are nonnegative, β_i's are nondecreasing. Thus, there exists an index r, $0\leq r\leq p$, such that $\beta_i<2$ for all $i=1,\ldots,r$, and $\beta_i\geq 2$ for $i=r+1,\ldots,p$. On the other hand, for every i, we have $\ell_i=\frac{1}{\beta_i}-\frac{1}{\beta_{i+1}}$. Using this and inequalities (7) and (8), the objective function of the above program can be bounded in terms of β_i's as follows:

$$\begin{aligned}\sum_{j=1}^{p}\ell_j\alpha_j &\leq \sum_{j=1}^{r}\ell_j\beta_j+\sum_{j=r+1}^{p}2\ell_j\\ &=\sum_{j=1}^{r}(\frac{1}{\beta_j}-\frac{1}{\beta_{j+1}})\beta_j+\frac{2}{\beta_{r+1}}\\ &=r-\left(\frac{\beta_1}{\beta_2}+\frac{\beta_2}{\beta_3}+\cdots+\frac{\beta_r}{\beta_{r+1}}\right)+\frac{2}{\beta_{r+1}}\end{aligned}$$

By definition of r, $\beta_r\leq 2$. Therefore, the above expression is a decreasing function of β_{r+1}. On the other hand, by the definition of r, $\beta_{r+1}\geq 2$. Thus, the above expression can be bounded by:

$$\begin{aligned}\sum_{j=1}^{p}\ell_j\alpha_j &\leq r-\left(\frac{\beta_1}{\beta_2}+\frac{\beta_2}{\beta_3}+\cdots+\frac{\beta_r}{2}\right)+1\\ &\leq r-r\left(\frac{\beta_1}{2}\right)^{1/r}+1,\end{aligned}$$

where the last inequality follows from the inequality between geometric and arithmetic means. By inequality (10), $\beta_1\geq 1$, and hence the above expression is at most $1+r\left(1-2^{-1/r}\right)$. It is straightforward to see that this is an increasing function of r. By inequality (9), $\beta_p\geq 2$ and therefore $r\leq k-1$. Thus, the objective function of the maximization program can be bounded by

$$1+(k-1)\left(1-2^{-1/(k-1)}\right).$$

These results can be summarized in the following theorem.

Theorem 1. *For every fixed k, Algorithm 1 is a polynomial-time approximation algorithm for the lane covering problem with an approximation ratio at most $1 + (k-1)\left(1 - 2^{-1/(k-1)}\right)$. If k is part of the input, the approximation ratio of this algorithm is at most $1 + \ln(2)$.*

Proof. The theorem follows from Lemmas 2, 3, and 1, and the fact that for every k, the value $1 + (k-1)\left(1 - 2^{-1/(k-1)}\right)$ is less than $1 + \ln(2) < 1.69$, and tends to $1 + \ln(2)$ as k tends to infinity.

4.1 A Tight Example

The factor-revealing program (5) suggests how one can find a tight example for the algorithm. In this section, we use this approach to show that the approximation guarantee given by Theorem 1 is asymptotically tight. We construct an example in which the cycles of the optimal solution consist entirely of edges in L, but the algorithm still returns a sub-optimal solution. The idea is to place the cycles of the optimal solution close together so that non-optimal cycles go tight as well, confusing the greedy algorithm.

Theorem 2. *For every $\epsilon > 0$ there is an instance of the lane covering problem such that the ratio of the cost of the solution found by Algorithm 1 to the optimal solution is at least $1 + \ln 2 - \epsilon$.*

Proof Sketch. Let k be even and consider a $(k/2)$-regular bipartite graph H with girth at least $2k$ (for existence of such graphs, see for example [14]). By Konig's theorem, H is $(k/2)$-edge-colorable. Below, we construct a new graph G by replacing each vertex of H with a cycle and adding edges between cycles corresponding to adjacent vertices in H. Cycles corresponding to the vertices of H will give an optimal cycle cover for G, while Algorithm 2 (which is equivalent to Algorithm 1) will only pick cycles corresponding to the edges of H.

Each vertex of H is replaced by a directed cycle consisting of k arcs of length $1/k$. Let B denote the set of such cycles. The arcs of cycles in B form the set L. Fix a $(k/2)$-edge-coloring of H with colors from $\Sigma := \{1, \ldots, k/2\}$. For each vertex v in H, color the arcs of the cycle corresponding to v with colors in $\Sigma \cup \{0\}$ such that every other arc in the cycle is colored with 0 and every color in Σ is used exactly once in this cycle. We would like to add non-lane edges between these cycles so that Algorithm 2 covers every color-i arc $e \in L$ at time

$$\alpha_i = \begin{cases} k/(k-i+1) & \text{if } i \geq 1 \\ 2 & \text{if } i = 0. \end{cases}$$

To achieve this, for every edge uv of color i in H, we add two parallel non-lane edges between the endpoints of the color-i arcs in the cycles corresponding to u and v. More precisely, if the color-i arcs in these two cycles are $a_u b_u$ and $a_v b_v$ we add two non-lane edges, one between b_u and a_v, and one between b_v and a_u, each of length $\frac{i-1}{k(k-i+1)}$. This creates a cycle $a_u b_u a_v b_v$. Let A_i denote the set of such cycles. The length of a cycle $a_u b_u a_v b_v$ in A_i is $\frac{2}{k} + \frac{2(i-1)}{k(k-i+1)} = \frac{2}{k-i+1}$, so

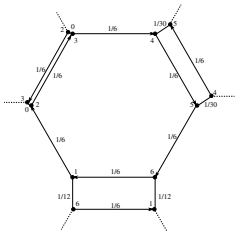


Fig. 2. A local view of the construction for $k = 6$

Algorithm 2 picks this cycle at time $k/(k-i+1)$, assuming neither arc $a_u b_u$ nor $a_v b_v$ is covered at an earlier time. Thus, color-i arcs are covered by time $k/(k-i+1)$. Let G denote the resulting graph. See Figure 2 for an example when $k = 6$. An instance of the lane covering problem is obtained by setting the length of all edges to the length of the shortest path between their endpoints in the underlying undirected graph of G.

We now sketch a proof of the fact that the only cycles picked by Algorithm 2 at any time before 2 in the above instance are those in $\cup_{i=1}^{k/2} A_i$. For the sake of contradiction, assume there is a cycle outside $\cup_{i=1}^{k/2} A_i$ that is picked by the algorithm at a time before 2 and let C be the first such cycle. Let $c_1, \ldots, c_m$ be the cycles of B which intersect C in at least one vertex. These cycles correspond to some vertices of H. The proof considers the subgraph H' of H induced by these vertices and argues that, due to the large girth of H and the assumed bound on α, H' must be a single vertex. This essentially implies that the only cycles Algorithm 2 can pick at a time earlier than 2 are cycles in A_i and cycles in B. One can further show that whenever the algorithm can choose a cycle in B, it also has a valid choice in $\cup_{i=1}^{k/2} A_i$. Therefore, some implementation of Algorithm 2 buys all cycles in $\cup_{i=1}^{k/2} A_i$ along with a bunch of cycles with two edges and spends $\sum_{i=1}^{k/2} \frac{\alpha_i}{k} + \frac{k}{2} \cdot \frac{\alpha_0}{k}$ per cycle in B, whereas the optimal solution spends 1 per cycle in B. Thus, the greedy solution costs λ times more than the optimal solution, where λ is $\lambda = \sum_{i=1}^{k/2} \frac{1}{k-i+1} + 1 = 1 + \mathrm{H}_k - \mathrm{H}_{k/2}$. This tends to $1 + \ln 2$ as k tends to infinity.

Theorem 2 shows that our analysis is asymptotically tight. For the case of $k = 3$, our analysis is also tight. Again, we can use the factor-revealing program to construct an example and prove that factor $3 - \sqrt{2}$ is tight for this algorithm. The example is deferred to the full version of the paper.

5 Covering with Small Cycles

As claimed in the previous section, the approximation factor of Algorithm 1 can be as bad as $3 - \sqrt{2} \approx 1.59$ when $k = 3$. In this section, we show how to improve this factor. The idea is to grow the *budget* of each edge e in Algorithm 2 at a rate proportional to ℓ_e^r, for some $r > 1$, instead of growing it at a rate

proportional to ℓ_e, and then optimize over r. Similar to the previous section, we can derive a factor-revealing program that computes the approximation ratio of this algorithm.

For $r = 1.18$, numerical results indicate that the approximation factor of the resulting algorithm for $k = 3$ is at most 1.54, and thus it performs better than Algorithm 1 in the worst-case. For $k = 4$ and $k = 5$, the approximation factor improves to 1.59 and 1.62 from 1.61 and 1.63, respectively.

In the tight example given in Theorem 2, the length of all lane edges are equal, so this algorithm can not improve upon the approximation factor of Algorithm 1 when k is not a constant.

6 Extensions

In this section, we show that our algorithmic ideas can be adapted to solve related covering problems as well. In fact, we can solve any variant of our problem that allows cycles of size two so long as the subproblem of finding the most cost-effective cycle can be solved in polynomial time. As we will see, it is sufficient to have an approximation algorithm for this subproblem (the approximation factor of the subproblem will affect the final approximation factor). We study two problems in particular: the length-constrained lane covering problem, and the cycle cover problem with simple short cycles.

6.1 Length-Constrained Lane Covering Problem

Recall that in the length-constrained lane covering problem, an additional input B is given, and the objective is to cover the lanes with cycles with at most k edges *and* total length at most B. The following theorem shows that although finding the most cost-effective cycle is as hard as the NP-hard shortest weight-constrained path problem [6], Algorithm 1 gives a $(1 + \ln 2 + \epsilon)$-approximation for this problem for every $\epsilon > 0$.

Theorem 3. *For every $\epsilon > 0$, Algorithm 1 is a polynomial-time $(1 + \ln 2 + \epsilon)$-approximation algorithm for the length-constrained lane covering problem.*

Proof Sketch. In order to use Algorithm 1, we need to find the most cost effective cycle of length at most B. Similar to the proof of Lemma 1, for a given R we need to check if the cost effectiveness of a cycle in G is greater than R or not. We construct a new graph H whose edges have the same lengths as in G. We set the cost of an edge $e \in E(H)$ to be $c(e) = R\ell_e$ for $e \in E(H)\backslash L$ and $c(e) = (R-1)\ell_e$ for $e \in L$. In order to check if there is a cycle of length at most B with cost effectiveness R in G, we need to check if there is a cycle of length at most B with negative cost in H, or, equivalently, find cheapest length-constrained paths in H. If costs are from polynomially bounded integer numbers, we can solve this problem optimally using dynamic programming. Thus, by rounding the costs to multiples of δ we can check if there exists a path of cost at most $n\delta$ with length at most B (where n is the number of vertices in the graph). Similar

to the proof of Lemma 1, we use binary search to find the maximum value of R for which there is cycle of length at most B and cost at most $n\delta$. We can find such a cycle in polynomial time. Using this method of finding the most cost-effective cycle, instead of inequality 2 in the factor-revealing program, we have $\alpha_i \sum_{j \geq i} \ell_j \leq \ell_C + n\delta$. Simple calculations prove that the approximation factor of this polynomial-time algorithm is at most $(1 + \ln 2 + \epsilon)$ for δ small enough.

6.2 Cycle Cover with Simple Short Cycles

Our techniques also give results on the *bounded cycle cover problem* [9]. In the bounded cycle cover problem, we look for cycles of size at most k with the added restriction that the cycles are *simple*, i.e., do not repeat any edge. We show that an algorithm similar to Algorithm 1 gives an $O(\ln k)$-approximation for the bounded cycle cover problem in the special case of uniform graphs. To the best of our knowledge, this is the first approximation known for this problem.

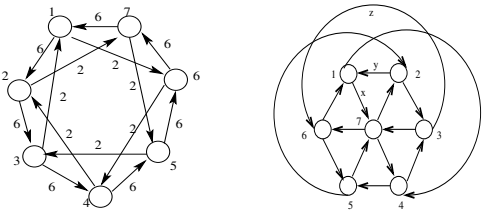
Given a graph G, our algorithm first checks that the instance is feasible (i.e., that every edge is in a cycle of size at most k). Then it greedily selects the most cost-effective feasible cycle and iterates until all edges are covered by a cycle in the solution set. As in Lemma 1, this can be done in polynomial time, even with the added restriction that cycles be simple.

We follow the analysis in Section 4. First, we derive inequalities for the factor-revealing program. Fix a cycle C of the optimal solution. Our input graph is no longer a complete bidirected graph, so we no longer have the inequality $\alpha_i \leq 2$. However, by the feasibility of the instance, we know that each edge is in a cycle of size at most k. Therefore, $\alpha_i \leq k$. Furthermore, the graph is uniform, so after scaling $\ell_i = \frac{1}{p}$ where p is the size of the cycle C. Thus, $\alpha_i \sum_{j \geq i} \ell_i \leq 1$ implies $\alpha_i \leq \frac{p}{p-i+1}$. From these inequalities, it is easy to see that $\sum_{i=1}^{p} \ell_i \alpha_i \leq \sum_{i=1}^{p} \frac{1}{p} \min(\frac{p}{p-i+1}, k) = \sum_{i=1}^{s} \frac{p}{p-i+1} + \sum_{i=s}^{p} \frac{k}{p} = H(p+1) - H(\frac{p}{k}) + 1 = O(\ln k)$ where $s = 1 + (\frac{k-1}{k})p$ and H is the harmonic series.

7 Lower Bounds

In this section, we prove two results regarding the hardness of approximating the lane covering problem. Our first result shows the APX-hardness of the lane covering problem via a reduction from a version of the maximum satisfiability problem, *5-OCC-MAX-3SAT*. In our reduction, all edges of the graph whose lengths are not given by the underlying path metric are lane edges, and so this result actually proves that any variant of the Chinese postman problem which constrains the size or length of covering cycles, such as the bounded cycle cover problem mentioned in the introduction, is APX-hard.

Our reduction is based on a reduction used by Holyer [10] to prove NP-hardness of some edge-partitioning problems. Given a satisfiability formula, Holyer constructs a series of graphs for every variable and clause. By gluing



(a) $k = 3$ (b) $k = 4$

Fig. 3. Integrality gap examples

these graphs together in a structure as dictated by the satisfiability formula, Holyer guarantees that if the formula is unsatisfiable, then the resulting graph will have no triangle partitioning. We adapt this proof to work for our setting, using the maximum satisfiability problem *5-OCC-MAX-3SAT* in order to prove an APX-hardness result. In an instance of the 5-OCC-MAX-3SAT problem, we are given a CNF formula with n variables and $m = \frac{5n}{3}$ clauses of exactly three literals in which each variable occurs exactly five times, and we want to find a truth assignment satisfying the maximum number of clauses. Fiege [5] proved that it is NP-hard to distinguish between a 5-OCC-MAX-3SAT instance in which all the clauses can be satisfied and one in which at most a b fraction of the clauses can be satisfied for some constant b. We prove a relationship between the number of satisfiable clauses in a 5-OCC-MAX-3SAT formula and the cost of an optimal lane covering in our corresponding construction. This means that an algorithm with a suitably small approximation factor can be used to distinguish between the types of 5-OCC-MAX-3SAT formulas mentioned above. The proof is deferred to the full version of the paper.

Theorem 4. *The lane covering problem is APX-hard for any constant k.*

Our second result addresses the effectiveness of LP-based approaches for the lane covering problem. Although Theorem 2 shows that Algorithm 2 is asymptotically tight, there might be a better LP-based rounding algorithm for the set cover LP formulation. We can lower bound the approximation ratio of any such algorithm by analyzing the integrality gap of the set cover LP, LP 1(a) For $k = 3$, consider the union of two cycles of size 7 with edge lengths as specified in Figure 3(a). It is not hard to check that the optimal fractional solution of the LP for this example is 61.25 and optimal integral solution is 67. Thus, the integrality gap is $\frac{67}{61.25} \approx 1.09$. We achieve our best lower bound of 1.15 for the integrality gap when $k = 4$. In this case, the example is the union of squares and triangles such that every edge is in exactly two cycles of size at most 4 (Figure 3(b)).

Acknowledgements. We would like to thank Ozlem Ergun for introducing us to this problem and Michel Goemans for helpful comments.

References

1. M. Blaser and B. Siebert. Computing cycle covers without short cycles. In *Proceedings of the 34st Annual European Symposium of Algorithms*, 2001.
2. J. Edmonds and E. Johnson. Matching euler tours and the chinese postman problem. *Mathematical programming*, 5:88–124, 1973.
3. Ozlem Ergun, Gultekin Kuyzu, and Martin Savelsbergh. Collaborative logistics: The shipper collaboration problem. submitted to Computers and Operations Research Odysseus 2003 Special Issue, 2003.
4. Ozlem Ergun, Gultekin Kuyzu, and Martin Savelsbergh. The lane covering problem. manuscript, 2003.
5. U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45:634–652, 1998.
6. M. R. Garey and D.S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, 1979.
7. O. Goldschmidt, D. Hochbaum, C. Hurkens, and G. Yu. Approximation algorithms for the k-clique covering problem. *SIAM Journal of Discrete Math.*, 9(3):492–509, 1996.
8. M. Guan. Graphic programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.
9. D. Hochbaum and E. Olinick. The bounded cycle-cover problem. *INFORMS Journal on Computing*, 13(2):104–119, 2001.
10. Holyer. The np-completeness of some edge partitioning problems. *SIAM journal of Computing*, 10:713–717, 1981.
11. A. Itai, R.J. Lipton, C.H. Papadimitriou, and M. Rodeh. Covering graphs by simple circuits. *SIAM Journal on Computing*, 10:746–750, 1981.
12. K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM*, 50(6):795–824, November 2003.
13. J. Kennington, V. Nair, and M. Rahman. Optimization based algorithms for finding minimal cost ring covers in survivable networks. *Computational Optimization and Applications*, 14(2):219–230, 1999.
14. F. Lazebnik and V.A. Ustimenko. Explicit construction of graphs with arbitrary large girth and of large size. *Discrete Applied Mathematics*, 60:275–284, 1995.
15. Christos H. Papadimitriou. On the complexity of edge traversing. *Journal of the ACM*, 23(3):544–554, 1976.
16. B. Rachavachari and J. Veerasamy. A 3/2-approximation algorithm for the mixed postman problem. *SIAM journal of Disc. Math.*, 12:425–433, 1999.
17. J.B. Slevinsky, W.D. Grover, and M.H. MacGregor‘. An algorithm for survivable network design employing multiple self-healing rings. In *GLOBECOM ’93*, pages 1568–1573, 1993.
18. C. Thomassen. On the complexity of finding a minimum cycle cover of a graph. *SIAM journal of computing*, 26:675–6777, 1997.
19. V. V. Vazirani. *Approximation Algorithms.* Springer-Verlag, Berlin, 2001.
20. C.Q. Zhang. *Integer flows and cycle cover of graphs.* Marcel Dekker, Inc, 1997.

A Polynomial Time Algorithm for Minimum Cycle Basis in Directed Graphs

Telikepalli Kavitha and Kurt Mehlhorn*

Max-Planck-Institut für Informatik,
Saarbrücken, Germany
{kavitha, mehlhorn}@mpi-sb.mpg.de

Abstract. We consider the problem of computing a minimum cycle basis in a directed graph G with m arcs and n vertices. The arcs of G have non-negative weights assigned to them. We give an $\tilde{O}(m^4n)$ algorithm, which is the first polynomial time algorithm for this problem. We also present an $\tilde{O}(m^3n)$ randomized algorithm. The problem of computing a minimum cycle basis in an *undirected* graph has been well-studied. However, it is not known if an efficient algorithm for undirected graphs automatically translates to an efficient algorithm for directed graphs.

1 Introduction

1.1 The Problem

Let $G = (V, A)$ be a directed graph with vertex set V and arc set A (no self-loops). We will consider cycles in the underlying undirected graph and assign each such cycle C a vector in $\{-1, 0, 1\}^{|A|}$. This incidence vector, also called C, is defined as follows. For each arc $a \in A$

$$C(a) = \begin{cases} 1 & \text{if } C \text{ traverses } a \text{ in forward direction} \\ -1 & \text{if } C \text{ traverses } a \text{ in backward direction} \\ 0 & \text{if } a \notin C \end{cases}$$

The *cycle space* of G is the vector space over $\mathbb{Q}$ that is spanned by these incidence vectors. The cycle space of a connected digraph has dimension $d = m - n + 1$, where $|A| = m$ and $|V| = n$. A *cycle basis* of G is a set of cycles $C_1, ..., C_d$ whose incidence vectors permit a unique linear combination of the incidence vector of any cycle of G.

We assume that there is a weight function $w : A \to \mathbb{R}^+$, i.e., the arcs of G have non-negative weights assigned to them. The weight of a cycle is the sum of the weights of its arcs. The weight of a cycle basis is the sum of the weights of its cycles. A *minimum cycle basis* of G is a cycle basis with minimum weight, that is, a cycle basis $\mathcal{B}$ such that $\sum_{C \in \mathcal{B}} \sum_{a \in C} w(a)$ is minimum. We consider the problem of computing a minimum cycle basis in a given digraph.

* Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 654–665, 2005.

1.2 Background

The importance of the problem of computing a minimum cycle basis lies in its use as a preprocessing step in several algorithms. That is, a cycle basis is generally not wanted for its own sake, but to be used as an input for a later algorithm. And the importance of a minimum cycle basis is to reduce the amount of work that has to be done by this later algorithm. In the problem of computing a minimum cycle basis of an undirected graph $U = (N, E)$, with each cycle we associate a $\{0, 1\}$ vector x, indexed on E, where $x_e = 1$ if e is an edge of C, $x_e = 0$ otherwise. The vector space over GF(2) generated by these vectors is called the *cycle space* of U. A minimum cycle basis of U is a set of linearly independent (over GF(2)) cycles that span the cycle space of U and whose sum of weights is minimum. The problem of computing a minimum cycle basis in undirected graphs has been well-studied [2, 5, 7, 8, 9, 10] and the current fastest algorithm for computing a minimum cycle basis in an undirected graph with m edges and n vertices runs in $O(m^2n + mn^2 \log n)$ time [10].

In many cases the network graphs of interest are intrinsically directed. For a directed graph G, we obtain the underlying undirected graph of G by removing the directions from the arcs. A set of cycles $C_1, ..., C_d$ of G projects onto an undirected cycle basis, if by removing the orientations of the arcs in the cycles, we obtain a cycle basis for the underlying undirected graph. It was shown by Liebchen and Peeters in [11] that if $\mathcal{C} = \{C_1, ..., C_d\}$ is a set of cycles in a directed graph G that projects onto an undirected cycle basis, then $\mathcal{C}$ is a cycle basis of G. But the the converse is not true. Similarly, a minimum cycle basis of a digraph need not project onto a cycle basis of the underlying undirected graph. The books by Deo [6] and Bollobás [3] have an in-depth coverage of the subject of cycle bases.

Our Results. In this paper we give an an $\tilde{O}(m^3n)$ randomized algorithm and an $\tilde{O}(m^4n)$ deterministic algorithm to compute a minimum cycle basis in a digraph $G = (V, A)$ where $|A| = m$ and $|V| = n$. Very recently, Liebchen and Rizzi [12] have also given an $\tilde{O}(m^4n)$ deterministic algorithm to compute a minimum cycle basis in a directed graph. They adapt Horton's greedy approach [9] and using fast matrix multiplication, their algorithm can be implemented in $\tilde{O}(m^{\omega+1}n)$ time, where ω is the best exponent of matrix multiplication. Our approach is complementary to theirs. Our algorithms use simple linear algebra and elementary number theory and are in the domain of arithmetical algorithms. The techniques used here might be of independent interest.

2 The Algorithm

Our algorithm is broadly based on the approach used in [5, 2, 10] for computing a minimum cycle basis in an undirected graph. The basic idea is to have an iterative algorithm that computes a new cycle C_i of the minimum cycle basis in the i-th iteration. There is no loss of generality in assuming that the underlying undirected graph of G is connected. Then $d = m - n + 1$ is the dimension of the cycle space of G. We can assume that $m \geq 2$.

2.1 The Basic Idea

Recall that each cycle in G is encoded as a $\{-1, 0, 1\}$ vector in $\mathbb{Q}^m$. Let $\langle S, C\rangle = \sum_{i=1}^{m} s_i c_i$ denote the standard inner product between $S = (s_1, ..., s_m)$ and $C = (c_1, ..., c_m)$, which are vectors in the space $\mathbb{Q}^m$. A high-level description of our algorithm is as follows.

For $i = 1, ..., d$ do:

1. let $S_i \in \mathbb{Q}^m$ be a non-zero vector such that $\langle S_i, C_j\rangle = 0$ for all j where $j < i$.
2. compute C_i to be a shortest cycle in G such that $\langle S_i, C_i\rangle \neq 0$.

That is, S_i is a non-zero vector orthogonal to the cycles computed in the first $i-1$ iterations. And the shortest cycle which is *not* orthogonal to S_i is C_i. Before we get into the details of how to implement these steps, let us first check if this approach gives us what we seek.

Theorem 1. *The set $\{C_1, \ldots, C_d\}$ is a minimum cycle basis of G.*

Proof. It is easy to see that C_i is linearly independent of $\{C_1, ..., C_{i-1}\}$. S_i is a witness of this linear independence since $\langle S_i, C_j\rangle = 0$ for all $j < i$, so the inner product of S_i with any linear combination of $C_1, \ldots, C_{i-1}$ has to be zero but $\langle S_i, C_i\rangle \neq 0$. Hence the whole set $\{C_1, \ldots, C_d\}$ is linearly independent.

Suppose $\{C_1, \ldots, C_d\}$ does not form a minimum cycle basis. Then there exists a minimal i such that $\{C_1, \ldots, C_i\} \not\subseteq$ any minimum cycle basis. So $\{C_1, \ldots, C_{i-1}\} \subseteq$ some minimum cycle basis $\mathcal{B}$. Then

$$C_i = \lambda_1 B_1 + \lambda_2 B_2 + \cdots + \lambda_l B_l \text{ where each } \lambda_t \in \mathbb{Q} \text{ and each } \lambda_t \neq 0,$$

for some $\{B_1, \ldots, B_l\} \subseteq \mathcal{B}$. Since $\langle S_i, C_i\rangle \neq 0, \exists B_k \in \{B_1, ..., B_l\}$ such that $\langle S_i, B_k\rangle \neq 0$. Then by the very definition of C_i, it follows that weight$(B_k) \geq$ weight(C_i). Hence $\mathcal{B}' = B \cup \{C_i\} \setminus \{B_k\}$ is also a minimum cycle basis. The cycle B_k that has been omitted from $\mathcal{B}'$ cannot be one of $C_1, ..., C_{i-1}$ since the inner product of each of $C_1, ..., C_{i-1}$ with S_i is zero whereas $\langle S_i, B_k\rangle \neq 0$. Hence, $\{C_1, ..., C_i\} \subseteq \mathcal{B}'$, which is a minimum cycle basis - a contradiction. □

So our basic idea works. Let us now consider how to implement the two steps in the basic idea.

2.2 Implementation

Computing a shortest cycle C_i such that $\langle S_i, C_i\rangle \neq 0$ for $S_i \in \mathbb{Q}^m$ can be reduced to computing a shortest cycle C_i such that $\langle S_i, C_i\rangle \neq 0$ for $S_i \in \mathbb{Z}^m$. So let us look at the following implementation. More specifically, in the i-th iteration:

Step 1. Compute $S_i \in \mathbb{Z}^m$ such that S_i is a nontrivial solution to the set of equations:

$$\langle \boldsymbol{x}, C_j\rangle = 0 \ \forall j < i.$$

We will show that we can find an S_i with $\|S_i\|_\infty \leq 2^{f(i)}$, where $f(i)$ is $O(i \log i)$.

Step 2. Compute $f(i)+1$ distinct primes $p_0, ..., p_{f(i)}$, where each $p_t \geq m$.
For $t = 0, ..., f(i)$ do:
- compute a shortest cycle B_t such that $\langle S_i, B_t \rangle \neq 0 \pmod{p_t}$.
- Now we have a list (probably, a multiset) of cycles $(B_0, \ldots, B_{f(i)})$.

$$C_i := \min(B_0, \ldots, B_{f(i)}).$$

That is, C_i is assigned to be that cycle which has the least weight in this list. If there is more than one cycle with the same least weight, then C_i can be any one of such cycles.

Lemma 1. *C_i is a shortest cycle in G such that $\langle S_i, C_i \rangle \neq 0$.*

Proof. Suppose there is a shorter cycle D_i. Since D_i is not in the list$(B_0, ..., B_{f(i)})$, it must be the case that $\langle S_i, D_i \rangle = 0 \pmod{p_0}, \pmod{p_1}, \cdots, \pmod{p_{f(i)}}$. This forces $\langle S_i, D_i \rangle$ to be a multiple of $\Pi_t p_t$ since all the p_t's are distinct primes. Since each $p_t \geq m$, $\Pi_t p_t > m^{f(i)+1}$.

Since $\|S_i\|_\infty \leq 2^{f(i)}$ and D_i is a vector in $\{-1, 0, 1\}^m$, we have

$$|\langle S_i, D_i \rangle| \leq m 2^{f(i)} \leq m^{f(i)+1} < \Pi_t p_t.$$

So the only way $\langle S_i, D_i \rangle$ can be a multiple of $\Pi_t p_t$ is that $\langle S_i, D_i \rangle = 0$.

Hence, any cycle D_i with a lesser weight than C_i necessarily has to obey $\langle S_i, D_i \rangle = 0$. □

A question that needs to be answered is why should there always be some cycle C_i such that $\langle C_i, S_i \rangle \neq 0$. We will show that the S_i that we compute has the property that such a cycle always exists.

2.3 Computing S_i

Let us first order the arcs in the arc set A so that $a_{d+1}, ..., a_m$ form the edges of a spanning tree T of the underlying undirected graph. This means that in the incidence vector representation of cycles, the first d coordinates correspond to arcs $a_1, \ldots, a_d$ which are outside the tree T and the last $n-1$ coordinates are the arcs of T.

This will enable us to maintain the invariant that each S_i is of the form $(s_{i1}, \ldots, s_{ii}, 0, \ldots, 0)$ with $s_{ii} \neq 0$. So only the first i coordinates of S_i can be non-zero and s_{ii} has to be non-zero. The fundamental cycle F_i formed by the adding the arc a_i to the edges of the spanning tree T has the incidence vector $(0, \ldots, 0, 1, 0, \ldots, 0, *, \ldots, *)$. That is, in the first d coordinates only $F_i(a_i) \neq 0$ and the $*$'s, which take $\{-1, 0, 1\}$ values, are in the last $n-1$ coordinates. $\langle F_i, S_i \rangle = s_{ii} \neq 0$. Hence, there is always at least one cycle whose inner product with S_i is non-zero.

In the first iteration, S_1 is any non-zero vector. So we assign S_1 to be the vector $(1, 0, \ldots, 0)$. Thus S_1 satisfies our invariant. In the i-th iteration we need to find a nontrivial solution to the set of equations $\langle \boldsymbol{x}, C_j \rangle = 0 \ \forall j < i$. We do this as follows.

– compute a vector $(r_1, ..., r_{i-1}, 1, 0, ...0) \in \mathbb{Q}^m$ that is orthogonal to C_j for each $j < i$.

Let the j-th cycle C_j have the incidence vector $(c_{j1}, ..., c_{jm})$. Since the vector $(r_1, ..., r_{i-1}, 1, 0, ...0)$ is orthogonal to C_j,

$$\sum_{k=1}^{i-1} c_{jk} r_k = -c_{ji}, \text{ for } 1 \leq j \leq i-1.$$

Let $\tilde{C}_j = (c_{j1}, ...c_{j(i-1)})$ be the restriction of C_j to its first $i-1$ coordinates. So $(r_1, ..., r_{i-1})$ is a solution to the set of equations:

$$\tilde{C}_j \cdot (x_1, ..., x_{i-1}) = -c_{ji} \text{ for } j = 1, ..., i-1.$$

A solution always exists to the above set of equations because $\tilde{C}_1, \ldots, \tilde{C}_{i-1}$ are linearly independent. Suppose the linear combination

$$\sum_{j=1}^{i-1} \alpha_j \tilde{C}_j = \mathbf{0} \tag{1}$$

and not all α_j are 0. Then consider the largest k such that $\alpha_k \neq 0$ and take the inner product of both sides of Equation (1) with that $\tilde{S}_k$, where $\tilde{S}_k$ is the restriction of the vector S_k to its first $i-1$ coordinates. (Note that $\tilde{S}_k$ has all the non-zero entries of S_k for each $1 \leq k \leq i-1$. So $\langle \tilde{C}_j, \tilde{S}_k \rangle = \langle C_j, S_k \rangle$.) Then the left hand side is $\sum_{j=1}^{k} \alpha_j \langle C_j, S_k \rangle = \alpha_k \langle C_k, S_k \rangle$ since $\langle C_j, S_k \rangle = 0$ for all $j < k$. Since α_k and $\langle C_k, S_k \rangle$ are non-zero while the right hand side is zero, we get a contradiction. Hence each $\alpha_j = 0$ for $1 \leq j \leq i-1$.

Thus the $(i-1) \times (i-1)$ matrix of $\tilde{C}$'s which has $\tilde{C}_1, \ldots, \tilde{C}_{i-1}$ as its rows is invertible and so there exists a unique solution to the set of equations:

$$\begin{pmatrix} \tilde{C}_1^T \\ \vdots \\ \tilde{C}_{i-1}^T \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} -c_{1i} \\ \vdots \\ -c_{(i-1)i} \end{pmatrix} \tag{2}$$

Let $(r_1, \ldots, r_{i-1})$ be the solution to the above set of equations. Then $S'_i = (r_1, \ldots, r_{i-1}, 1, 0, ..., 0)$ is a vector in $\mathbb{Q}^m$ that is orthogonal to $C_1, ..., C_{i-1}$.

By Cramer's rule, each r_j is of the form $r_j = y_j/k$, where k is the determinant of the matrix of $\tilde{C}$'s (call this matrix $\mathcal{M}_i$) and y_j is the determinant of the matrix obtained by replacing the j-th column of $\mathcal{M}_i$ by the vector on the right hand side of Equation (2). In order to get an integral vector S_i from S'_i, we multiply S'_i with k. So $S_i = kS'_i = (y_1, ..., y_{i-1}, k, 0, ..., 0)$ is an integral vector that is orthogonal to all the cycles $C_1, ..., C_{i-1}$. And we have also maintained our invariant that S_i has non-zero entries in only its first i coordinates and its i-th coordinate is non-zero. Equivalently, $(y_1, ..., y_{i-1})$ is the (integral) solution to the set of equations:

$$\tilde{C}_j \cdot \boldsymbol{x} = -kc_{ji} \text{ for } j = 1, ..., i-1. \tag{3}$$

Let us now bound the L_∞ norm of S_i. Since k is the determinant of an $(i-1)\times(i-1)$ matrix whose entries are $-1, 0$ or 1, using Hadamard's inequality we get

$$|k| \leq \Pi_{j=1}^{i-1}\|\tilde{C}_j\| \leq \Pi_{j=1}^{i-1}\sqrt{i} \leq 2^{(i\log i)/2}.$$

Similarly, each $|y_j| \leq 2^{(i\log i)/2}$. Hence $\max\{y_1,\ldots,y_j,\ldots,k\} \leq 2^{(i\log i)/2}$. Thus we have shown that $\|S_i\|_\infty \leq 2^{f(i)}$, where $f(i) = (i\log i)/2$.

The vector $(y_1,\ldots,y_{i-1})$ can be obtained by Gaussian elimination, or by multiplying the matrix $\mathcal{M}_i^{-1}$ with the column vector $(-kc_{1i},\ldots,-kc_{(i-1)i})$. These computations can be implemented in $O(i^\omega)$ steps, where ω is the best exponent of matrix multiplication. We also need to account for the cost of performing arithmetic operations, since we do arithmetic on large numbers. Assuming that arithmetic on $O(\ell\log\ell)$ bits takes $O(\ell)$ time, we have the following lemma.

Lemma 2. *A nontrivial vector $S_i \in \mathbb{Z}^m$ such that $\langle S_i, C_j\rangle = 0\ \forall j < i$ and $\|S_i\|_\infty \leq 2^{f(i)}$ can be computed in $O(m^{\omega+1})$ time.*

3 Computing B_t

In order to compute a shortest cycle whose inner product with S_i is non-zero modulo p_t, we build an undirected graph $U_{i,t}$ using the given directed graph G, the vector S_i and the number p_t. The graph $U_{i,t}$ can be visualised as p_t levels of the digraph G. Call these levels as level 0,..., level (p_t-1). Each level has a copy of every vertex $v \in V$. Let v_j be the copy of vertex v in level j. The edge set of $U_{i,t}$ also consists of p_t copies of each arc $a \in A$. The edges corresponding to arc $a = (u,v)$ are (u_j, v_k) where $k = (j + S_i(a))$ modulo p_t for each $j = 0, 1, \ldots, p_t - 1$.

That is, the edge in $U_{i,t}$ that corresponds to copy j (for $j = 0,\ldots,p_t-1$) of the arc $a = (u,v)$ of G goes from the copy of vertex u in level j to the copy of vertex v in level $(j + S_i(a))$ mod p_t. Also, each edge (u_j, v_k) in $U_{i,t}$ inherits the weight of its corresponding arc (u,v) of G.

Thus there is a well-defined map from the vertex set of $U_{i,t}$ to the vertex set V of G and from the edge set of $U_{i,t}$ to the arc set A of G. We can extend this map to paths of $U_{i,t}$. So given any path q in $U_{i,t}$, we can map q to a *chain*[1] in G by mapping the vertices and edges of q to their images in G.

Lemma 3 captures the essence of the graph $U_{i,t}$ and Lemma 4 gives us an efficient way of computing the desired cycle. These lemmas are simple to show and their proofs will be included in the full version of the paper.

Lemma 3. *Any (v_r, v_s) path in $U_{i,t}$, whose edges map to distinct arcs of G, maps to a cycle C in G. The incidence vector of such a cycle C satisfies $\langle C, S_i\rangle = \pm(s-r) (\mathrm{mod} p_t)$.*

[1] A chain is an alternating sequence of vertices and arcs $(x_0, a_1, x_1, a_2, ..., a_r, x_r)$ such that either $a_k = (x_{k-1}, x_k)$ or $a_k = (x_k, x_{k-1})$.

Relabel the arcs of G so that $a_{d+1}, \ldots, a_m$ are the arcs of a spanning tree of the underlying undirected graph.
Compute distinct primes $p_0, \ldots, p_{f(m)}$, where each prime $\geq m$. {This can be done by a sieving algorithm.}
for $i = 1, \ldots, d$ **do**
  Compute $S_i = (s_{i1}, \ldots, s_{ii}, 0, \ldots, 0) \in \mathbb{Z}^m$ such that $s_{ii} \neq 0$ and $\langle S_i, C_j \rangle = 0$ for all $j < i$.
  for $t = 0, \ldots, f(i)$ **do**
    Compute the graph $U_{i,t}$ from G using S_i and p_t (as described in Section 3).
    Let $q = \min_v \min_{\ell \neq 0}$ shortest (v_0, v_ℓ) path in $U_{i,t}$. Let B_t be the cycle in G that the path q corresponds to.
  end for
  $C_i = \min(B_0, \ldots, B_{f(i)})$.
end for
Return $\{C_1, \ldots, C_d\}$.

Fig. 1. Algorithm-MCB: Algorithm to compute a minimum cycle basis in a digraph

Lemma 4. *Let $q = \min_v \min_{\ell \neq 0}$ shortest (v_0, v_ℓ) path[2] in the graph $U_{i,t}$. Then q corresponds to a shortest cycle B_t in G such that $\langle B_t, S_i \rangle \neq 0 \pmod{p_t}$.*

Remark. Whenever $S_i \bmod p_t$ is not the zero vector, then there is always a path in $U_{i,t}$ between v_0 and v_ℓ for some $v \in V$ and $\ell \neq 0$. If $S_i \bmod p_t$ is the zero vector, then q does not exist and so there would be no cycle B_t in the list $(B_0, \ldots, B_{f(i)})$. Indeed, there can be no cycle in G whose inner product with S_i is non-zero modulo p_t, given that $S_i \bmod p_t$ is the zero vector.

Cost of Computing B_t. Computation of the path q can be accomplished by a shortest paths computation in the graph $U_{i,t}$ from each vertex v_0 in level 0 and taking the shortest $(v_0, v_\ell), \ell \neq 0$ path over all $v \in V$. This can be done in $O(n(p_t m + p_t n \log p_t n))$ time since one single-source shortest paths computation in $U_{i,t}$ would take $O(p_t m + p_t n \log p_t n)$ time by Dijkstra's algorithm.

The value of $\pi(r)$, the number of primes less than r, is given by $r/6 \log r \leq \pi(r) \leq 8r/\log r$ [1]. So each of the primes p_t can be bounded from above by $O(f(m) \log m)$. Hence we have shown the following lemma.

Lemma 5. *We can compute a shortest cycle B_t such that $\langle B_t, S_i \rangle \neq 0 \pmod{p_t}$ in $\tilde{O}(nmf(m))$ time.*

3.1 The Entire Algorithm

A summary of our algorithm to compute a minimum cycle basis in $G = (V, A)$ is given in Fig. 1. The correctness of the algorithm follows from Lemmas 1, 3, 4 and Theorem 1. Lemmas 2 and 5 ensure polynomial running time of the algorithm.

[2] In case of many paths tied for the minimum, choose q to be any of these paths that has the least number of edges.

Running Time. Recall that the cost of computing S_i is $O(m^{\omega+1})$ (by Lemma 2). This is $o(m^3 n)$ since $\omega < 2.376$ [4]. The limiting factor in the running time of the i-th iteration is the computation of the cycles $B_0, \ldots, B_{f(i)}$. Since each of them can be computed in $\tilde{O}(nmf(m))$ time (by Lemma 5), the time required to compute C_i is $\tilde{O}(nmf(m)f(i))$. Since $f(i)$ is $O(i \log i)$, the i-th iteration takes $\tilde{O}(m^3 n)$ time. Thus Theorem 2 immediately follows.

Theorem 2. *Algorithm-MCB computes a minimum cycle basis of G in $\tilde{O}(m^4 n)$ time.*

4 An $\widetilde{O}(m^3 n)$ Randomized Algorithm

In this section we present a Monte Carlo algorithm to compute a minimum cycle basis in G. The underlying ideas are the same as in Algorithm-MCB, except that we will use the primes $p_0, \ldots, p_{f(m)}$ more sparingly now.

Let us call a prime $p \in \{p_0, \ldots, p_{f(i)}\}$ a *witness* of C_i if $\langle S_i, C_i \rangle \neq 0 \pmod{p}$. Lemma 1 shows that since $\|S_i\|_\infty \leq 2^{f(i)}$, there is at least one witness of C_i in $\{p_0, \ldots, p_{f(i)}\}$. We can easily extend this idea to get the following lemma.

Lemma 6. *If $p_1, \ldots, p_{2f(i)}$ are distinct primes where each prime $\geq m$, then C_i has at least $f(i)$ witnesses in $\{p_1, \ldots, p_{2f(i)}\}$.*

So a prime p chosen uniformly at random from $\{p_1, \ldots, p_{2f(i)}\}$ has probability $\geq 1/2$ of being a witness of C_i. So instead of computing $f(i)+1$ cycles $B_0, \ldots, B_{f(i)}$ and taking their minimum as C_i, we could first sample a few primes with uniform distribution from $\{p_0, \ldots, p_{2f(i)}\}$ and compute the cycles corresponding only to these few sampled primes. We will call the minimum of these cycles as C_i. If the number of sampled primes is $\text{poly}(\log m)$, then we spend only $\tilde{O}(m^2 n)$ time to compute C_i now. But of course, we have introduced some error. So this C_i need not always be the cycle that we seek, however we can bound the error probability.

The more difficult problem is to efficiently compute $S_i = (s_{i1}, \ldots, s_{ii}, 0, \ldots, 0)$ in $\mathbb{Z}^m$ where $s_{ii} \neq 0$ and $\langle S_i, C_j \rangle = 0$ for all $j < i$. We can no longer afford to spend $\Theta(m^{\omega+1})$ time to compute S_i now.

4.1 Computing S_i More Efficiently

The important observation is that we do not really need S_i, what we need is $S_i \bmod p_t$, that is, the vector $(s_{i1} \bmod p_t, \ldots, s_{ii} \bmod p_t, 0, \ldots, 0)$ in order to compute B_t. If $q_0, \ldots, q_r$ are the few sampled primes of iteration i, then $S_i \bmod q_0, \ldots, S_i \bmod q_r$ are the vectors that we need. We had computed S_i as $(y_1, \ldots, y_{i-1}, k, 0, \ldots, 0)$ where $(y_1, ..., y_{i-1})$ is the (integral) solution to the set of equations:

$$\tilde{C}_j \cdot \boldsymbol{x} = -kc_{ji} \text{ for } j = 1, ..., i-1$$

(this is Equation (3) from Section 2.3). The integer $k = \det(\mathcal{M}_i)$, where $\mathcal{M}_i$ denotes the matrix of $\tilde{C}$'s on the left of the above equation. Now we want

- Compute $3f(m)$ distinct primes $p_1, \ldots, p_{3f(m)}$, where each prime $\geq m$.
- For $i = 1, \ldots d$ do:
  1. initialize $Q = \emptyset$.
  2. For $j = 1, \ldots, \lfloor \log^2 m \rfloor$ do:
     - let r_j be a random element of $\{p_1, \ldots, p_{3f(m)}\} \setminus \{r_1, \ldots, r_{j-1}\}$ picked with uniform distribution.
     - if $\det(\mathcal{M}_i) \neq 0 \pmod{r_j}$, then $Q = Q \cup r_j$.
       ($\mathcal{M}_i$ is the $(i-1) \times (i-1)$ matrix discussed above. $\mathcal{M}_1$ is the empty matrix; let its determinant be ∞.)
  3. if $|Q| \leq \lceil \log m \rceil$, then declare *failure* and exit the program.
  4. For each $q_t \in Q$
     - compute in the field $\mathbb{Z}_{q_t}$: $\ell_i = \det(\mathcal{M}_i)$ and $(\ell_1, \ldots, \ell_{i-1}) = \mathcal{M}_i^{-1} b$.
     - set $S_i \bmod q_t = (\ell_1, \ldots, \ell_i, 0, \ldots, 0)$. (And when $i = 1$, we set $S_1 \bmod q_t = (1, 0, \ldots, 0)$.)
     - compute $B_t =$ shortest cycle such that $\langle B_t, S_i \bmod q_t \rangle \neq 0 \pmod{q_t}$.
  5. $C_i = \min(B_1, \ldots, B_{|Q|})$.
- Return $\{C_1, \ldots, C_d\}$.

Fig. 2. Randomized-MCB: Randomized algorithm to compute a minimum cycle basis in a digraph

to determine $S_i \bmod p$ directly, where p is a sampled prime. Let $S_i \bmod p = (\ell_1, \ldots, \ell_i, 0, \ldots, 0)$. It follows from Equation (3) that $(\ell_1, \ldots, \ell_{i-1})$ satisfies the set of equations:

$$\tilde{C}_j \cdot \boldsymbol{x} = -\ell_i c_{ji} \quad \text{for } j = 1, ..., i-1 \quad \text{in the field} \quad \mathbb{Z}_p \tag{4}$$

where $\ell_i = \det(\mathcal{M}_i) (\mathrm{mod} p)$. Whenever $\ell_i \neq 0$, $\mathcal{M}_i^{-1} b \pmod{p}$ is the unique solution of $\mathcal{M}_i \boldsymbol{x} = b$ in $\mathbb{Z}_p$, where b denotes the column vector of $-\ell_i c_{ji}$'s of Equation (4). Then we can determine $S_i \bmod p$ directly by computing $\det(\mathcal{M}_i)$ and $\mathcal{M}_i^{-1} b$ in the field $\mathbb{Z}_p$.

We know that $\det(\mathcal{M}_i) \neq 0$ and that $|\det(\mathcal{M}_i)| \leq 2^{f(i)}$ from Hadamard's inequality (see Section 2.3). So, by exactly the same argument as in the proof of Lemma 1, we can show the following lemma.

Lemma 7. *If $p_1, \ldots, p_{f(i)}$ are distinct primes, then for at least one prime p in $\{p_1, \ldots, p_{f(i)}\}$, $\det(\mathcal{M}_i) \neq 0 (\mathrm{mod} p)$.*

Call such a prime p a *witness* of $\mathcal{M}_i$. Again we can extend this argument as in Lemma 6 to show that if $p_1, \ldots, p_{3f(m)}$ are distinct primes, then $\mathcal{M}_i$ has at most $f(i) - 1$ *non-witnesses* in $\{p_1, \ldots, p_{3f(m)}\}$. So if we take a sample P of $\lfloor \log^2 m \rfloor$ elements from $\{p_1, \ldots, p_{3f(m)}\}$, each choice made uniformly at random (without replacement), then we can show that with high probability, there are at least $\lceil \log m \rceil$ witnesses for $\mathcal{M}_i$ in P. For each of these witnesses p, we can compute $S_i \bmod p$ in $O(m^\omega)$ time because arithmetic in $\mathbb{Z}_p$ takes $O(1)$ time. So the total time spent to compute $S_i \bmod p$ for all the elements in P is $O(m^\omega |P|)$ or $\tilde{O}(m^\omega)$. This is $o(m^2 n)$.

Based on these ideas, we have the Monte Carlo algorithm presented in Fig. 2. In some runs Randomized-MCB declares "failure" and does not return any set of cycles. In other runs it returns a set of cycles $\{C_1, \ldots, C_d\}$. It is easy to see that these cycles are always linearly independent, but they may not always be a minimum cycle basis. Call iteration i a "success" if in iteration i, the cycle C_i that Randomized-MCB computes is indeed a shortest cycle whose inner product with S_i is non-zero. Let A_i denote the event that iteration i is a success. When the event $A_1 \cap \cdots \cap A_d$ occurs, then Randomized-MCB remains faithful to the basic idea (Section 2.1) and so the cycle basis $\{C_1, \ldots, C_d\}$ computed by the algorithm is indeed a minimum cycle basis. So the probability that Randomized-MCB outputs a minimum cycle basis is $\Pr(A_1 \cap \cdots \cap A_d)$.

Lemma 8. *For each* $1 \leq i \leq d$, $\Pr[A_i | (A_1 \cap \cdots \cap A_{i-1})] \geq 1 - 1/m$.

Proof. For iteration i to begin, it must be the case that in the first $i-1$ iterations, the algorithm did not declare "failure" and exit the program. The event $A_1 \cap \cdots \cap A_{i-1}$ ensures that this is indeed the case. So iteration i begins and iteration i is a success whenever the random sample of $\lfloor \log^2 m \rfloor$ primes, that we pick in iteration i, has at least $\lceil \log m \rceil + 1$ witnesses of $\mathcal{M}_i$ and among these witnesses of $\mathcal{M}_i$, there is at least one witness of C_i. We can bound from below the probability that this event occurs by upper bounding the complement event. The complement is the union of two events: (i) the event that $|Q| \leq \lceil \log m \rceil$ and (ii) the event that Q has no witnesses of C_i given that $|Q| \geq \lceil \log m \rceil + 1$.

Let us look at the first of these two events. We know that there are at most $f(i) - 1$ non-witnesses of $\mathcal{M}_i$ in $\{p_1, \ldots, p_{3f(m)}\}$. In iteration i, when we pick the first random element r_1, the probability that r_1 is not a witness of $\mathcal{M}_i$ is at most $(f(i) - 1)/(3f(m))$. The j-th random element r_j is chosen uniformly at random from $\{p_1, \ldots, p_{3f(m)}\} \setminus \{r_1, \ldots, r_{j-1}\}$. The probability that r_j is a witness of $\mathcal{M}_i$ depends on how many of $\{r_1, \ldots, r_{j-1}\}$ are witnesses of $\mathcal{M}_i$. But we do not need this exact value. We can easily see that for each $j = 1, \ldots, \lfloor \log^2 m \rfloor$

$$\Pr[r_j \text{ is not a witness for } \mathcal{M}_i] \leq \frac{f(i) - 1}{3f(m) - j + 1} \leq \frac{m}{3m - \log m} \leq \frac{1}{2}$$

So the probability that there are exactly $\lceil \log m \rceil$ witnesses of $\mathcal{M}_i$ in Q is at most $\binom{\log^2 m}{\lceil \log m \rceil} \cdot (1/2)^{\lfloor \log^2 m \rfloor - \lceil \log m \rceil}$. Hence the probability that there are at most $\lceil \log m \rceil$ witnesses of $\mathcal{M}_i$ in Q is upper bounded by $\lceil \log m \rceil + 1$ times this number. This value is at most

$$(\lceil \log m \rceil + 1)(\log^2 m)^{\lceil \log m \rceil} \frac{4}{m^{\log m - 1}} < \frac{1}{2m} \quad \forall m \geq \text{ some constant } m_0.$$

(Also, we will modify Randomized-MCB so that for small m, i.e. when $m < m_0$, we do no random sampling - so Randomized-MCB is identical to our deterministic algorithm for small m.)

Let us now look at the second event that contributes to the failure of iteration i. Given that $|Q| > \lceil \log m \rceil$, we would like to upper bound the probability that Q has no witnesses of C_i. There are at least $3f(m) - f(i) \geq 2f(m)$ witnesses of

C_i in $\{p_1, \ldots, p_{3f(m)}\}$. $\mathcal{M}_i$ also has at least $2f(m)$ witnesses in $\{p_1, \ldots, p_{3f(m)}\}$. So at least half the witnesses of $\mathcal{M}_i$ are also witnesses of C_i. So the probability that a random subset Q of witnesses of $\mathcal{M}_i$ contains no witness of C_i is at most $(1/2)^{|Q|} \leq 1/2m$ since $|Q| \geq \log m + 1$.

So the total error probability is at most $1/2m + 1/2m = 1/m$. Hence $\Pr[A_i|(A_1 \cap \cdots \cap A_{i-1})] \geq 1 - 1/m$, for each $1 \leq i \leq d$. □

Since $\Pr(A_1 \cap \cdots \cap A_d) = \Pi_{i=0}^{d} \Pr[A_i|(A_1 \cap \cdots \cap A_{i-1})]$, Lemma 8 shows that the success probability of Randomized-MCB is at least $(1 - 1/m)^d > (1 - 1/m)^m \approx 1/e$. Hence, by running Randomized-MCB a constant number of times and taking the cycle basis whose weight is the least, we can make the error probability less than δ for any given constant $\delta > 0$. The running time of the algorithm follows from the discussion at the beginning of Section 4. Hence Theorem 3 follows.

Theorem 3. *A minimum cycle basis can be computed with high probability in* $\tilde{O}(m^3 n)$ *time.*

5 Further Analysis

In this section we would like to prove that *any* minimum cycle basis from the set of all minimum cycle bases of G has a chance to be returned as $\{C_1, \ldots, C_d\}$ by a variant of Algorithm-MCB (Fig. 1). First, fix any spanning tree T of the underlying undirected graph of G and let $\{a_1, \ldots, a_d\}$ be the arcs of $G \setminus T$. Let $\{D_1, \ldots, D_d\}$ be some minimum cycle basis of G. Let us assume that these cycles are sorted by their weights. So we have $\text{weight}(D_1) \leq \cdots \leq \text{weight}(D_d)$. Let us form the $d \times d$ matrix $\mathcal{D}$ whose i-th column is the incidence vector of D_i restricted to the arcs $\{a_1, \ldots, a_d\}$. It is simple to show that $\mathcal{D}$ is a nonsingular matrix. The next observation is that the rows of $\mathcal{D}$ can be permuted so that for each i, the $i \times i$ submatrix consisting of the first i rows and first i columns is nonsingular. Alternately, the LUP decomposition of $\mathcal{D}$ gives us a permutation matrix P such that $P\mathcal{D}$ has this property. Permuting the rows of $\mathcal{D}$ is just renumbering the arcs $a_1, \ldots, a_d$.

Let us now describe the slight variation in Algorithm-MCB so that we can claim that $\{D_1, \ldots, D_d\}$ can be returned by our algorithm. The variation is that we do an extra step, right at the beginning, where we permute the order of the arcs in $G \setminus T$. That is, we generate a random permutation σ on $\{1, \ldots, d\}$ and the order of coordinates in the incidence vectors of cycles will be $a_{\sigma(1)}, \ldots, a_{\sigma(d)}, a_{d+1}, \ldots, a_m$. After this step, the rest of the algorithm is the same as Algorithm-MCB (Fig. 1). In the case of a tie while computing a shortest path or shortest cycle, let us assume that the algorithm breaks ties randomly so that each of the candidate cycles tied as the shortest cycles has a chance to be picked.

Our algorithm has a tree of possible executions and we want to show that there is at least one execution where $\{D_1, \ldots, D_d\}$ is computed as a minimum cycle basis. In the first step let us assume that the permutation π was generated,

where π is the permutation corresponding to the permutation matrix P in the LUP decomposition of $\mathcal{D}$. Using the special property of the $d \times d$ matrix $P\mathcal{D}$, that is, for each i, its leading $i \times i$ submatrix is nonsingular, we can show that for $1 \leq i \leq d$, there is a vector $L_i = (\ell_{i1}, \ldots, \ell_{ii}, 0, \ldots, 0)$ in $\mathbb{Q}^m$ such that (i) $\langle L_i, D_j \rangle = 0$ for all j where $j < i$ and (ii) D_i is a shortest cycle such that $\langle L_i, D_i \rangle \neq 0$.

The above statement is the crux of the argument and Theorem 4 follows directly from this. The details will be given in the full version of the paper.

Theorem 4. *The minimum cycle basis* $\{D_1, \ldots, D_d\}$ *can be returned by the modified Algorithm-MCB.*

Acknowledgment. We thank Jaikumar Radhakrishnan for his helpful comments.

References

1. T. M. Apostol. *Introduction to Analytic Number Theory.* Springer-Verlag, 1997.
2. F. Berger, P. Gritzmann, and S. de Vries. Minimum Cycle Bases for Network Graphs. *Algorithmica*, 40(1): 51-62, 2004.
3. B. Bollobás. *Modern Graph Theory* volume 184 of *Graduate Texts in Mathematics*, Springer, Berlin, 1998.
4. D. Coppersmith and S. Winograd. Matrix multiplications via arithmetic progressions. *Journal of Symb. Comput.*, 9:251–280, 1990.
5. J.C. de Pina. *Applications of Shortest Path Methods.* PhD thesis, University of Amsterdam, Netherlands, 1995.
6. N. Deo. *Graph Theory with Applications to Engineering and Computer Science* Prentice-Hall Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, 1982.
7. Alexander Golynski and Joseph D. Horton. A polynomial time algorithm to find the minimum cycle basis of a regular matroid. In *8th Scandinavian Workshop on Algorithm Theory*, 2002.
8. David Hartvigsen and Russell Mardon. The all-pairs min cut problem and the minimum cycle basis problem on planar graphs. *Journal of Discrete Mathematics*, 7(3):403–418, 1994.
9. J. D. Horton. A polynomial-time algorithm to find a shortest cycle basis of a graph. *SIAM Journal of Computing*, 16:359–366, 1987.
10. T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. A faster algorithm for Minimum Cycle Basis of graphs. *In Proc. of ICALP*, LNCS 3142: 846-857, 2004.
11. C. Liebchen and L. Peeters. On Cyclic Timetabling and Cycles in Graphs. Technical Report 761/2002, TU Berlin.
12. C. Liebchen and R. Rizzi. A Greedy Approach to compute a Minimum Cycle Basis of a Directed Graph. Technical Report 2004/31, TU Berlin.

All-Pairs Nearly 2-Approximate Shortest-Paths in $O(n^2 \operatorname{polylog} n)$ Time

Surender Baswana[1], Vishrut Goyal[2], and Sandeep Sen[3]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
sbaswana@mpi-sb.mpg.de
[2] Persistent Systems Private Limited, Pune, India
vishrut_goyal@persistent.co.in
[3] Department of Computer Science and Engineering,
I.I.T. Kharagpur, India
ssen@cse.iitkgp.ernet.in

Abstract. Let $G(V, E)$ be an unweighted undirected graph on $|V| = n$ vertices. Let $\delta(u, v)$ denote the shortest distance between vertices $u, v \in V$. An algorithm is said to compute all-pairs t-approximate shortest-paths/distances, for some $t \geq 1$, if for each pair of vertices $u, v \in V$, the path/distance reported by the algorithm is not longer/greater than $t \cdot \delta(u, v)$.

This paper presents two randomized algorithms for computing all-pairs nearly 2-approximate distances. The first algorithm takes expected $O(m^{2/3} n \log n + n^2)$ time, and for any $u, v \in V$ reports distance no greater than $2\delta(u, v) + 1$. Our second algorithm requires expected $O(n^2 \log^{3/2})$ time, and for any $u, v \in V$ reports distance bounded by $2\delta(u, v) + 3$.

This paper also presents the first expected $O(n^2)$ time algorithm to compute all-pairs 3-approximate distances.

1 Introduction

All-pairs shortest path problem is undoubtedly one of the most fundamental algorithmic graph problem. Given a graph $G(V, E)$ on $n(= |V|)$ vertices and $m(= |E|)$ edges, the problem requires computation of shortest-paths/distances between each pair of vertices. There are various versions of this problem depending on whether the graph is directed or undirected, edges are weighted or unweighted, weights are positive or negative. In its most generic version, that is, for directed graph with real edge-weights, the best known algorithm [6] for this problem requires $O(mn + n^2 \log \log n)$ time. However, for graphs with $m = \theta(n^2)$, this algorithm has a running time of $\theta(n^3)$ which matches that of the old and classical algorithm of Floyd and Warshal. The best known upper bound on the time complexity of this problem is $O(n^3 \sqrt{\log \log n} / \log n)$ due to Zwick [10], which is marginally sub-cubic.

In the recent past, there has been a growing interest in designing efficient (sub-cubic running time) and simple algorithms for all-pairs *approximate* shortest-paths, and successful attempts have been made for undirected graphs. Zwick [9]

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 666–679, 2005.

provides an excellent survey on algorithms for computing approximate shortest paths. An algorithm is said to compute all-pairs t-approximate distances, if for any pair of vertices $u, v \in V$, the distance $\delta^*(u,v)$ reported by the algorithm satisfies $1 \leq \frac{\delta^*(u,v)}{\delta(u,v)} \leq t$. In the following paragraph, we provide a very brief summary of the current state-of-the-art algorithms for all-pairs t-approximate shortest paths.

Cohen and Zwick [2], building upon the work of Dor et al. [3], designed an algorithm that given any undirected weighted graph with n vertices and m edges, computes all-pairs 2-approximate shortest paths in $O(n^{3/2}\sqrt{m})$ time and all-pairs 3-approximate shortest paths in just $O(n^2 \log n)$ time. For unweighted graphs, given arbitrarily small $\zeta, \epsilon, \rho > 0$, Elkin [4] designed an algorithm that requires $O(mn^\rho + n^{2+\zeta})$ time, and for any pair of vertices $u, v \in V$, reports distance $\delta^*(u,v)$ satisfying the inequality :

$$\delta(u,v) \leq \delta^*(u,v) \leq (1+\epsilon)\delta(u,v) + \beta$$

where β is a function of ζ, ϵ, ρ. If the two vertices $u, v \in V$ are separated by sufficiently long distances in the graph, the stretch $\frac{\delta^*(u,v)}{\delta(u,v)}$ ensured by Elkin's algorithm is quite close to $(1+\epsilon)$. But the stretch factor may be quite huge for short paths since β depends on ζ as $(1/\zeta)^{\log 1/\zeta}$, depends inverse exponentially on ρ and inverse polynomially on ϵ. Thorup and Zwick [7] introduced a remarkable data-structure called approximate distance oracle, that requires sub-cubic preprocessing time and sub-quadratic space, and yet answers an approximate-distance query in constant time (hence the name oracle). For a given integer $k \geq 2$, the space of the approximate distance oracle is $O(kn^{1+1/k})$ and it reports any $(2k-1)$-approximate distance query in $O(k)$ time. The preprocessing time for $(2k-1)$-approximate distance oracle is $O(mn^{1/k})$ which has been improved to $O(\min(mn^{1/k}, n^2 \log n))$ for unweighted graphs in [1]. Thorup and Zwick [7] also show that for any $t < 3$, a data-structure that answers any t-approximate distance query in constant time must occupy $\theta(n^2)$ space. This implies a lower bound of $\Omega(n^2)$ on space as well as on time complexity of any algorithm that answers any 2-approximate distance query in constant time. As mentioned above, the algorithm of Cohen and Zwick [2] establishes an upper bound of $O(n^{3/2}\sqrt{m})$ on time complexity of all-pairs 2-approximate shortest path problem.

1.1 Our Contribution

As an important contribution of this paper, we show that we can, in time $O(n^2 \operatorname{polylog} n)$, compute all-pairs *nearly* 2-approximate shortest paths for unweighted undirected graphs.

1. We first design a data-structure that, given any $u, v \in V$, requires constant time to report distance bounded by $2\delta(u,v) + 1$, that is, an additive error of one unit over the 2-approximate distance. The expected preprocessing time required to build this data-structure is $O(m^{2/3} n \log n + n^2)$. In this way, our new algorithm, at the expense of introducing an additive error of

just one unit, achieves a significant improvement in the running time over the previous best algorithm [2] for all-pairs 2-approximate distances. The improvement is by a factor of at-least $n^{1/6}$ for the range $m > n^{3/2}$, whereas for $m < n^{3/2}$, the new algorithm takes expected $O(n^2)$ time.

2. We further reduce the expected preprocessing time to $O(n^2 \log^{3/2} n)$ at the expense of increasing the additive error to 3, that is, given any pair of vertices $u, v \in V$, the distance $\delta^*(u, v)$ reported by our data-structure satisfies

$$\delta(u, v) \leq \delta^*(u, v) \leq 2\delta(u, v) + 3$$

As would become clear subsequently from the paper, the additive error shows up only in some restricted worst case only. In general, the algorithm will behave very much like a 2-approximate shortest path algorithm.

3. As an additional and final contribution, this paper shows that it takes expected $O(n^2)$ time to compute 3-approximate distance oracle of size $O(n^{3/2})$.

Without any modifications, all our data-structures for reporting approximate distances can also be used to report approximate shortest-paths in optimal time.

2 A New Scheme for 2-Approximate Shortest Paths

Let $G(V, E)$ be an unweighted graph. The basic construct of our scheme is a *restricted* breadth-first-search (BFS) tree defined as Ball as follows.

Definition 1. *For a vertex u and a set $R \subset V$ of vertices, $Ball(u, R)$ denotes the set of vertices of the graph, such that the distance from u to these vertices is less than the distance from u to the nearest vertex of the set R.*

New scheme for approximate distance

Let $R \subset V$ be a set of vertices. Let n_u denote the vertex from the set R nearest to u.

1. **Global distance information**
   For each vertex $s \in R$, keep a BFS tree storing distance to all the vertices in the graph.
2. **Local distance information**
   For each vertex $u \in V \backslash R$, compute distance to all the vertices of $Ball(u, R)$ and its nearest vertex n_u.
3. Keep a data-structure to determine, in constant time, whether any two Balls overlap (share a common vertex) or not.

The above scheme may appear similar to 3-approximate distance oracle of Thorup and Zwick [7] except the third step. It is this step that proves to be crucial in achieving 2-approximate distances.

Now we shall describe how our scheme can be used to answer a distance query with stretch 2.

Answering distance query using new scheme

$\mathcal{Q}(u, v)$: We answer a distance query between u and v in the following order.

- If $v \in Ball(u, R)$ or $u \in Ball(v, R)$:

  report $\delta(u, v)$

- Else if $Ball(u, R)$ and $Ball(v, R)$ overlap :

  report $\delta(u, w) + \delta(v, w)$ for some $w \in Ball(u, R) \cap Ball(v, R)$

- Else :

  report **minimum** of $(\delta(u, n_u) + \delta(n_u, v))$ and $(\delta(v, n_v) + \delta(n_v, u))$

The query Procedure $\mathcal{Q}(u, v)$ explores all the three possible cases in the fixed order. In the first two cases, we manage to report distance using only the local distance information stored at vertices u and v. In the final and the third case, when the two Balls are non-overlapping, we use the global distance information stored at n_u and n_v.

Lemma 1. *Given a graph $G(V, E)$ and any two vertices $u, v \in V$, the approximate distance between u and v as reported by the query procedure $\mathcal{Q}(u, v)$ is bounded by $2\delta(u, v) + 1$.*

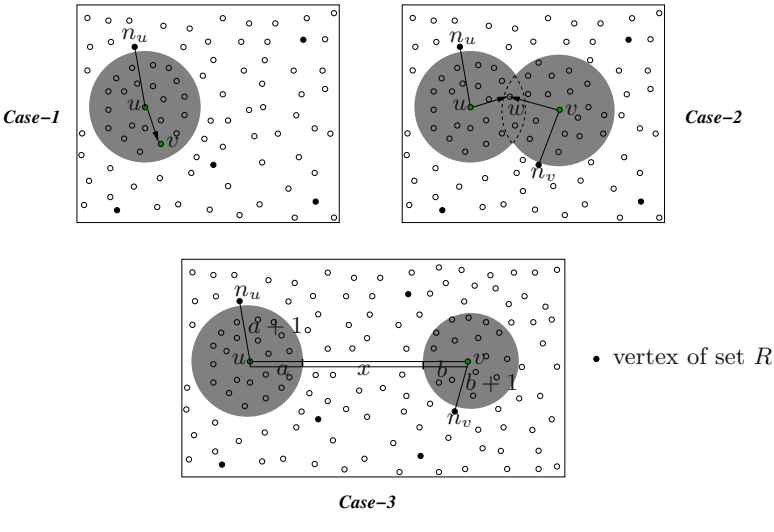


Fig. 1. Three cases in reporting distance between u and v

Proof. Let a and b be the radii of $Ball(u,R)$ and $Ball(v,R)$ respectively. The approximation factor associated with the distance reported by $\mathcal{Q}(u,v)$ depends on which of the three steps, we report the distance at. So we analyze the three cases as follows:

Case 1 : The distance is reported in the first step of $\mathcal{Q}(u,v)$.
In this case, either u or v lie in the Ball of the other. Without loss of generality, let us assume that v lies in $Ball(u,R)$ (see Fig. 1, *Case-1*). Here we report the *exact* distance between u and v.

Case 2 : The distance is reported in the second step of $\mathcal{Q}(u,v)$.
Note that since the query procedure failed to report the distance in the first step, therefore, the distance between u and v is more than the radius of $Ball(u,R)$ and $Ball(v,R)$. In other words, $\delta(u,v)$ is more than a and b. (see Fig. 1, *Case-2*).

Let w be a vertex lying in both $Ball(u,R)$ and $Ball(v,R)$. Clearly, $\delta(u,w) \leq a$ and $\delta(v,w) \leq b$. Therefore, the distance reported in this step is bounded by $a+b$, which is no more than $2\delta(u,v)$ as explained above.

Case 3 : The distance is reported in the third step of $\mathcal{Q}(u,v)$.
Since the query procedure failed to report the distance in the second step, $Ball(u,R)$ and $Ball(v,R)$ are separated by distance $x \geq 1$. So the shortest path between u and v can be viewed as consisting of three sub-paths : the first subpath is the portion of the path lying inside $Ball(u,R)$ and has length a, the second sub-path is the portion of the path lying outside the two Balls and has length x, and the third sub-path is the portion of the path lying inside $Ball(v,R)$ and has length b. Hence, the distance between u and v is $a+x+b$ for some $x \geq 1$. (see Fig. 1, *Case-3*).

In the third step, we report the minimum of $(\delta(u,n_u)+\delta(n_u,v))$ and $(\delta(v,n_v)+\delta(n_v,u))$. It can be noted that $\delta(u,n_u) = a+1$ and $\delta(v,n_v) = b+1$. Now considering the path from n_u to v passing through u, we can observe that $\delta(n_u,v)$ is bounded by $2a+x+b+1$. Similarly, analyzing the path from n_v to u passing through v, we can observe that $\delta(n_v,u)$ is bounded by $2b+x+a+1$. Therefore, the distance reported by $\mathcal{Q}(u,v)$ is bounded as follows.

$$\begin{aligned}
\min((\delta(u,n_u)+\delta(n_u,v)) \ , \ (\delta(v,n_v)+\delta(n_v,u))) & \\
&\leq \min(3a+x+b+2, 3b+x+a+2) \\
&= \min(3a+b, 3b+a)+x+2 \\
&= 3b+a+x+2 \quad \{\text{wlog assume that } a \geq b\} \\
&\leq 2a+2b+x+2 \quad \{\text{since } a \geq b\} \\
&\leq 2(a+x+b)+1 \quad \{\text{since } x \geq 1\} \\
&= 2\delta(u,v)+1
\end{aligned}$$

Hence, the distance between u, v as reported by $\mathcal{Q}(u,v)$ is bounded by $2\delta(u,v)+1$.

Remark: It is worth noting that the distance between any two vertices $u, v \in V$, as reported by $\mathcal{Q}(u,v)$, is bounded by $2\delta(u,v)$ even in the **Case-3**, if at-least one of the following conditions hold:

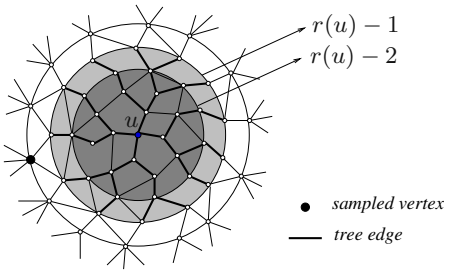


Fig. 2. To compute $Ball(u, R_p)$, we need to explore adjacency list of vertices lying in inner-shaded shell (of radius $r(u) - 2$) only

(*i*) $x > 1$, that is, the two Balls are separated by a path longer than one edge.
(*ii*) $a \neq b$, that is, the radii of the two Balls differs.

3 Efficient Sub-routines for Realization of the New Scheme

3.1 An Efficient Algorithm for Computing Balls

Let R_p be a set formed by selecting each vertex independently with probability $p < 1$. We shall now present an algorithm for computing $Ball(u, R_p)$, for all $u \in V \backslash R_p$.

Let $r(u)$ denote the distance from u to n_u. It follows from Definition 1 that the vertices of $Ball(u, R_p)$ and their distance from u can be computed by building a BFS tree at u up to level (distance) $r(u) - 1$. Therefore, prior to the computation of $Ball(u, R_p)$, we compute $r(u)$. In fact, the following procedure shows that it requires just a single BFS traversal to compute $r(u)$ and n_u, for all $u \in V \backslash R_p$.

Add a dummy vertex y to the given graph and connect it to all the vertices of set R_p. Compute a full BFS tree rooted at y. If a vertex u lies at level ℓ (hence at distance ℓ from y) in this BFS tree, it is at distance $\ell - 1$ from n_u, that is, $r(u) = \ell - 1$. In addition to $r(u)$, we can also compute n_u for each vertex u from this BFS tree.

If $r(u) = 1$, then $Ball(u, R_p)$ consists of vertex u only and we are done. For the case when $r(u) \geq 2$, we build a BFS tree upto level $r(u) - 1$ to compute $Ball(u, R_p)$, and the computation time required in doing so is of the order of the number of edges explored. Since the graph is undirected, an edge will be explored at-most twice (once by each of its end-point), and we would charge the cost of exploring an edge to that end-point, which explores it first. Let $v_1 (= u), v_2, \cdots, v_n$ be the sequence of vertices of the given graph arranged in non-decreasing order of their distance from u. Note that computing BFS tree upto level $r(u) - 1$ requires exploring the adjacency list of vertices up to level $r(u) - 2$ only (see Fig. 2). Therefore, in the computation of $Ball(u, R_p)$, we shall explore adjacency list of v_i if the following two events happen:

$\mathcal{E}_1^i$: There is no vertex in the set $\{v_j | j < i\}$ which is selected in the sample R_p.
$\mathcal{E}_2^i$: There is no vertex from $\{v_j | j > i\}$ that is adjacent to v_i and also a sampled vertex.

The events $\mathcal{E}_1^i$ and $\mathcal{E}_2^i$ are independent (since the vertices are sampled independently). Following our charging scheme mentioned above, exploring adjacency list of vertex v_i would contribute $O(d'(v_i))$ to the computation time of $Ball(u, R_p)$, where $d'(v_i)$ is the number of edges incident on v_i from vertices $\{v_j | j > i\}$. So the expected cost of computing $Ball(u, R_p)$ is

$$\begin{aligned}\sum_{i=1}^{n} \left(\Pr(\mathcal{E}_1^i) \cdot \Pr(\mathcal{E}_2^i) \cdot d'(v_i)\right) &= \sum_{i=1}^{n} \left((1-p)^{i-1}(1-p)^{d'(v_i)} d'(v_i)\right) \\ &\leq \sum_{i=1}^{n} \left((1-p)^{i-1} \sum_{j=1}^{d'(v_i)} (1-p)^{j-1}\right) \\ &\leq \sum_{i=1}^{n} \left((1-p)^{i-1} \frac{1}{p}\right) \leq \frac{1}{p} \sum_{i=1}^{n} (1-p)^{i-1} \leq \frac{1}{p^2}.\end{aligned}$$

Theorem 1. *Given an unweighted graph $G(V, E)$, and $p < 1$, let R_p be a set formed by selecting each vertex independently with probability p. There exists an algorithm for computing $Ball(u, R_p)$, for all $u \in V \backslash R_p$ in expected time $O(m + \frac{n}{p^2})$.*

In a similar way, it can be shown that the expected number of vertices in $Ball(u, R_p)$ is $O(1/p)$ (observe that for vertex v_i to belong to $Ball(u, R_p)$, the event $\mathcal{E}_1^i$ must happen).

Lemma 2. *Given an unweighted graph $G(V, E)$, a uniformly random sample $R \subset V$ of size $\sqrt{n}$ induces a bound of $\sqrt{n}$ on expected size of $Ball(u, R)$.*

It follows from Definition 1 that $Ball(u, X) \subset Ball(u, Y)$, for all $Y \subset X \subset V$. Therefore, our algorithm would require less time to compute $Ball(u, X)$ than to compute $Ball(u, Y)$. So we can state the following corollary based on Theorem 1.

Corollary 1. *Given an unweighted graph $G(V, E)$ and $p < 1$, let R_p be a set formed by selecting each vertex independently with probability $p < 1$. For any set $R \supset R_p$, it takes expected $O(m + \frac{n}{p^2})$ time to compute $Ball(u, R)$, for all $u \in V \backslash R$.*

3.2 Computing Overlap Matrix $\mathcal{O}$

To determine, for a pair of vertices $u, v \in V$, whether there exists a vertex common to both $Ball(u, R)$ and $Ball(v, R)$, we keep a matrix $\mathcal{O}$ such that $\mathcal{O}[u, v]$ is null if $Ball(u, R) \cap Ball(v, R) = \emptyset$, otherwise $\mathcal{O}[u, v]$ stores a vertex that belongs to both the Balls. To build the matrix $\mathcal{O}$ efficiently, we form the following sets,

$$C(v, R) = \{u \in V | v \in Ball(u, R)\}.$$

It is easy to observe that we can form the sets $\{C(v,R)|v \in V\}$ by a single scan of the sets $\{Ball(u,R)|u \in V\}$. Now once we have $C(v,R), \forall v \in V$, we can compute the matrix $\mathcal{O}$ as follows.

Algorithm for computing overlap matrix $\mathcal{O}$

```
For each v ∈ V\R do
    For each u ∈ C(v,R) do
        For each w ∈ C(v,R) do
            O[u,w] ← v
```

The running time of the above algorithm for computing the overlap matrix $\mathcal{O}$ is of the order of $\sum_{v \in V} |C(v,R)|^2 + n^2$. In order to compute the overlap matrix $\mathcal{O}$ in $O(n/p^2 + n^2)$ time (which also matches the time required to compute Balls), we would need a set $R \subset V$ which would ensure that $|C(v,R)| = O(1/p)$, for all $v \in V$. We shall employ the random sampling scheme given by Thorup and Zwick [8] to compute the desired sample R as follows.

Procedure for computing the sample set R

```
Procedure sample(G, p) {
    R = ∅; V' = V;
    While (V' ≠ ∅)
        Add a uniform sample of size np from V' to R;
        For every u ∈ V\R do
            Compute Ball(u,R);
        For every v ∈ V\R do
            C(v,R) ← {u ∈ V | v ∈ Ball(u,R)};
        V' ← {v ∈ V | |C(v,R)| > 4/p};
    Return R;
}
```

For the first iteration, R is a uniform sample from V, that is, $R = R_p$. So using Theorem 1, the first iteration requires expected $O(m + n/p^2)$ time. Note that in each subsequent iteration, the sample R only grows. Hence it follows from Corollary 1 that each subsequent iteration will also require expected $O(m+n/p^2)$ time. It is shown in [8] that in every iteration, the size of V' decreases by a factor of 2 with probability at-least $1/2$. Hence after expected $\log n$ iterations, V' would be reduced to $\emptyset$. Thus, the expected size of the final sample R would be $np \log n$ and $C(v,R)$ will be bounded by $O(\frac{1}{p})$, for each $v \in V$.

Theorem 2. *Given an unweighted graph $G(V,E)$ and $p < 1$, a set $R \subset V$ of size $O(np \log n)$ can be computed in expected $O(m \log n + \frac{n}{p^2} \log n)$ time ensuring that*

- *It takes a total of $O(m + n/p^2)$ time to compute $Ball(u,R), \forall u \in V\backslash R$.*
- *It takes $O(n^2 + \frac{n}{p^2})$ time to build the overlap matrix $\mathcal{O}$.*

4 Algorithms for Nearly 2-Approximate Shortest Paths

4.1 Algorithm I

Our first algorithm for computing nearly 2-approximate distances is a realization of the scheme mentioned in Sect. 2.

Algorithm I

Preprocessing

Let R be the set of vertices as defined by Theorem 2.

1. For each $u \in V \backslash R$, compute $Ball(u, R)$.
2. Compute overlap matrix $\mathcal{O}$.
3. For each $v \in R$, build a full BFS tree rooted at v in the graph.

Reporting distance between $u, v \in V$

$\mathcal{Q}(u, v)$

Computing BFS tree from vertices of the set R requires $O(m|R|) = O(mnp \log n)$ expected time. Hence applying Theorem 2, it follows that the total expected time for preprocessing the graph in Algorithm I is given by

$$m \log n + n^2 + \frac{n}{p^2} \log n + mnp \log n = n^2 + m^{2/3} n \log n \qquad \{ \text{ for } p = \tfrac{1}{\sqrt[3]{m}} \}.$$

Theorem 3. *An unweighted graph $G(V, E)$ can be preprocessed in $O(m^{2/3} n \log n + n^2)$ expected time to output a data-structure of size $O(n^2)$ that, given any $u, v \in V$, requires constant time to report distance $\delta^*(u, v)$ satisfying*

$$\delta(u, v) \leq \delta^*(u, v) \leq 2\delta(u, v) + 1.$$

The previous best known algorithm [2] for computing 2-approximate distances requires $O(n^{3/2}\sqrt{m})$ running time. Thus, in the worst case, we have been able to improve the running time by a factor of $O(n^{1/6})$ at the expense of introducing an additive error of just one unit.

4.2 Algorithm II

The preprocessing time of the first two steps in Algorithm I described above can be bounded by $O(n^2 \log n)$ with a suitable choice of p. The third step that computes BFS trees from vertices of set R requires $O(m|R|)$ time, which is certainly not $O(n^2 \log n)$ when the graph is dense. To improve its preprocessing time to $(n^2 \operatorname{polylog} n)$, one idea is to perform BFS from R on a spanner (having $o(n^2)$ edges) of the original graph. A spanner is a subgraph that is sparse but still preserves approximate distance between vertices in the graph.

Definition 2. *Given $\alpha \geq 1, \beta \geq 0$, a subgraph $G(V, E')$, $E' \subset E$ is said to be an (α, β)-spanner of $G(V, E)$ if for each pair of vertices $u, v \in V$, the distance $\delta_s(u, v)$ in the spanner is bounded by $\alpha\delta(u, v) + \beta$.*

The sparsity of a spanner comes along with the stretching of the distances in the graph. So one has to be careful in employing an (α, β)-spanner (with $\alpha > 1$) in the third step, lest one should end up computing nearly 2α-approximate distances instead of nearly 2-approximate distances. To explore the possibility of using spanner in our algorithm, let us revisit our distance reporting scheme $\mathcal{Q}(u, v)$. The full BFS trees rooted at the vertices of set R serve to provide global distance information in the scheme $\mathcal{Q}(u, v)$ and they are required, only when u and v belong to non-overlapping Balls. In the analysis of this case, we partitioned the shortest path between u and v into three sub-paths (see Fig. 1, *Case-3*): the sub-paths of lengths a and b covered by $Ball(u, R)$ and $Ball(v, R)$ respectively, and the sub-path of length x lying between the two Balls and not covered by either Ball. We showed that the distance $\delta^*(u, v)$ as reported by $\mathcal{Q}(u, v)$ is bounded by $2a + 2b + x + 2$. A comparison of this expression of $\delta^*(u, v)$ with $\delta(u, v) = a + x + b$ suggests that there is a possibility of stretching the uncovered sub-path (of length x) between the Balls by a factor of 2 and still keeping the distance reported to be nearly 2-approximate. So we may employ an (α, β)-spanner in the third step of our algorithm, provided α (the multiplicative stretch) is not greater than 2 and more importantly, for each vertex $u \in V \backslash R$, the shortest path from n_u to u as well as the shortest paths from u to all the vertices of $Ball(u, R)$ are preserved in the spanner. To ensure these additional features, we shall employ the parameterized spanner introduced in [1].

Parameterized (2,1)-Spanner [1]

Given a graph $G(V, E)$, and a parameter $X \subset V$, a subgraph $G(V, E')$ is said to be a parameterized $(2, 1)$-spanner with respect to X if
(*i*) $G(V, E')$ is a $(2, 1)$-spanner.
(*ii*) All those edges whose at-least one endpoint is not adjacent to any vertex from
the set X are surely present in the spanner too.

An $O(m)$ time algorithm has been presented in [1] to compute a parameterized $(2, 1)$-spanner for a given $X \subset V$ (as a parameter). To ensure that the spanner is a parameterized $(2, 1)$-spanner, the algorithm establishes the following lemma.

Lemma 3. *[1] For an edge $e(u, v)$ not present in the spanner, there is a vertex $x \in X$ adjacent to u in the spanner such that there is a path from x to v in the spanner of length no more than 2.*

The feature (*ii*) of the parameterized $(2, 1)$-spanner suggests that if we choose R as the parameter, all the edges lying inside a Ball are present in the parameterized spanner, and hence all the shortest paths that lie within a Ball are

preserved too. Now observe that the shortest path from n_u to u lies fully inside $Ball(u, R)$ except the first edge of this path which is incident on n_u. So to ensure that the shortest path from n_u to u is also preserved, it would suffice if we augment the spanner with all the edges in the original graph that are incident on n_u.

Algorithm II

Preprocessing

Let R be the set of vertices as defined by Theorem 2.

1. For each $u \in V \setminus R$, compute $Ball(u, R)$.
2. Compute overlap matrix $\mathcal{O}$.
3. (a) Let $G(V, E')$ be a parameterized $(2, 1)$-spanner with respect to R for the given graph $G(V, E)$.
   (b) For each $v \in R$, compute a full BFS tree rooted at v in $G(V, E' \cup E(v))$. ($E(v)$ denotes the edges incident on v in the original graph $G(V, E)$)

Reporting distance between $u, v \in V$

$\mathcal{Q}(u, v)$

From the discussion above, it follows that for any pair of vertices $u, v \in V$, the distance reported in Case-3 by $\mathcal{Q}(u, v)$ will be

$$\begin{aligned} \delta^*(u,v) &\leq 2(a+b)+(2x+1)+2 \quad \{\text{since } x \text{ is stretched to } 2x+1\ \} \\ &= 2(a+x+b)+3 \quad = \quad 2\delta(u,v)+3. \end{aligned}$$

To analyze the running time of Algorithm II, observe that we perform BFS on a $(2, 1)$-spanner. Therefore, a bound on the size of the spanner is required. We shall use the following lemma from [1].

Lemma 4. *[1] Let R_p be a set formed by selecting each vertex independently with probability p. For any set $R \supset R_p$, the expected size of parameterized $(2, 1)$-spanner will be $O(|R|n + n/p)$.*

Lemma 4 implies that the size of $(2, 1)$-spanner with parameter R (as determined in Sect. 3.2) would be $O(n/p + nnp \log n)$. Hence the expected preprocessing time of Algorithm II will be of the order of

$$m \log n + \frac{n}{p^2} \log n + np \log n \left(\frac{n}{p} + n^2 p \log n\right) = O(n^2 \log^{\frac{3}{2}} n) \quad \{\text{for } p = \frac{1}{\sqrt{n}\sqrt[4]{\log n}}\}$$

Theorem 4. *An unweighted graph $G(V, E)$ can be preprocessed in $O(n^2 \log^{3/2} n)$ expected time to output a data-structure of size $O(n^2)$ that, given any $u, v \in V$, requires constant time to report distance $\delta^*(u, v)$ satisfying*

$$\delta(u, v) \leq \delta^*(u, v) \leq 2\delta(u, v) + 3.$$

5 Algorithm for 3-Approximate Shortest Paths

Algorithm *3-approx*

Preprocessing

Let R be a set formed by picking each vertex with probability $1/\sqrt{n}$.

1. For each $u \in V \backslash R$, compute $Ball(u, R)$.
2. (a) Let $G(V, E')$ be a parameterized $(2, 1)$-spanner with respect to R.
   (b) For each $v \in R$, compute a full BFS tree rooted at v in $G(V, E' \cup E(v))$.

Reporting distance between vertices $u, v \in V$

If $v \in Ball(u, R)$ or $u \in Ball(v, R)$,
report $\delta(u, v)$
Else
report **minimum** of $(\delta(u, n_u) + \delta_s(n_u, v))$ and $(\delta(v, n_v) + \delta_s(n_v, u))$
(note that $\delta_s(x, y)$ denotes distance between x and y in the underlying spanner.)

Lemma 5. *The Algorithm 3-approx reports all-pairs 3-approximate distance.*

Proof. Let neither $u \in Ball(v, R)$ nor $v \in Ball(u, R)$. We have two cases now.

Case 1: $Ball(u, R) = \{u\}$
Let $v_0(= u), v_1, \cdots, v_l(= v)$ be the shortest path between u and v. Since $Ball(u, R)$ consists of vertex u only, u must be adjacent to n_u. Also by Lemma 3, there is a path of length at-most 2 units between n_u and v_1 in the parameterized spanner. Furthermore, the distance $\delta_s(v_1, v_l)$ between v_1 and v_l in the spanner is no greater than $3(l-1)$. Hence $\delta_s(n_u, v) \leq 3(l-1) + 2$. Since $\delta(u, n_u) = 1$, the distance reported by the Algorithm *3-approx* is no more than $3l$.

Case 2: $Ball(u, R) \neq \{u\}$
Consider $Ball(v, R)$. Let its radius be $a \geq 0$. The vertex u lies outside this Ball. The shortest path from v to u can be visualized as consisting of two segments (see Fig. 3) : the sub-path $\mathcal{P}_{vw}$ of length a lying inside the Ball and the sub-path $\mathcal{P}_{wu}$ of length $x \geq 1$ outside the Ball. Let the length of path $\mathcal{P}_{wu}$ be stretched to x' in the parameterized spanner. Since the parameterized spanner preserves

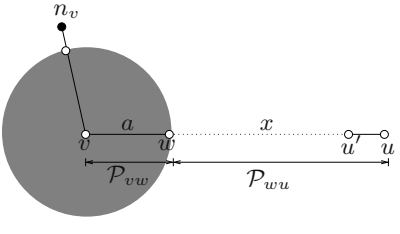


Fig. 3. Analyzing the path from v to u when $Ball(u, R) \neq \{u\}$

all-paths within a Ball, therefore, the distance $(\delta(v, n_v) + \delta_s(n_v, u))$ reported by the algorithm would be $3a + 2 + x'$. To ensure that this distance is no more than three times the actual distance $\delta(u, v) = a + x$, all we need to show is that $x' \leq 3x - 2$. Let $e(u', u)$ be the last edge of the path $\mathcal{P}_{wu}$. Now observe that $Ball(u, R) \neq \{u\}$ implies that $Ball(u, R)$ has radius at-least one. So the edge $e(u', u)$ must be present in the spanner (see feature (ii) of the parameterized spanner). Now the part of the path $\mathcal{P}_{wu}$ excluding the edge $e(u', u)$ is of length $x-1$, and can't be stretched to more than $3(x-1)$ in the spanner. Hence $x' \leq 3(x-1)+1 = 3x-2$, and we are done.

The set R used in the Algorithm *3-approx* is a uniform random sample of $\sqrt{n}$ vertices. Hence using Theorem 1, it follows that a total of $O(n^2)$ expected time is required to compute $Ball(u, R), \forall u \in V \backslash R$. Also Lemma 4 implies that the number of edges in $(2, 1)$-spanner with parameter R is $O(n\sqrt{n})$. So the expected time required for building full BFS trees on all the vertices of R in the $(2, 1)$-spanner is $O(n^2)$. Hence the expected preprocessing time of the Algorithm *3-approx* is $O(n^2)$.

Space Requirement: Keeping distance information from each $x \in R$ to all the other vertices requires a total of $O(n^{3/2})$ storage. For each vertex $u \in V \backslash R$, we can store the vertices belonging to $Ball(u, R)$ along with their distance from u in a 2-level hash table of [5] with optimal size $O(|Ball(u, R)|)$. Using this hash table, it can be determined whether $v \in Ball(u, R)$ or not in worst case constant time. So the space requirement of the data-structure of our algorithm (3-approximate distance oracle) given above is $O(n^{3/2})$, which is optimal as shown in [7].

Theorem 5. *An unweighted graph $G(V, E)$ can be preprocessed in expected $O(n^2)$ time to output a data-structure of size $O(n^{3/2})$ that can answer any 3-approximate distance query in constant time.*

6 Conclusion and Open Problems

Given an undirected unweighted graph $G(V, E)$ on $|V| = n$ vertices, we can compute all-pairs 3-approximate distances in $O(n^2)$ time. More importantly, we can compute nearly 2-approximate distances in $O(n^2 \operatorname{polylog} n)$ time. This upper bound is quite close to the $\Omega(n^2)$ lower bound on computing 2-approximate distances. It would be quite interesting to remove the additive error from our algorithm completely or to prove an $\Omega(n^{2+\epsilon})$ lower bound for computing all-pairs 2-approximate distances.

References

1. S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 271–280, 2004.

2. E. Cohen and U. Zwick. All-pairs small stretch paths. *Journal of Algorithms*, 38:335–353, 2001.
3. D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *Siam Journal on Computing*, 29:1740–1759, 2000.
4. M. Elkin. Computing almost shortest paths. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 53–62, 2001.
5. M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $o(1)$ worst case time. *Journal of ACM*, 31:538–544, 1984.
6. S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312:47–74, 2004.
7. M. Thorup and U. Zwick. Approximate distance oracle. In *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pages 183–192, 2001.
8. M. Thorup and U. Zwick. Compact routing schemes. In *13th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2001.
9. U. Zwick. Exact and approximate distances in graphs - a survey. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA)*, pages 33–48, 2001.
10. U. Zwick. Slightly improved sub-cubic bound for computing all-pairs shortest paths. In *15th Annual International Symposium on Algorithms and Computation (ISAAC), to appear*, 2004.

Pattern Occurrences in Multicomponent Models

Massimiliano Goldwurm and Violetta Lonati

Dip. Scienze dell'Informazione, Università degli Studi di Milano
Via Comelico 39/41, 20135 Milano – Italy
{goldwurm, lonati}@dsi.unimi.it

Abstract. In this paper we determine some limit distributions of pattern statistics in rational stochastic models, defined by means of nondeterministic weighted finite automata. We present a general approach to analyse these statistics in rational models having an arbitrary number of connected components. We explicitly establish the limit distributions in the most significant cases; these ones are characterized by a family of unimodal density functions defined by polynomials over adjacent intervals.

Keywords: Automata and Formal Languages, Limit Distributions, Nonnegative Matrices, Pattern Statistics, Rational Formal Series.

1 Introduction

This work presents some results on the limit distribution of pattern statistics. The major problem in this context is to estimate the frequency of pattern occurrences in a random text. This is a classical problem that has applications in several research areas of computer science and biology: for instance, it is considered in connection with the search of motifs in DNA sequences [6, 14] while the earlier motivations are related to code synchronization [10] and approximated pattern-matching [12, 18, 5]. In the usual setting, established in the seminal paper [11] and developed in many subsequent works (see for instance [15, 13, 3]), one considers a finite alphabet Σ, a set of patterns $R \subseteq \Sigma^*$, a probabilistic source P generating words at random in Σ^*, and studies the number X_n of occurrences of elements of R in a word of length n generated by P. Typical goals are the asymptotic evaluation of the moments of X_n, its limit distribution (also in the local sense) and the corresponding large deviations. These results depend in particular on the stochastic model P, which is usually assumed to be a Bernoulli or a Markovian model.

A rather general result is presented in [13], where Gaussian limit distributions are obtained, for any regular set of patterns R and any Markovian source P, under a primitivity hypothesis on the associated stochastic matrix. This result is extended in [2] to the so-called *rational stochastic model*, where the text is generated at random according to a probability distribution defined by means of a rational formal series in noncommutative variables. In particular cases, this is simply the uniform distribution over the set of words of given length in an

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 680–692, 2005.

arbitrary regular language. For this reason, results for this model are also related to the analysis of additive functions over strings [9].

The rational stochastic model properly extends the Markovian models in the following sense: the frequency problem of regular patterns in a text generated in the Markovian model (as studied in [13]) is a special case of the frequency problem of a single symbol in a text over a binary alphabet generated in the rational stochastic model; it is also known that the two models are not equivalent [2]. We recall that extensions of the Markovian models have already been considered in the literature [3]. Furthermore, finding results under more general probabilistic assumptions is of interest since, for some applications, the Markovian models seem to be too restrictive.

Also in the rational stochastic models, Gaussian limit distributions are obtained under a primitive hypothesis, i.e. when the matrix associated with the rational formal series (counting the transitions between states) is primitive [2]. A complete study of the limit distributions is given in [4] in the bicomponent models, that is when the previous matrix has two primitive components.

In this paper we present a general approach to the analysis of multicomponent rational models, explicitly establishing the limit distribution in the most significant cases. The paper is organized as follows. In Section 2 we give the definition and the main properties of rational models. In Section 3 we show how this model can be decomposed and we introduce the notions of main chain and simple model. Under a special assumption on the main chain, in Section 4 we determine the limit distributions of pattern statistics for simple models. They are characterized by an interesting family of unimodal density functions defined by polynomials over adjacent intervals. Finally in Section 5 we extend the results to all simple models and also provide a natural method to determine the limit distribution in the general case.

2 Rational Models for Pattern Statistics

In this section we recall some basic notions on rational formal series [16, 1] and the corresponding stochastic models to study the number of symbol occurrences in words chosen at random.

Let $\mathbb{R}_+$ be the semiring of all nonnegative real numbers and consider a finite alphabet Σ. A formal series over Σ with coefficients in $\mathbb{R}_+$ is a function $r : \Sigma^* \longrightarrow \mathbb{R}_+$, usually represented in the form $r = \sum_{\omega\in\Sigma^*}(r,\omega)\cdot\omega$, where (r,ω) denotes the value of r at $\omega \in \Sigma^*$. Moreover, r is called *rational* if it admits a *linear representation*, that is a triple (ξ,μ,η) where, for some integer $m > 0$, ξ and η are (column) vectors in $\mathbb{R}_+^m$ and $\mu : \Sigma^* \longrightarrow \mathbb{R}_+^{m\times m}$ is a monoid morphism, such that $(r,\omega) = \xi^T\mu(\omega)\ \eta$ holds for each $\omega \in \Sigma^*$. Observe that considering such a triple (ξ,μ,η) is equivalent to defining a (weighted) nondeterministic automaton, where the state set is given by $\{1,2,\ldots,m\}$ and the transitions, the initial and the final states are assigned weights in $\mathbb{R}_+$ by μ, ξ and η, respectively. To avoid redundancy it is convenient to assume that (ξ,μ,η) is trim (meaning that all indices are used to define the series), i.e. for every index i there are two

indices p, q and two words $x, y \in \Sigma^*$ such that $\xi_p \mu(x)_{pi} \neq 0$ and $\mu(y)_{iq} \eta_q \neq 0$. We say that (ξ, μ, η) is *primitive* if $M = \sum_{\sigma \in \Sigma} \mu(\sigma)$ is a primitive matrix, that is for some $n \in \mathbb{N}$ all entries of M^n are strictly positive. We also recall that a matrix $M \in \mathbb{R}_+^{m \times m}$ is called *irreducible* if for every pair of indices p, q there exists $n \in \mathbb{N}$ such that $M^n_{pq} > 0$.

Any formal series can define a stochastic model for studying the frequency of occurrences of a letter in a word of given length. Consider the binary alphabet $\{a, b\}$ and, for any $n \in \mathbb{N}$, let $\{a, b\}^n$ denote the set of all words of length n in $\{a, b\}^*$. Consider a formal series $r : \{a, b\}^* \longrightarrow \mathbb{R}_+$ and let n be a positive integer such that $(r, x) \neq 0$ for some $x \in \{a, b\}^n$. A probability measure over $\{a, b\}^n$ can be defined by setting

$$\Pr\{\omega\} = \frac{(r, \omega)}{\sum_{x \in \{a,b\}^n} (r, x)} \qquad (\omega \in \{a, b\}^n). \tag{1}$$

In particular, if r is the characteristic series χ_L of a language $L \subseteq \{a, b\}^*$, then Pr is just the uniform probability function over $L \cap \{a, b\}^n$. Then, we define the random variable (r.v. for short) $Y_n : \{a, b\}^n \to \{0, 1, \ldots, n\}$ such that $Y_n(\omega) = |\omega|_a$ for every $\omega \in \{a, b\}^n$. For every $j = 0, 1, \ldots, n$, we have

$$\Pr\{Y_n = j\} = \frac{\sum_{|\omega|=n, |\omega|_a = j} (r, \omega)}{\sum_{x \in \{a,b\}^n} (r, x)} \; .$$

If $r = \chi_L$ for some $L \subseteq \{a, b\}^*$, then Y_n represents the number of occurrences of a in a word chosen at random in $L \cap \{a, b\}^n$ under uniform distribution.

When r is rational, the probability space given by (1) defines a stochastic model we call *rational* stochastic model. It is a generalization of the Markovian models in the sense that the r.v.'s Y_n for rational r represent, in special cases, the number of occurrences of patterns from an arbitrary regular language in words generated at random by Markovian processes [2–Section 2.1].

Let (ξ, μ, η) be a linear representation for the rational series r and set $\mathcal{A} = \mu(a)$, $\mathcal{B} = \mu(b)$, $\mathcal{M} = \mathcal{A} + \mathcal{B}$. To study the behaviour of the random variables Y_n and in particular their limit distribution, it is useful to introduce the sequence of functions $\{r_n(z)\}_n$ in one complex variable z defined by

$$r_n(z) = \sum_{x \in \{a,b\}^n} (r, x) \cdot e^{z|x|_a} = \xi^T (\mathcal{A} e^z + \mathcal{B})^n \eta.$$

Indeed, it is immediate to see that the characteristic function of Y_n satisfies the relation

$$\Psi_{Y_n}(t) = \mathbb{E}(e^{itY_n}) = \frac{r_n(it)}{r_n(0)} \tag{2}$$

for $t \in \mathbb{R}$. We recall that a sequence of random variables X_n converges in distribution to a random variable X if and only if the sequence of characteristic functions $\Psi_{X_n}(t)$ pointwise converges to $\Psi_X(t)$ [7].

Now consider the generating function of $\{r_n(z)\}_n$. Note that $\sum_{n=0}^{\infty} r_n(z) w^n = \xi^T H(z, w) \eta$, where $H(z, w)$ is the matrix function defined by

$$H(z, w) = \sum_{n=0}^{\infty} (\mathcal{A} e^z + \mathcal{B})^n w^n = (I - w(\mathcal{A} e^z + \mathcal{B}))^{-1} \, . \tag{3}$$

If $\mathcal{M}$ is irreducible, by the Perron–Frobenius Theorem (see [17–Theorem 1.5]) it has a nonnegative real eigenvalue λ of maximum modulus. Moreover, if $\mathcal{M}$ is primitive, then all other eigenvalues have modulus strictly lower than λ. If further $\mathcal{A} \neq 0 \neq \mathcal{B}$, then there are two constants $\beta \in (0,1)$, $\gamma > 0$, both depending on the matrix $\mathcal{M}$ and its eigenvectors (see [2] for details), such that, as n tends to infinity, the following relations hold:

$$\mathbb{E}(Y_n) = \beta n + \mathrm{O}(1)\ , \qquad \mathbb{V}ar(Y_n) = \gamma n + \mathrm{O}(1)\ . \tag{4}$$

For sake of brevity we say that β and γ are the *mean constant* and the *variance constant* of the primitive matrix $\mathcal{M}$, respectively. Under the same hypothesis, one can also prove [2] that the distribution of $\frac{Y_n - \beta n}{\sqrt{\gamma n}}$ converges to the normal distribution of mean value 0 and variance 1.

3 Decomposition of a Rational Model

Up to now, the properties of Y_n have been studied only in the primitive models [2] and in the case of two primitive components [4]. Here we present a general approach to deal with an arbitrary rational model. To this aim, we describe the construction of the reduced graph of the strongly connected components of the corresponding linear representation. This is a usual approach in the analysis of counting problems on regular languages (see for instance [8] for an application concerning trace languages).

Let (ξ, μ, η) be a linear representation over the alphabet $\{a, b\}$ with coefficients in $\mathbb{R}_+$. As in the previous section, set $\mathcal{A} = \mu(a)$, $\mathcal{B} = \mu(b)$, $\mathcal{M} = \mathcal{A}+\mathcal{B}$ and consider the directed graph defined by $\mathcal{M}$, where the set of nodes is $\{1, 2, \ldots, m\}$ and (p, q) is an (oriented) edge if and only if $\mathcal{M}_{pq} \neq 0$. Then, let $C_1, C_2, \ldots, C_s$ be the strongly connected components of the graph and define C_i *initial* (resp. *final*) if $\xi_p \neq 0$ (resp. $\eta_p \neq 0$) for some $p \in C_i$. The *reduced graph* of (ξ, μ, η) is then defined as the directed acyclic graph G where $C_1, C_2, \ldots, C_s$ are the vertices and any pair (C_i, C_j) is an edge if and only if $i \neq j$ and $\mathcal{M}_{pq} \neq 0$ for some $p \in C_i$ and some $q \in C_j$.

Up to a permutation of indices, the matrix $\mathcal{M}$ can be represented as a triangular block matrix of the form

$$\mathcal{M} = \begin{pmatrix} M_1 & M_{12} & M_{13} & \cdots & M_{1s} \\ 0 & M_2 & M_{23} & \cdots & M_{2s} \\ & & \cdots & & \\ 0 & 0 & 0 & \cdots & M_s \end{pmatrix}$$

where each M_i corresponds to the strongly connected component C_i and every M_{ij} corresponds to the transitions from vertices of C_i to vertices of C_j in the original graph of $\mathcal{M}$. Also $\mathcal{A}$, $\mathcal{B}$, ξ and η admit similar decompositions: we define the matrices A_i, A_{ij}, B_i, B_{ij} and the vectors ξ_i, η_i in the corresponding way and we say that the component C_i *degenerates* if $A_i = 0$ or $B_i = 0$. Since each M_i is either irreducible or null, by the Perron–Frobenius Theorem it has a nonnegative real eigenvalue λ_i of maximum modulus. We call *main eigenvalue* of $\mathcal{M}$ the value

$\lambda = \max\{\lambda_i \mid i = 1, 2, \ldots, s\}$ and we say that C_i is a *dominant component* if $\lambda_i = \lambda$. Observe that $\lambda_i = 0$ only if C_i reduces to a loopless single node and hence from now on we assume $\lambda > 0$. If further M_i is primitive, we say that C_i is a *primitive component*.

The block decomposition of $\mathcal{M}$ also induces a decomposition of the matrix $H(z, w)$ defined in (3). More precisely, the blocks under the diagonal are all null, while the upper triangular part is composed by a family of matrices, say $H_{ij}(z, w)$, $1 \le i \le j \le s$. Note that the bivariate generating function $\xi^T H(z, w)\eta$, which is the main tool of our investigation, is now given by

$$\xi^T H(z, w)\eta = \sum_{n=0}^{\infty} \xi^T (\mathcal{A}e^z + \mathcal{B})^n \eta \cdot w^n = \sum_{1 \le i \le j \le s} \xi_i^T H_{ij}(z, w)\eta_j \ . \tag{5}$$

Setting $M_{ij}(z) = A_{ij}e^z + B_{ij}$ and reasoning by induction on $j - i$, one can prove that, for each $1 \le j \le s$, the following equality holds

$$H_{jj}(z, w) = (I - w(A_j e^z + B_j))^{-1} = \frac{\mathrm{Adj}(I - w(A_j e^z + B_j))}{\det(I - w(A_j e^z + B_j))} \ , \tag{6}$$

while for each $1 \le i \le j \le s$ we have

$$H_{ij}(z, w) = \sum_{*} H_{i_1 i_1}(z, w) M_{i_1 i_2}(z) H_{i_2 i_2}(z, w) \cdots M_{i_{\ell-1} i_\ell}(z) H_{i_\ell i_\ell}(z, w) \cdot w^{\ell-1}, \tag{7}$$

where the sum $(*)$ is extended over all sequences of integers $(i_1, i_2, \ldots, i_\ell)$, $\ell \ge 2$ such that $i_1 = i$, $i_t < i_{t+1}$ for each $t = 1, \ldots, \ell - 1$ and $i_\ell = j$.

The previous equation suggests us to introduce the notion of chain of the reduced graph G associated with (ξ, μ, η). A *chain* is a simple path in G, i.e. any sequence of distinct components $\kappa = (C_{i_1}, C_{i_2}, \ldots, C_{i_\ell})$, $\ell \ge 1$, such that $M_{i_j i_{j+1}} \ne 0$ for every $j = 1, 2, \ldots, \ell - 1$. We say that ℓ is the *length* of κ while the *order* of κ is the number of its dominant components. Let Γ denote the family of all chains in G starting with an initial component and ending with a final component. We say that a chain κ is a *main chain* if $\kappa \in \Gamma$ and its order is maximal in Γ. We denote by Γ_m the set of all main chains in G.

In Section 3.1 we illustrate the role of main chains, which leads us to study the simple but representative case when the model has just one main chain, say κ. We first determine the limit distribution of Y_n when all dominant components of κ are primitive, non-degenerate and have distinct mean constants. A similar approach can be developed when the above mean constants are partially or totally coincident.

For this reason we introduce the notion of simple model. Formally, we say that (ξ, μ, η) is a *simple* linear representation, or just a *simple model*, if Γ_m contains only one chain κ and, for every dominant component C_i in κ, M_i primitive and $A_i \ne 0 \ne B_i$. Note that, for such a matrix M_i, the mean constant β_i and the variance constants γ_i can be defined as in (4), $0 < \beta_i < 1$ and $\gamma_i > 0$.

In simple models the limit distribution of Y_n first depends on the order k of κ, i.e. the number of its dominant components. If $k \le 2$ the limit distribution is known and derives from the analysis of the bicomponent models given in [4]:

– If κ has only one dominant component C_i then the limit distribution of $\frac{Y_n - \beta_i n}{\sqrt{\gamma_i n}}$ is a Gaussian distribution of mean value 0 and variance 1;
– If κ has two dominant components C_i, C_j then we have the following three subcases:
  1. If $\beta_i \neq \beta_j$ then Y_n/n converges in law to a random variable uniformly distributed in the interval $[b_1, b_2]$, where $b_1 = \min\{\beta_i, \beta_j\}$ and $b_2 = \max\{\beta_i, \beta_j\}$;
  2. If $\beta_i = \beta_j = \beta$ but $\gamma_i \neq \gamma_j$ then the limit distribution of $\frac{Y_n - \beta n}{\sqrt{n}}$ is a mixture of normal distributions of mean value 0 and variance uniformly distributed in the interval $[c_1, c_2]$, where $c_1 = \min\{\gamma_i, \gamma_j\}$ and $c_2 = \max\{\gamma_i, \gamma_j\}$. In other words, $\frac{Y_n - \beta n}{\sqrt{n}}$ converges in law to a random variable with density function

$$f(x) = \frac{1}{c_2 - c_1} \int_{c_1}^{c_2} \frac{e^{-x^2/(2v)}}{\sqrt{2\pi v}} dv \quad ;$$

  3. If $\beta_i = \beta_j = \beta$ and $\gamma_i = \gamma_j = \gamma$ then the distribution of $\frac{Y_n - \beta n}{\sqrt{\gamma n}}$ again converges to a Gaussian distribution of mean value 0 and variance 1.

In Section 4 we determine the limit distribution for simple models having main chain (of arbitrary order) with distinct mean constants of the dominant components; this result generalizes point 1) above. In Section 5, we extend these results to all simple models (with partially or totally coincident mean constants of dominant components) and also to all models whose main chains are simple (i.e. with primitive, non-degenerate dominant components).

We observe that the only cases not covered by our analysis concern the rational models where some dominant component of main chain is either non-primitive or degenerate. In the first case periodicity phenomena occur while in the second one a large variety of possible behaviours can be obtained even in bicomponent models [4].

3.1 The Role of Main Chains

In this section we show how the main chains determine the limit distribution of the sequence $\{Y_n\}$ associated with the linear representation (ξ, μ, η). Intuitively, this is a consequence of two facts. First, by equation (2) the characteristic function of (a normalization of) Y_n depends on the sequences $\{r_n(z)\}$ for z near 0, and hence on the generating function $\xi^T H(z, w)\eta$. Second, by (5), this function is a sum of products of the form given in (7), each of which is identified by a chain.

Thus, let us examine such terms. First consider the case $i = j$ and hence the terms of the form $\xi_j^T H_{jj}(z, w)\eta_j$. Relation (6) implies that, as z tends to 0, the singularities of each of its entries approach the inverses of eigenvalues of M_j. Then, we can distinguish three cases according whether M_j is dominant and primitive, dominant but non-primitive, or non-dominant. In each of these

cases, the Perron–Frobenius theory gives us the necessary information on the eigenvalues of M_j, that allows us to analyse the singularities of $\xi_j^T H_{jj}(z,w)\eta_j$ in a neighbourhood of $z = 0$.

The results of this analysis can be applied to functions $\xi_i^T H_{ij}(z,w)\eta_j$ where $i \neq j$. Recalling (7), we consider an arbitrary chain $\kappa = (C_{i_1}, C_{i_2}, \ldots, C_{i_\ell})$ with $\ell \geq 2$ and we define the sequence $\{r_n{}^{(\kappa)}(z)\}$ by setting

$$\sum_{n=0}^{\infty} r_n^{(\kappa)}(z) w^n = \xi_{i_1}^T H_{i_1 i_1}(z,w) M_{i_1 i_2}(z) H_{i_2 i_2}(z,w) \cdots M_{i_{\ell-1} i_\ell}(z) H_{i_\ell i_\ell}(z,w) \eta_{i_\ell} \cdot w^{\ell-1}. \quad (8)$$

Then one can prove that for $z = c/n$, $c \in \mathbb{C}$, the terms corresponding to the main chains have singularities of smallest modulus with the largest degree, and hence they yield the main asymptotic contribution to the associated sequence $\{r_n(c/n)\}$. Formalizing the previous intuitive argument, one gets the following result.

Theorem 1. *If all dominant components of the main chains are primitive and non-degenerate then, for every constant $c \in \mathbb{C}$, we have*

$$r_n(c/n) = \sum_{\kappa \in \Gamma_m} r_n^{(\kappa)}(c/n) \ (1 + O(1/n)) = \Theta(\lambda^n n^{k-1})$$

where k is the order of the main chains.[1]

We observe that Theorem 1 may not hold if the main chains admit non-primitive dominant components.

4 Main Results

In this section we determine the limit distribution of Y_n in the simple models that satisfy the following additional property: the dominant components of the main chain have (pairwise) distinct mean constants. This is related to a special family of distribution functions we call *polynomial*.

Consider a tuple $b = (b_1, b_2, \ldots, b_k)$ of $k \geq 2$ real numbers such that $0 < b_1 < b_2 < \cdots < b_k < 1$ and let $f_b : \mathbb{R} \longrightarrow \mathbb{R}$ be the function defined by

$$f_b(x) = \begin{cases} 0 & \text{if } x < b_1 \\ (k-1) \sum_{j=r}^{k} c_j (b_j - x)^{k-2} & \text{if } b_{r-1} \leq x < b_r \text{ for some } 1 < r \leq k \\ 0 & \text{if } x \geq b_k \end{cases} \quad (9)$$

where $c_j = \prod_{i \neq j} (b_j - b_i)^{-1}$ for every $j = 1, 2, \cdots, k$. In the following we say that a random variable X is a *polynomial* r.v. of parameters b if f_b is its density function. Note that if $k = 2$ then f_b is the uniform density function over the interval (b_1, b_2).

[1] In this work, for any pair of sequences $\{f_n\}$, $\{g_n\} \subseteq \mathbb{C}$, the expression $f_n = \Theta(g_n)$ means that there exist two positive constants a, b such that $a|g_n| \leq |f_n| \leq b|g_n|$ holds for every n large enough.

Theorem 2. *Let Y_n be defined in a simple model of main chain κ having order k and let $\beta = (\beta_1, \ldots, \beta_k)$ be the tuple of mean constants of dominant components in κ in non-decreasing order. If $k \geq 2$ and all β_j's are distinct then Y_n/n converges in law to a polynomial random variable of parameters β.*

Sketch of the proof. First consider $r_n(it/n)$. Theorem 1 allows us to focus on the contribution of $r_n{}^{(\kappa)}(it/n)$ corresponding to the main chain κ. Then, by the singularity analysis of its generating function (the right handside of equation (8)), one can show that for every $t \in \mathbb{R}$

$$r_n\left(\frac{it}{n}\right) = \sum_{h=0}^{k-1} S_h\left(\frac{it}{n}\right) \lambda^{n-h} D_h\left(\frac{it}{n}\right) \cdot (1 + \mathrm{O}(1/n)) \quad \text{as } n \to +\infty,$$

where, for each h, the function $S_h(z)$ is analytic at $z = 0$ and D_h is defined by

$$D_h(it/n) = \sum_{j=1}^{k} \frac{(1 + it\beta_j/n)^{n-h+k-1}}{\prod_{\ell \neq j}(it\beta_j/n - it\beta_\ell/n)} \quad \text{if } t \neq 0, \quad D_h(0) = \binom{n-h+k-1}{k-1}.$$

Recalling that the characteristic function $\Psi_{Y_n/n}(t)$ equals $r_n(it/n)/r_n(0)$, one can show that, as n tends to infinity, $\Psi_{Y_n/n}(t)$ converges to

$$\Phi_\beta(t) = \frac{(k-1)!}{(it)^{k-1}} \sum_{j=1}^{k} \frac{e^{i\beta_j t}}{\prod_{\ell \neq j}(\beta_j - \beta_\ell)}.$$

Finally, one can prove that $f_\beta(x)$ is a density function such that $\Phi_\beta(t)$ is its characteristic function (for details see Proposition 7). □

The properties of the family of polynomial distributions, together with the most relevant parts of the proof of Theorem 2, are all based on the convolutions of sequences defined by powers of complex numbers. In the following section we illustrate such properties and give some details of the proof sketched above.

4.1 Polynomial Distributions

Let us first consider the function $G_a(w) = w^{k-1} \cdot \prod_{i=1}^{k}(1 - a_i w)^{-1}$ where the tuple $a = (a_1, a_2, \ldots, a_k)$ has $k \geq 2$ nonnull complex components. Then G_a is the generating function of the convolution of the sequences $\{a_1^n\}_n, \{a_2^n\}_n, \ldots,$ $\{a_k^n\}_n$ shifted of $k-1$ indices. More precisely, at the point $w = 0$ such a function admits the power series expansion $G_a(w) = \sum_{n \geq 0} g_a(n) w^n$ such that

$$g_a(n) = \begin{cases} 0 & \text{if} \quad 0 \leq n \leq k-2 \\ \displaystyle\sum_{*} a_1^{i_1} a_2^{i_2} \cdots a_k^{i_k} & \text{if} \quad n \geq k-1 \end{cases} \tag{10}$$

where the sum (*) is extended over all k-tuples $(i_1, \ldots, i_k) \in \mathbb{N}^k$ such that $i_1 + \cdots + i_k = n - k + 1$. When all a_j's are distinct, the following proposition allows us to express the terms of the sequence $\{g_a(n)\}_{n \geq 0}$ in a useful form and provides us an important relationship among the a_j's.

Proposition 3. *Let $a = (a_1, a_2, \ldots, a_k)$ be a tuple of $k \geq 2$ distinct nonnull complex numbers and let the sequence $\{g_a(n)\}_n$ be defined by (10). Then, for every $n \in \mathbb{N}$, we have*

$$g_a(n) = \sum_{j=1}^{k} c_j \; a_j^n$$

where $c_j = \prod_{i \neq j}(a_j - a_i)^{-1}$ for every $j = 1, 2, \cdots, k$. Moreover, the polynomial $\sum_j c_j (a_j - x)^s$ is identically null for each $0 \leq s \leq k-2$ and in particular $\sum_j c_j a_j^s = 0$. Finally we have $\sum_j c_j a_j^{k-1} = 1$.

The application of the previous proposition yields the following results on f_b.

Proposition 4. *If $k \geq 3$ then f_b is continuously differentiable all over $\mathbb{R}$ up to the order $k-3$. Moreover the $(k-2)$-th derivative of f_b is well defined in $\mathbb{R} \backslash \{b_1, \ldots, b_k\}$ and is constant in each of the intervals (b_i, b_{i+1}), $i = 1, \cdots, k-1$.*

Lemma 5. *Let $f : \mathbb{R} \to \mathbb{R}$ be a function admitting j-th derivative all over $\mathbb{R}$ for some $j \geq 1$. Also assume that, for some reals $a < b$, f has m zeros in (a, b) and $f(x) = 0$ for each $x \leq a$ or $x \geq b$. Then, for every $i = 1, \ldots, j$, the i-th derivative of f admits at least $m + i$ zeros in (a, b).*

Proof. We reason by induction on $i = 1, \ldots, j$. If $i = 1$, then consider the $m+1$ intervals determined by the zeros of f in $[a, b]$. For each of them, say (x_1, x_2), Rolle's Theorem guarantees that $f'(x) = 0$ for some $x \in (x_1, x_2)$.

Now assume $1 < i < j$ and consider the i-th derivative of f, that is $g = f^{(i)}$. By the properties of f, we have $g(a) = g(b) = 0$ and by the inductive hypotheses g admits $m + i$ zeros in (a, b). Therefore, by applying the previous argument to g, one proves that $g' = f^{(i+1)}$ admits $m + i + 1$ zeros in (a, b). □

Proposition 6. *For every $k \geq 3$, the function f_b is nonnegative and admits a unique maximum all over $\mathbb{R}$.*

Proof. If $k = 3$ the property follows by a direct inspection of the function, which is linear and nonnull in the intervals (b_1, b_2) and (b_2, b_3). If $k \geq 4$, let us consider the $(k-3)$-th derivative ${f_b}^{(k-3)}(x)$ of $f_b(x)$. It is immediate to see that ${f_b}^{(k-3)}(x)$ is linear with respect to x in each of the $k-1$ intervals (b_i, b_{i+1}), $i = 1, \ldots k-1$. Moreover, by Proposition 3, it does not vanish in $(b_1, b_2) \cup (b_{k-1}, b_k)$. Thus, ${f_b}^{(k-3)}$ has at most $k-3$ many zeros in (b_1, b_k).

Now, assume by contradiction that f_b is not unimodal. Then its derivative f_b' vanishes in at least 3 points in the interval (b_1, b_k) and hence f_b' satisfies the hypotheses of Lemma 5 with $i = k-4$ and $m = 3$. As a consequence, ${f_b}^{(k-3)}$ admits at least $k-1$ zeros in (b_1, b_k), which contradicts the previous property. □

Fig.1 and Fig.2 show the graphics of the functions f_b having parameters $b = (0.1,\ 0.3, 0.4, 0.8)$ and $b = (0.008, 0.95, 0.96, 0.97, 0.98, 0.99)$, respectively. In each figure the first picture represents the entire curve, while the others show the details of the function in some subintervals. The vertical bars indicate the values

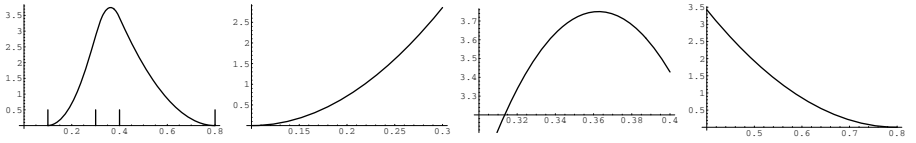


Fig. 1. Graphics of the function $f_b(x)$, where $b_1 = 0.1$, $b_2 = 0.3$, $b_3 = 0.4$, $b_4 = 0.8$. The vertical bars indicate the values of b_j's

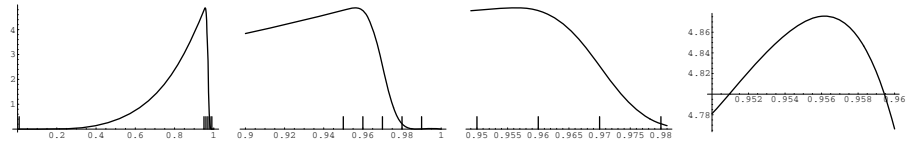


Fig. 2. Graphics of the function $f_b(x)$, where $b_1 = 0.008$, $b_2 = 0.95$, $b_3 = 0.96$, $b_4 = 0.97$, $b_5 = 0.98$, $b_6 = 0.99$. The vertical bars indicate the values of b_j's

of b_j's. Note that if $k = 4$ the maximum necessarily lays in the intermediate interval (b_2, b_3). On the contrary, if $k > 4$ the maximum can lay in any interval between b_2 and b_{k-1}. For instance in Fig. 2, due to the asymmetric position of the points b_j's, it lays in the second interval (b_2, b_3).

Proposition 7. *For every* $b = (b_1, b_2, \ldots, b_k) \in \mathbb{R}^k$ *such that* $0 < b_1 < b_2 < \cdots < b_k < 1$ *and* $k \geq 2$, $f_b(x)$ *is a density function and* $\Phi_b(t)$ *is its characteristic function.*

Proof. Using Proposition 3, one can show that $\lim_{t\to 0} \Phi_b(t) = 1$ by a direct computation. Therefore, it suffices to show that $\int_{-\infty}^{+\infty} f_b(x) e^{itx} dx = \Phi_b(t)$ for every $t \in \mathbb{R}$. We prove this equality by using Proposition 3 again. Set $I(t) = \int_{-\infty}^{\infty} f_b(x) e^{itx} dx$ and $c_j = \prod_{i \neq j} (b_j - b_i)^{-1}$ for every $j = 1, \ldots, k$. Observe that

$$I(t) = (k-1) \sum_{r=2}^{k} \sum_{j=r}^{k} c_j \int_{b_{r-1}}^{b_r} (b_j - x)^{k-2} e^{itx} dx \, .$$

Integrating by parts one can verify that for $t \neq 0$ the function $e^{itx}(c-x)^p$ admits the antiderivative

$$\frac{e^{itx}}{it} \sum_{s=0}^{p} \frac{p!\,(c-x)^{p-s}}{(p-s)!\,(it)^s}.$$

Hence we can write $I(t) = \sum_{r=2}^{k} \sum_{j=r}^{k} c_j (A_{r,j} - A_{r-1,j})$ where

$$A_{r,j} = e^{itb_r} \sum_{s=0}^{k-2} \frac{(k-1)!\,(b_j - b_r)^{k-2-s}}{(k-2-s)!\,(it)^{s+1}} \quad \text{and in particular} \quad A_{r,r} = \frac{(k-1)!}{(it)^{k-1}}\, e^{itb_r}.$$

Now set $B_r = \sum_{j=r}^{k} c_j A_{r,j}$ and $C_r = \sum_{j=r}^{k} c_j A_{r-1,j}$. For each $2 \leq r \leq k-1$ we have $B_r - C_{r+1} = c_r A_{r,r}$ and moreover $B_k = c_k A_{k,k}$. Finally, by Proposition 3

we have $C_2 = \sum_{j=1}^{k} c_j A_{1,j} - c_1 A_{1,1} = -c_1 A_{1,1}$. As a consequence we get the result, since the integral can be computed as follows

$$I(t) = \sum_{r=2}^{k} (B_r - C_r) = \sum_{j=1}^{k} c_j A_{j,j} = \frac{(k-1)!}{(it)^{k-1}} \sum_{j=1}^{k} c_j e^{itb_j} = \Phi_b(t) \; . \qquad \square$$

5 Further Developments

The analysis presented in the previous section can be extended to all simple models, also when the mean constants β_j's (associated with the dominant components of the main chain) are partially or totally coincident. The limit distributions of our statistics in this more general case are defined extending the notion of polynomial density function given in (9) by allowing multiplicities in the associated tuple b and proving an analogue of Proposition 3 for convolutions with multiplicities.

To state these results precisely we only have to introduce the following characteristic function. Let $b = (b_1, b_2, \ldots, b_r)$ be a tuple of $r \geq 2$ distinct real numbers lying in the interval $(0, 1)$ and let $m = (m_1, m_2, \ldots, m_r) \in \mathbb{N}^r$ be a tuple of multiplicities, where $m_j \geq 1$ for each j and $m_1 + \ldots + m_r = k$. Then define the function

$$\Phi_{b,m}(t) = (k-1)! \sum_{j=1}^{r} \sum_{s=1}^{m_j} c_{j,s} \cdot \frac{e^{itb_j}}{(it)^{k-s}(s-1!)}$$

$$\text{where} \quad c_{j,s} = (-1)^{m_j - s} \sum_{\sum_{\ell \neq j} h_\ell = m_j - s} \; \prod_{\ell \neq j} \binom{m_\ell + h_\ell - 1}{m_\ell - 1} \cdot \frac{1}{(b_j - b_\ell)^{m_\ell + h_\ell}} \; .$$

One can prove that this is a characteristic function and the corresponding density function can be obtained from (9) by a continuity argument. The main difference is that the new density may be non-continuous at the points $x = b_j$ such that $m_j > 1$, $j = 1, \ldots, k$.

Now, let Y_n be defined in a simple model having main chain κ of order k. Let $\beta_1, \ldots, \beta_k$ and $\gamma_1, \ldots, \gamma_k$ be, respectively, the mean and variance constants of the dominant components in κ. We also denote by β and γ the tuples of distinct β_j's and γ_j's in increasing order and by u and v the tuples of the corresponding multiplicities. Clearly, if $\beta_1, \ldots, \beta_k$ are pairwise distinct then Theorem 2 applies. Otherwise we have the following cases:

- If $\beta_1, \ldots, \beta_k$ are partially but not totally coincident (i.e. $\beta_i = \beta_j$ and $\beta_s \neq \beta_t$ for some indices i, j, s, t, $i \neq j$), then Y_n/n converges in distribution to a random variable of characteristic function $\Phi_{\beta,u}(t)$;
- If $\beta_j = \beta_1$ for all $j = 2, \ldots, k$ and all γ_j's are pairwise distinct, then $\frac{Y_n - \beta_1 n}{\sqrt{n}}$ converges in distribution to a random variable of characteristic function $\Phi_\gamma(-t^2/(2i))$;

- If $\beta_j = \beta_1$ for all $j = 2, \ldots, k$ and $\gamma_1, \ldots, \gamma_k$ are partially but not totally coincident, then $\frac{Y_n - \beta_1 n}{\sqrt{n}}$ converges in distribution to a random variable of characteristic function $\Phi_{\gamma,v}(-t^2/(2i))$;
- If $\beta_j = \beta_1$ and $\gamma_j = \gamma_1$ for all $j = 2, \ldots, k$, then $\frac{Y_n - \beta_1 n}{\sqrt{\gamma_1 n}}$ converges in distribution to a normal random variable of mean 0 and variance 1.

The previous results can be further extended by a standard conditioning argument (already used in [4]) to all rational models (ξ, μ, η) whose main chains are "simple", i.e. for every $\kappa \in \Gamma_m$ all dominant components in κ are primitive and non-degenerate. In this case, by equation (8), for every $\kappa \in \Gamma_m$ one can easily see that

$$r_n^{(\kappa)}(z) = s_\kappa(z)\lambda^n n^{k-1} + \mathrm{O}(\lambda^n n^{k-2})$$

where k is the degree of κ and $s_\kappa(z)$ is a nonnull analytic function at $z = 0$. Then, by Theorem 1, we have

$$r_n(0) = R\lambda^n n^{k-1} + \mathrm{O}(\lambda^n n^{k-2})$$

where $R = \sum_{\kappa \in \Gamma_m} s_\kappa(0)$. We can also associate each $\kappa \in \Gamma_m$ with the probability value p_κ, given by $p_\kappa = s_\kappa(0)/R$. Note that the values $\{p_\kappa\}_{\kappa \in \Gamma_m}$ define a discrete probability measure and they can be explicitly computed from the triple (ξ, μ, η).

Moreover, each $\kappa \in \Gamma_m$ defines a simple rational model in its own right, with an associate sequence of random variables $\{Y_n{}^{(\kappa)}\}$ having its own limit distribution according to Theorem 2 and list items above. In particular, $Y_n{}^{(\kappa)}/n$ always converges in distribution to a random variable of distribution function $F_\kappa(x)$ defined according to the previous results. Note that if all constants β_j's are here equal, then $F_\kappa(x)$ reduces to the degenerate distribution of mass point β_1. Now it is not difficult to see that the overall statistics Y_n/n converges in distribution to a r.v. of distribution function $F(x)$ defined by $F(x) = \sum_{\kappa \in \Gamma_m} F_\kappa(x) p_\kappa$.

References

1. J. Berstel and C. Reutenauer. *Rational series and their languages*, Springer-Verlag, New York - Heidelberg - Berlin, 1988.
2. A. Bertoni, C. Choffrut, M. Goldwurm, and V. Lonati. On the number of occurrences of a symbol in words of regular languages. *Theoret. Comput. Sci.*, 302(1-3):431–456, 2003.
3. J. Bourdon and B. Vallée. Generalized pattern matching statistics. *Mathematics and computer science II: algorithms, trees, combinatorics and probabilities.* Proc. of Versailles Colloquium, Birkhauser, 249–265, 2002.
4. D. de Falco, M. Goldwurm, V. Lonati, Frequency of symbol occurrences in bicomponent stochastic models. *Theoret. Comput. Sci.*, 327 (3):269–300, 2004.
5. I. Fudos, E. Pitoura and W. Szpankowski. On pattern occurrences in a random text. *Inform. Process. Lett.*, 57:307–312, 1996.
6. M. S. Gelfand. Prediction of function in DNA sequence analysis. *J. Comput. Biol.*, 2:87–117, 1995.
7. B.V. Gnedenko. *The theory of probability* (translated by G. Yankovsky). Mir Publishers - Moscow, 1976.

8. M. Goldwurm, Probabilistic estimation of the number of prefixes of a trace, *Theoret. Comp. Sci.*, 92:249–268, 1992.
9. P. Grabner and M. Rigo. Additive functions with respect to numeration systems on regular languages. *Monatshefte für Mathematik*, 139: 205–219, 2003.
10. L. J. Guibas and A. M. Odlyzko. Maximal prefix-synchronized codes. *SIAM J. Appl. Math.*, 35:401–418, 1978.
11. L. J. Guibas and A. M. Odlyzko. String overlaps, pattern matching, and nontransitive games. *Journal of Combinatorial Theory. Series A*, 30(2):183–208, 1981.
12. P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. *Proceedings MFCS'91*, Lect. Notes in Comput. Sci., vol. n. 520, Springer, 1991, 248–248.
13. P. Nicodeme, B. Salvy, and P. Flajolet. Motif statistics. *Theoret. Comput. Sci.*, 287(2):593–617, 2002.
14. B. Prum, F. Rudolphe and E. Turckheim. Finding words with unexpected frequencies in deoxyribonucleic acid sequence. *J. Roy. Statist. Soc. Ser. B*, 57:205–220, 1995.
15. M. Régnier and W. Szpankowski. On pattern frequency occurrences in a Markovian sequence. *Algorithmica*, 22 (4):621–649, 1998.
16. A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*, Springer-Verlag, 1978.
17. E. Seneta. *Non-negative matrices and Markov chains*, Springer–Verlag, New York Heidelberg Berlin, 1981.
18. E. Ukkonen. Approximate string-matching with q-grams and maximal matchings. *Theoret. Comput. Sci.*, 92: 191–211, 1992.

Automatic Presentations for Finitely Generated Groups

Graham P. Oliver and Richard M. Thomas

Department of Computer Science,
University of Leicester, Leicester LE1 7RH, UK

Abstract. A structure is said to be computable if its domain can be represented by a set which is accepted by a Turing machine and if its relations can then be checked using Turing machines. Restricting the Turing machines in this definition to finite automata gives us a class of structures with a particularly simple computational structure; these structures are said to have *automatic presentations*. Given their nice algorithmic properties, these have been of interest in a wide variety of areas.

An area of particular interest has been the classification of automatic structures. One of the prime examples under consideration has been the class of groups. We give a complete characterization in the case of finitely generated groups and show that such a group has an automatic presentation if and only if it is virtually abelian.

1 Introduction

In this paper we will be concerned with "structures"; we first explain what this means. A *structure* $\mathcal{A} = (A, R_1, \ldots R_n)$ consists of:

- a set A, called the *domain* (or *universe*) of $\mathcal{A}$;
- for each i with $1 \leqslant i \leqslant n$, there exists $r_i \geqslant 1$ such that R_i is a subset of A^{r_i}; r_i is called the *arity* of R_i.

For example, a group can be viewed as a structure $(G,\ \circ,\ e,\ ^{-1})$, where $\circ$ has arity 3, e has arity 1, and $^{-1}$ has arity 2. A natural area of research is to consider which structures are computable. Taking the Turing machine as our computational paradigm, a computable structure is one for which there exist Turing machines which 'check' the relations in the structure. More formally, a structure $\mathcal{A} = (A, R_1, \ldots R_n)$ is said to be *computable* if:

- the domain A of $\mathcal{A}$ is recognized by a Turing machine;
- for each relation R_i in $\mathcal{A}$, there is a decision-making Turing machine that, on input $(a_1, \ldots, a_{r_i})$, outputs true if $(a_1, \ldots, a_{r_i}) \in R_i$ and false otherwise.

When we say that A is recognized by a Turing machine, we mean that there is a set of symbols I such that A is a recursively enumerable subset of I^*. In fact, when we consider automatic presentations (see Section 2), we allow a mapping from a subset of I^* onto A; in this case we will also need an automaton to check

V. Diekert and B. Durand (Eds.): STACS 2005, LNCS 3404, pp. 693–704, 2005.

when two words in I^* represent the same element of A. In general, the way that elements of A are represented in I^* is clearly important.

Khoussainov and Nerode have introduced [12] a very interesting restriction of that general idea, to *automatic structures*, i.e. those structures whose domain and relations can be checked by finite automata as opposed to Turing machines. A structure isomorphic to an automatic structure is said to have an *automatic presentation*. Given their nice algorithmic properties and the diversity of natural examples of such structures, these have been of interest in a variety of areas.

The general idea of using finite automata to read structures is not entirely new; for example, in group theory, a group is said to be *automatic* if, when we code elements of the group as strings of generators, there is a regular subset L of the set of all strings of generators such that there are finite automata to check multiplication of words in L by generators. This concept was introduced in [6], motivated by work in hyperbolic manifolds as well as a general interest in computing with groups. The considerable success of the theory of automatic groups gives one motivation to have a general notion of automatic structures. See also [17, 20].

However, there are other motivations for a general study of automatic structures, most importantly, perhaps, the decidability properties that come with finite automata. In particular, the first-order theory of an automatic structure is decidable. Another motivation for the study of automatic presentations is that of extending some of the techniques of finite model theory to infinite structures that have finite presentations; see [2, 3] for example.

One interesting result presented in [12] is that all finitely generated abelian groups have automatic presentations. Some natural questions follow from this:

- How far can this be extended - are there other (finitely generated) groups with automatic presentations?
- What are the necessary or sufficient conditions for a (finitely generated) group to have an automatic presentation?
- How does the class of finitely generated groups with automatic presentations compare with that of automatic groups?

These questions are the main incentive for the work presented here. Note that, while an automatic group is finitely generated (this is essentially part of the definition), a group with an automatic presentation need not be finitely generated (such as the Prüfer group $\mathbb{Q}_p/\mathbb{Z}$; see [13]). However, we will only be concerned with finitely generated groups here and we give a complete answer to these questions in that case (see Theorems 8 and 9 below). In particular, we show that a finitely generated group has an automatic presentation if and only if it is virtually abelian, and hence that a finitely generated group with an automatic presentation is necessarily automatic (but the converse is false).

Classifying structures with automatic presentations is clearly of significant interest and we see this characterization of finitely generated groups with such presentations as part of this general programme. There are some other important classifications such as that of the ordinals with automatic presentations in [5] and the Boolean algebras with automatic presentations in [13].

2 Automatic Presentations

Before we present the definition of an automatic presentation, we need to introduce the idea of a "convolution" (see [12] for example).

If I is an alphabet, we define the *convolution* of $(x_1,\ x_2,\ldots,\ x_n) \in (I^*)^n$, where $x_i = x_i^1 x_i^2 \ldots x_i^{p_i}$ $(x_i^j \in I)$, to be

$$\mathrm{conv}(x_1,\ldots,\ x_n) = (\bar{x}_1^1,\ \bar{x}_2^1,\ldots,\ \bar{x}_n^1)(\bar{x}_1^2,\ \bar{x}_2^2,\ldots,\ \bar{x}_n^2)\ldots(\bar{x}_1^p,\ \bar{x}_2^p,\ldots,\ \bar{x}_n^p),$$

where $p = \max\{p_i : 1 \leqslant i \leqslant n\}$ and, for some $\Box \notin I$,

$$\bar{x}_i^j = \begin{cases} x_i^j & 1 \leqslant j \leqslant p_i \\ \Box & p_i < j \leqslant p \end{cases}.$$

These elements $\mathrm{conv}(x_1,\ldots,\ x_n)$ are words over the alphabet

$$I_\Box^n = ((I \cup \{\Box\}) \times (I \cup \{\Box\}) \times \ldots \times (I \cup \{\Box\})) \setminus \{(\Box, \Box, \ldots, \Box)\}.$$

The symbol $\Box$ is a *padding symbol* which is not in the original alphabet I. We now have our definition of an automatic presentation:

Definition 1. *A structure $\mathcal{A} = (A, R_1, \ldots R_n)$ has an automatic presentation (over an alphabet I) if*

1. *there is a language L over I and a map $c : L \to A$ such that c is surjective;*
2. *L is accepted by a finite automaton over I;*
3. *$L_= = \{conv(x,\ y) : c(x) = c(y),\ x, y \in L\}$ is accepted by a finite automaton over $I_\Box^2$;*
4. *for each relation R_i in $\mathcal{A}$, the language*

$$L_{R_i} = \{conv(x_1, \ldots, x_{r_i}) : (c(x_1), \ldots, c(x_{r_i})) \in R_i\}$$

is accepted by a finite automaton over $I_\Box^{r_i}$.

The tuple $(I,\ L,\ c,\ L_=,\ (L_{R_i})_{1 \leqslant i \leqslant n})$ is called an *automatic presentation* for $\mathcal{A}$. The presentation is called *injective* if c is injective and *binary* if $|I| = 2$. It is clear that all structures with a finite domain have an automatic presentation (as finite languages are regular); on the other hand, we see that the domain of such a structure must be countable. These facts will be implicitly assumed throughout.

3 Properties

In this section we list some properties of structures with automatic presentations. We start with two useful facts from [2] and [12]:

Proposition 1. *Let $\mathcal{A}$ be a structure with an automatic presentation; then:*

1. *$\mathcal{A}$ has a binary automatic presentation.*
2. *$\mathcal{A}$ has an injective automatic presentation.*

Remark 1. The proof that, if we have an automatic presentation

$$(I,\ L,\ c,\ L_=,\ (L_{R_i})_{1\leqslant i\leqslant n}),$$

then we have an injective automatic presentation $(J,\ K,\ c',\ K_=,\ (K_{R_i})_{1\leqslant i\leqslant n})$, uses the same alphabet and constructs a subset K of L; so we may put the two parts of Proposition 1 together and say that every structure with an automatic presentation has an injective binary automatic presentation. □

A common (and useful) way of working with finite automata is to use predicate calculus. This arises from the closure of regular languages under (finite) union and intersection, complementation, and the definability of the existential and universal quantifiers. As such, if R_1 and R_2 are relations recognised by finite automata, then $R_1 \wedge R_2$, $R_1 \vee R_2$, $\neg R_1$, $\exists x(R_1)$ and $\forall x(R_1)$ are also all recognised by finite automata; see [6] for details. The proofs are all constructive, which gives the following results from [12]:

Theorem 1. *Let $\mathcal{A}$ be a structure with an automatic presentation; then:*

1. *if P is a first-order definable relation on $\mathcal{A}$ then P is decidable;*
2. *the first-order theory of $\mathcal{A}$ is decidable.*

Remark 2. For future reference we note that the point about the proof of Theorem 1 is that, if $\mathcal{A} = (A,\ R_1,\ \ldots,\ R_n)$ is a structure with an automatic presentation, then any relation S built from the R_i using first-order constructs is also recognizable by a finite automaton; as a result, we would also have an automatic presentation for the structure $\mathcal{B} = (A,\ R_1, \ldots,\ R_n,\ S)$. □

Theorem 1 has been extended further. Let $\exists^\infty$ be the quantifier "there exist infinitely many" and let FO denote first-order logic; we then have [2, 3]:

Theorem 2. *Let $\mathcal{A}$ be a structure with an automatic presentation; then the FO($\exists^\infty$) theory of $\mathcal{A}$ is decidable.*

For more information about decidability results (and model-theoretic results in general) see [2].

We now come to "interpretations". Let $\mathcal{A}$ and $\mathcal{B}$ be structures; then an (n-dimensional) *interpretation* I of $\mathcal{B}$ in $\mathcal{A}$ consists of:

- a formula $\delta_I(x_1, \ldots, x_n)$ in $\mathcal{A}$; this is called the *domain formula* of I;
- for each unnested atomic formula $\phi(y_1, \ldots, y_m)$ of $\mathcal{B}$, a formula $\phi_I(\bar{x}_1, \ldots, \bar{x}_m)$ of $\mathcal{A}$, where the $\bar{x}_i$ are disjoint n-tuples of distinct variables; these are called the *defining formulae* of I;
- a surjective map $f_I : \delta_I(A^n) \to B$; this is called the *coordinate map* of I.

In addition we insist that, for all unnested atomic formulae of $\mathcal{A}$ and all $\bar{a}_i \in \delta_I(A^n)$, we have:

$$B \models \phi(f_I(\bar{a}_1), \ldots, f_I(\bar{a}_m)) \ \Leftrightarrow\ A \models \phi_I(\bar{a}_1, \ldots, \bar{a}_m).$$

If there is an interpretation of $\mathcal{B}$ in $\mathcal{A}$ where all the formulae are first-order then we say that $\mathcal{B}$ is *interpretable* in $\mathcal{A}$ and that $\mathcal{B}$ is a *subinterpretation* of $\mathcal{A}$. If we fix a particular logic L, then we get L-interpretations. See [11] for more details.

The following model-theoretic result [2, 3], as well as being useful in its own right, has many interesting consequences:

Proposition 2. *Let $\mathcal{A}$ be a structure and $\mathcal{B}$ be a structure with an automatic presentation; if $\mathcal{A}$ is $FO(\exists^\omega)$-interpretable in $\mathcal{B}$, then $\mathcal{A}$ has an automatic presentation.*

4 Results on Groups

Before presenting our results on groups with automatic presentations, it is worth commenting on the form of the structure for groups.

We asserted above that groups could be considered as having structure $(G, \circ, e, ^{-1})$. However, it is common to view groups just as having a single binary operation, and so we would have a structure $(G, \circ)$. Which is correct?

In a sense, the answer depends on how you are considering the structures. As noted in [11], the main difference is that of substructures: the substructures of groups as structures $(G, \circ)$ need only be subsemigroups, whereas, with $(G, \circ, e, ^{-1})$, they must be subgroups. For our purposes, we need not be too worried by this distinction. It is clear that, for the structure $(G, \circ)$, the properties of having an identity and having inverses are both first-order definable; so, if a group as a structure $(G, \circ)$ has an automatic presentation, then (as in Remark 2) this same presentation may be expanded to one for the structure $(G, \circ, e, ^{-1})$. With this in mind, we need only concentrate on $(G, \circ)$ in what follows, and we now have:

Definition 2. *A group $(G, \circ)$ is said to have an automatic presentation (over an alphabet I) if*

1. *there is a language L over I and a map $c : L \to G$ such that c is surjective;*
2. *L is accepted by a finite automaton over I;*
3. *$\{conv(x, y) : c(x) = c(y)\}$ is accepted by a finite automaton over $I^2_\square$;*
4. *$\{conv(x_1, x_2, x_3) : c(x_1) \circ c(x_2) = c(x_3)\}$ is accepted by a finite automaton over $I^3_\square$.*

The following result from [12] sums up much of what is already known concerning finitely generated groups with automatic presentations:

Proposition 3. *Finitely generated abelian groups have automatic presentations.*

We now need another definition. Let χ be a group property (such as being abelian); then a group G is said to be *virtually* (or *almost*) χ if G contains a subgroup of finite index with the property χ. We then have:

Theorem 3. *Finitely generated virtually abelian groups have automatic presentations.*

Proof. Let G be a finitely generated group with an abelian subgroup A of finite index; then G is FO-interpretable in A (see [1] for example). The result follows from Propositions 2 and 3. □

Remark 3. We make a note before continuing. Suppose that G is a finitely generated virtually abelian group, so that G has an abelian subgroup A of finite index. Then A is finitely generated and hence is a direct product $C_1 \times C_2 \times \ldots C_k$ of cyclic groups. If we consider the subgroup B of A generated by the infinite groups C_i (i.e. ignore the C_i which are finite cyclic groups), then B has finite index in A, and hence has finite index in G.

Now B is a free abelian group isomorphic to $\mathbb{Z}^n = \mathbb{Z} \times \mathbb{Z} \times \ldots \times \mathbb{Z}$ for some n; so every finitely generated virtually abelian group has a free abelian subgroup of finite index. Moreover, if H is a subgroup of finite index in a group G, then there is a normal subgroup N of G contained in H with N also of finite index in G; see Proposition 3.35 of [21] for example (or, for a proof couched in terms of finite automata, Proposition 1.5 of [10]). As a subgroup of a free abelian group is free abelian, we have that every finitely generated virtually abelian group has a normal free abelian subgroup of finite index. □

Remark 4. It is also possible to prove Theorem 3 from first principles by constructing appropriate automata. We give an outline of the proof here.

Let G be a finitely generated virtually abelian group. As in Remark 3, let $A = \langle x_1, x_2, \ldots, x_n \rangle$ be a normal subgroup of G of finite index isomorphic to $\mathbb{Z}^n$ and then let $T = \{t_1, t_2, \ldots, t_k\}$ be a set of coset representatives for A in G. The coding of the elements of G is fairly straightforward: we have a symbol t_j for the coset representative, a symbol for an n-tuple of $+$'s and $-$'s, and then symbols for n-tuples of 1's and 0's.

The point is that any element g of G can be expressed in the form $t_j a$ with $a \in A$, and then a can be written in the form $x_1^{\epsilon_1 m_1} x_2^{\epsilon_2 m_2} \ldots x_n^{\epsilon_n m_n}$ with $\epsilon_i \in \{1, -1\}$ and $m_i \in \mathbb{N}$ (if $m_i = 0$ we take $\epsilon_i = 1$). We then represent g as $t_j\,(\epsilon_1, \epsilon_2, \ldots, \epsilon_n)\,\mathrm{conv}(\overline{m_1}, \overline{m_2}, \ldots, \overline{m_n})$, where $\overline{m_i}$ is the representation of m_i in reverse order binary notation. Since A is normal in G, each $x_i t_j$ is of the form $t_j x_1^{u_{1,i,j}} x_2^{u_{2,i,j}} \ldots x_n^{u_{n,i,j}}$ for some $u_{h,i,j} \in \mathbb{Z}$; so multiplication in G is given by

$$t_i x_1^{a_1} \ldots x_n^{a_n} . t_j x_1^{b_1} \ldots x_n^{b_n} = t_i t_j x_1^{a'_1 + b_1} \ldots x_n^{a'_n + b_n}$$

where $a'_i = \sum_{k=1}^{n} a_k u_{i,k,j}$. Now let t_k and $c_1, c_2, \ldots, c_n$ be such that $t_i t_j = t_k x_1^{c_1} \ldots x_n^{c_n}$; then

$$t_i t_j x_1^{a'_1 + b_1} \ldots x_n^{a'_n + b_n} = t_k x_1^{a'_1 + b_1 + c_1} \ldots x_n^{a'_n + b_n + c_n}.$$

Given all this, we first create different transitions in our automaton for each possible pair of t_i's, and then, from these different transitions, for each possible combination of $+$ and $-$s. Then, based on the binary addition of n-tuples and

taking into account the $u_{1,i,j}$ and c_i, we construct the rest of the automaton. The states, roughly, represent the current value of the carry in the addition. As the total amount carried at each stage is bounded by $n-1$ we have a finite automaton. □

5 Growth

Currently there are not many techniques for showing that a structure does not have an automatic presentation. One such method follows from Theorem 2: if a structure has undecidable $\mathrm{FO}(\exists^\omega)$ theory then it does not have an automatic presentation; for further conditions see [13, 15, 19] for example.

Another important method involves "growth". Let $\mathcal{A}$ be a structure with domain A and an automatic presentation, and choose an injective automatic presentation for $\mathcal{A}$; then, for $x \in A$, let $l(x)$ denote the length of the coding of x in this presentation. We have the following result from [2]:

Theorem 4. *Let $f : A^n \to A$ be a first-order definable function on $\mathcal{A}$; then there exists a constant N such that*

$$\forall \overline{a} \in A^n,\ l(f(\overline{a})) \leqslant max\{l(a_0), \ldots, l(a_{n-1})\} + N.$$

In particular, this result has the following consequence for groups:

Corollary 1. *Let G be a group with an injective automatic presentation; then there is a constant N such that, for all $g_0, g_1, g_2 \in G$:*

$$g_0 g_1 = g_2 \ \Rightarrow\ l(g_2) \leqslant max\{l(g_0), l(g_1)\} + N.$$

There is a corresponding notion of growth in group theory. Let G be a group with a finite generating set Δ, and assume that Δ is closed under taking inverses. (To be precise, when we say "generating set", we are assuming that the set generates G as a semigroup, although this distinction will not be particularly significant in this paper.) Now let $\delta(g)$ be the minimum $n \in \mathbb{N}$ such that $g = a_1 a_2 \ldots a_n$ ($a_i \in \Delta$). The *growth function* of G is then defined to be

$$\gamma(n) = |\{g \in G : \delta(g) \leqslant n\}|.$$

The nature of this function (in the sense of its being bounded above by a polynomial function, below by an exponential function, or neither of these), is independent of which particular finite generating set we choose. As such, the nature of the growth function (in this sense) is a property solely of the group (as opposed to a property of the group together with a choice of finite generating set). In the three cases we have mentioned, the group is said to have (respectively) *polynomial growth*, *exponential growth* or *intermediate growth*; see [8] for a survey on growth in groups. We now prove the following result:

Theorem 5. *If a group G has an automatic presentation then G has polynomial growth.*

Before we do this, we first prove a useful proposition:

Proposition 4. *With notation as above, let* $R = max\{l(a) : a \in \Delta\}$*; then there is a constant* N *such that, for all* $m \geqslant 1$*, we have*

$$max\{l(a_1 \dots a_m) : a_i \in \Delta\} \leqslant R + \lceil \log_2 m \rceil N.$$

Proof. Let N be the constant of Corollary 1. We proceed by induction on m. We first consider the case $m = 1$. Here we clearly have

$$\max\{l(a_1) : a_1 \in \Delta\} = R = R + \lceil \log_2 1 \rceil N.$$

Now assume the result holds for $1 \leqslant m \leqslant k$. We split our proof into two cases.

Case one: k is odd, say $k = 2r - 1$. Then, using Corollary 1, we have

$$\begin{aligned}\max\{l(a_1 \dots a_{k+1}) : a_i \in \Delta\} &= \max\{l(a_1 \dots a_{2r}) : a_i \in \Delta\} \\ &\leqslant \max\{l(a_1 \dots a_r),\ l(a_{r+1} \dots a_{2r}) : a_i \in \Delta\} + N \\ &\leqslant \max\{R + \lceil \log_2 r \rceil N,\ R + \lceil \log_2 r \rceil N\} + N \\ &= R + \lceil \log_2 (k+1) \rceil N\end{aligned}$$

as required.

Case two: k is even, say $k = 2r$. This time we have

$$\begin{aligned}\max\{l(a_1 \dots a_{k+1}) : a_i \in \Delta\} &= \max\{l(a_1 \dots a_{2r+1}) : a_i \in \Delta\} \\ &\leqslant \max\{l(a_1 \dots a_r),\ l(a_{r+1} \dots a_{2r+1}) : a_i \in \Delta\} + N \\ &\leqslant \max\{R + \lceil \log_2 r \rceil N,\ R + \lceil \log_2 (r+1) \rceil N\} + N \\ &= R + \lceil \log_2 (r+1) \rceil N + N.\end{aligned}$$

We split our consideration of this case into two subcases.

Subcase one: r is not of the form 2^x with $x \geqslant 1$.

If $r \in \mathbb{N}$, $r > 0$ and $r \neq 2^x$, then $\lceil \log_2 (r+1) \rceil = \lceil \log_2 r \rceil$. This gives

$$\begin{aligned}&R + \lceil \log_2 (r+1) \rceil N + N = R + \lceil \log_2 r \rceil N + N = R + \lceil \log_2 2r \rceil N \\ &= R + \lceil \log_2 (2r+1) \rceil N = R + \lceil \log_2 (k+1) \rceil N.\end{aligned}$$

Subcase two: $r = 2^x$ $(x \geqslant 1)$.

Note first that $\lceil \log_2 (k+1) \rceil = \lceil \log_2 (2r+1) \rceil = \lceil \log_2 (2^{x+1} + 1) \rceil = x + 2$. Now

$$R + \lceil \log_2 (r+1) \rceil N + N = R + \lceil \log_2 2(r+1) \rceil N = R + (x+2)N = R + \lceil \log_2 (k+1) \rceil N$$

as required. □

Given Proposition 4, we can now prove Theorem 5:

Proof. By Remark 1 we may assume that the presentation for G is injective and binary. Then, as

$$\max\{l(a_1 \dots a_m) : a_i \in \Delta\} \leqslant R + \lceil \log_2 m \rceil N$$

by Proposition 4, the number of possible codes for words of the form $a_1 \dots a_m$ is

$$2^{R+\lceil \log_2 m \rceil N} \leqslant 2^R (2^{\log_2 m + 1})^N = k m^N$$

where $k = 2^R 2^N$ is a constant. So we have at most km^N possible elements g in G with $\delta(g) = m$; as a result, we have

$$\gamma(n) = |\{g \in G : \delta(g) \leqslant n\}| \leqslant k.1^N + k.2^N + \ldots + k.n^N \leqslant k.n^{N+1}.$$

So G has polynomial growth as required. □

6 Classification

We now quote two substantial known theorems which enable us to give a complete classification as to which finitely generated groups have an automatic presentation (to some extent solving a problem of [14]). We first need some more definitions from group theory; these are standard concepts (see [21] for example).

If G is a group and if H and K are subsets of G, then we let $[H, K]$ denote the set of all elements of G of the form $h^{-1}k^{-1}hk$ with $h \in H$ and $k \in K$. If H and K are subgroups of G, then $[H, K]$ is a subgroup of G and if, in addition, H and K are normal in G, then $[H, K]$ is a normal subgroup of G. We now define the following chains of normal subgroups of G:

$$G^{(0)} = G;\ G^{(1)} = [G, G];\ G^{(2)} = [G^{(1)}, G^{(1)}];\ G^{(3)} = [G^{(2)}, G^{(2)}];\ \ldots$$
$$\gamma_0(G) = G;\ \gamma_1(G) = [\gamma_0(G), G];\ \gamma_2(G) = [\gamma_1(G), G];\ \gamma_3(G) = [\gamma_2(G), G];\ \ldots$$

Note that $G \geqslant G^{(1)} \geqslant G^{(2)} \geqslant \ldots$ and that $G \geqslant \gamma_1(G) \geqslant \gamma_2(G) \geqslant \ldots$. A group G is said to be *solvable* if $G^{(r)} = \{e\}$ for some $r \in \mathbb{N}$ and *nilpotent* if $\gamma_r(G) = \{e\}$ for some $r \in \mathbb{N}$; in the first case we call r the *derived length* of G and, in the second case, r is called the *nilpotency class* of G. In addition, a nilpotent group G is *polycyclic* (i.e. there is a chain of subgroups $G \geqslant G_1 \geqslant G_2 \geqslant \ldots \geqslant G_n = \{e\}$ with each G_{i+1} normal in G_i and each G_i/G_{i+1} cyclic) and a polycyclic group is solvable (though the reverse implications are false).

Given all this, we can now state Gromov's classification [9] of groups with polynomial growth:

Theorem 6. *If a finitely generated group has polynomial growth then it is virtually nilpotent.*

Eršov showed [7] that a nilpotent group has decidable first-order theory if and only if it is virtually abelian. This was generalized by Romanovskii [18] to virtually polycylic groups and then by Noskov [16], who showed that a virtually solvable group has decidable first-order theory if and only if it is virtually abelian. We need the result for virtually nilpotent groups which is a special case of Romanovskii's theorem:

Theorem 7. *Let G be a finitely generated virtually nilpotent group with decidable first-order theory; then G is virtually abelian.*

These two results enable us to prove:

Theorem 8 (Classification). *Let G be a finitely generated group; then G has an automatic presentation if and only if G is virtually abelian.*

Proof. Assume that G has an automatic presentation. By Theorem 5, G has polynomial growth, and so, by Theorem 6, G is virtually nilpotent. By Theorem 1, G has decidable first-order theory, and so, by Theorem 7, G is virtually abelian.

The converse follows from Theorem 3. □

7 Automatic Groups

The theory of automatic groups (see [6] for example) was mentioned in the introduction as being one of the motivations for studying structures with automatic presentations. Naturally the connections between the two notions have been remarked upon elsewhere; see [3] for example. We make some further comments on the relationship between these concepts here.

Given a group $(G,\ \circ)$ with a finite set of generators $X = \{a_1, \ldots, a_n\}$, we form a new structure $\mathcal{G} = (G,\ R_1, \ldots,\ R_n)$ where $R_i(g, h)$ if and only if $g \circ a_i = h$; this new structure is called the *Cayley graph* of G with respect to X. If G is an automatic group then we have an encoding of the elements of G as words in X^* such that there are automata recognizing multiplication by elements of X; we see that this gives an automatic presentation for $\mathcal{G}$.

This connection has been observed before; see [12], for example, and the comments following Proposition 2.11 (which we have quoted as Proposition 3 above) there. The point is that the proof in [6] that a finitely generated abelian group is automatic is constrained (by definition) to using encodings of elements as words in the generators, but only needs to produce automata representing multiplication by generators; on the other hand, the proof in [12] that such a group has an automatic presentation permits a different encoding of the elements but needs an automaton recognizing multiplication of any two elements in the group.

We have a similar issue with automatic groups and automatic presentations for Cayley graphs. As we mentioned above, if G is an automatic group, then we have an automatic presentation for the Cayley graph $\mathcal{G}$ of G where the encodings of the elements are again words in the generators of G; however, an automatic presentation for $\mathcal{G}$ need not use such an encoding.

This distinction is significant. For example, let H be the *Heisenberg group*, i.e. the group of matrices

$$\left\{ \begin{pmatrix} 1 & x & z \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix} : x, y, z \in \mathbb{Z} \right\}.$$

It is noted in [3] that the Cayley graph of H has an automatic presentation, but that H is not an automatic group. As H is not virtually abelian, it also does not have an automatic presentation (as a group) by Theorem 8.

We also note that the choice of generating set for the group is not significant when considering whether Cayley graphs have automatic presentations:

Proposition 5. *If G is a group with finite generating sets X and Y, then the Cayley graph of G with respect to X has an automatic presentation if and only if the Cayley graph of G with respect to Y has an automatic presentation.*

Proof. If $X = \{x_1, x_2, \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, then we have a sequence of finite generating sets

$$X,\ X \cup \{y_1\},\ X \cup \{y_1, y_2\},\ \ldots,\ X \cup Y,\ \{x_1, x_2, \ldots, x_{m-1}\} \cup Y,$$
$$\{x_1, x_2, \ldots, x_{m-2}\} \cup Y,\ \ldots,\ \{x_1\} \cup Y,\ Y$$

for G. Note that we have only added or deleted one generator at a time.

It is easy to see that deleting a redundant generator does not affect the existence of an automatic presentation for a Cayley graph; we are simply omitting one of the relations in our structure. On the other hand, if we have a generating set $A = \{a_1, \ldots, a_k\}$ and we add a new generator b, then we may express b as a word $a_{i(1)}a_{i(2)} \ldots a_{i(k)}$ in A^+, and the new relation S we have introduced is the composition $R_{i(1)}R_{i(2)} \ldots R_{i(k)}$. Since each of the R_i are recognized by finite automata, the new relation S is recognized by a finite automaton as well. □

Restricting ourselves to finitely generated groups, let `AutoPres` represent the class of groups with automatic presentations, `Automatic` represent the class of automatic groups, and `CayleyAutoPres` represent the class of groups whose Cayley graphs have automatic presentations. We have

Theorem 9. `AutoPres` $\subsetneq$ `Automatic` $\subsetneq$ `CayleyAutoPres`.

Proof. All virtually abelian groups are automatic, but there are plenty of groups (such as free groups) that are automatic but do not have automatic presentations; this gives the first (proper) inclusion. As mentioned above, the automata required for automatic groups give automatic presentations for the Cayley graphs of these groups; however, the Cayley graph of the Heisenberg group has an automatic presentation, but the Heisenberg group is not automatic. This gives the second (proper) inclusion. □

Acknowledgements. The authors would like to thank Bakhadyr Khoussainov for helpful conversations about the material in this paper. The constructive suggestions of the referees were also appreciated. The second author would like to thank Hilary Craig for all her help and encouragement.

References

1. W. Baur, G. Cherlin and A. Macintyre, Totally categorical groups and rings, *J. Algebra* **57** (1979), 407–440.

2. A. Blumensath, *Automatic structures* (Diploma Thesis, University of Aachen, 1999).
3. A. Blumensath and E. Grädel, Finite presentations of infinite structures: automata and interpretations, *in* G. Gottlob, M. Hermann & M. Rusinowitch (eds.), *Proceedings of the 2nd International Workshop on Complexity in Automated Deduction* (2002); to appear in *Theory Comput. Syst.*
4. D. Cenzer and J. Remmel, Polynomial time versus recursive structures, *Annals Pure Appl. Logic* **54** (1991), 17–58.
5. C. Delhommé, V. Goranko and T. Knapik, Automatic linear orderings (preprint, 2003).
6. D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson and W. P. Thurston, *Word Processing in Groups* (Jones and Bartlett, 1992).
7. Ju. L. Eršov, Elementary group theories (Russian), *Dokl. Akad. Nauk SSR* **203** (1972), 1240–1243; English translation *Soviet Math. Dokl.* **13** (1972), 528–532.
8. R. I. Grigorchuk, On growth in group theory, *Proceedings of the International Congress of Mathematicians, Kyoto 1990, Volume I, II* (Math. Soc. Japan, 1991), 325–338.
9. M. Gromov, Groups of polynomial growth and expanding maps, *Publications Mathématique d'IHÉS* **53** (1981), 53–78.
10. T. Herbst and R. M. Thomas, Group presentations, formal languages and characterizations of one-counter groups, *Theoret. Comp. Sci.* **112** (1993), 187–213.
11. W. Hodges, *Model Theory* (Encyclopedia of Mathematics and its Applications **42**, Cambridge University Press, 1993).
12. B. Khoussainov and A. Nerode, Automatic presentations of structures *in* D. Leivant (ed.), *Logic and Computational Complexity* (Lecture Notes in Comp. Sci. **960**, Springer-Verlag, 1995), 367–392.
13. B. Khoussainov, A. Nies, S. Rubin and F. Stephan, Automatic structures: richness and limitations, *Proceedings of the 19th IEEE Symposium on Logic in Computer Science* (IEEE Computer Society, 2004), 110–119.
14. B. Khoussainov and S. Rubin, Some thoughts on automatic structures, *J. Autom. Lang. Comb.* **8** (2003), 287–301.
15. B. Khoussainov, S. Rubin and F. Stephan, On automatic partial orders, *Proceedings of the 18th IEEE Symposium on Logic in Computer Science* (IEEE Computer Society, 2003), 168–177.
16. G. A. Noskov, The elementary theory of a finitely generated almost solvable group (Russian), *Izv. Akad. Nauk SSSR Ser. Mat.* **47** (1983), 498–517; English translation *Math. USSR Izv.* **22** (1984), 465–482.
17. L. Pelecq, *Isomorphismes et automorphismes des graphes context-free, équationnels et automatiques* (PhD Thesis, Bordeaux 1 University, 1997).
18. N. S. Romanovskiĭ, On the elementary theory of an almost polycyclic group (Russian), *Math. Sb.* **111** (1980), 135–143; English translation *Math. USSR Sb.* **39** (1981).
19. S. Rubin, Finite automata and well ordered sets, *in* S. Yeates (ed.), *Third New Zealand Computer Science Research Students' Conference, Hamilton, New Zealand* (University of Waikato, 1999), 86–93.
20. G. Sénizergues, Definability in weak monadic second-order logic of some infinite graphs, *in* K. Compton, J.-E. Pin & W. Thomas (eds), *Automata Theory: Infinite Computations* (Dagstuhl Seminar **9202**, Wadern, Germany, 1992), 16.
21. G. C. Smith and O. M. Tabachnikova, *Topics in Group Theory* (Springer-Verlag, 2000).

Author Index